

VLADIMIR JOVANOVIKJ

DOCTORAL DISSERTATION

**MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL**

VLADIMIR JOVANOVIKJ

**CONTEXT-AWARE SECURITY FOR PROTECTION OF
UBIQUITOUS SOCIAL SYSTEMS**

DOCTORAL DISSERTATION

LJUBLJANA, MAY 2017

CONTEXT-AWARE SECURITY FOR PROTECTION OF UBIQUITOUS SOCIAL SYSTEMS

Vladimir Jovanovikj

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Asst. Prof. Dr. Tomaž Klobučar, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:

Prof. Dr. Borka Jerman Blažič, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Asst. Prof. Dr. Rok Bojanc, Member, ZZI, Ljubljana, Slovenia
Asst. Prof. Dr. Tanja Arh, Member, Jožef Stefan Institute, Ljubljana, Slovenia
Doc. Dr. Tonimir Kišasondi, Member, University of Zagreb, Croatia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL



Vladimir Jovanovikj

CONTEXT-AWARE SECURITY FOR PROTECTION OF UBIQUITOUS SOCIAL SYSTEMS

Doctoral Dissertation

ZAŠČITA VSEPRISOTNIH DRUŽBENIH SISTEMOV S PO-
MOČJO KONTEKSTNO-ODVISNIH VARNOSTNIH REŠITEV

Doktorska disertacija

Supervisor: Asst. Prof. Dr. Tomaž Klobučar

Ljubljana, Slovenia, May 2017

To Matea and Dijana, my home social group

Acknowledgments

First of all, I would like to thank my supervisor, Asst. Prof. Dr. Tomaž Klobučar, and my co-supervisor, Dr. Dušan Gabrijelčič, for their guidance, support, help and patience. Their constructive comments and insightful observations during our endless discussions have been of invaluable importance to this work, which could never have been carried out without them. At the same time, their expertise and rigor have improved my research skills, made me a better analytical thinker and sharpened my sense of computer security.

I am very grateful to Prof. Dr. Borka Jerman Blažič, who accepted me as a young researcher at the Laboratory for Open Systems and Networks, and monitored my progress during the study.

Thanks are also due to my colleagues from the Laboratory for Open Systems and Networks for maintaining a positive and friendly working environment in which I have been able to complete my PhD journey. In particular, I will never forget the cigarette breaks with Dušan, which we continued to take even after we both quit smoking.

I would especially like to thank the Slovenian Research Agency (ARRS) and the Laboratory for Open Systems and Networks, who financially supported me during my studies and enabled me to carry out the research involved in this dissertation.

I thank my wife Dijana with all my heart for her unconditional support, understanding and sacrifice throughout these years. As my best friend, she believed in me despite all the hard times I experienced during my PhD journey. Dijana, you are my muse and I love you so much.

Last but not least, I would like to thank my parents, Anica and Dragan, my sister Vesna, my aunt Elena, my in-laws Pavlina and Petko, and my brother-in-law Dejan, who were always there when I needed them, sincerely celebrating my achievements.

Abstract

Ubiquitous environments which embrace the trends of enterprise mobility and consumerization of IT have an increase in social importance. In these environments, the same mobile device and applications are simultaneously used for both personal and professional purposes. This type of usage blurs the boundaries between these domains and presents numerous challenges for information security. Context-aware security has been proposed as a solution for many of these; however, it is argued here that the existing context-aware security systems are limited, and deal mainly with targeted use cases. They do not provide a clear and complete understanding of the context relevant to security, and use contextual information with an arbitrary level of abstraction. Moreover, they utilize security policies which are burdened with context reasoning rules and policy enforcement actions. These policies are specified according to policy models, which consider context only as an add-on, and do not provide a means of modeling adaptable security behavior.

In order to address these issues, a new conceptual model of security context is proposed. The model identifies the important concepts of security context and also takes into account related social aspects. It represents the security context using a set of concepts at the appropriate level of abstraction. Based on this model, a security policy model and language are also proposed. These are centered around the notion of the security context, and enable the specification and analysis of simple but expressive security policies which are applicable to ubiquitous social systems. To achieve adaptable behavior, the policy model further introduces an object lifecycle diagram. In addition, a range of security policy templates are defined which capture recurring requirements in a diverse set of use cases.

The models and language are evaluated in an empirical and practical manner. First, it is shown that the conceptual model is able to represent the security context covered by other approaches. Then, the policy model is used to identify the possible threats to the assets of interest during various activities, together with their potential consequences. Next, both the models and the language are applied in order to analyze diverse ubiquitous social system use cases from the perspective of security. Finally, the feasibility of this approach is demonstrated by implementing a proof-of-concept prototype of a context-aware security system, which enables the specification and enforcement of security policies according to the policy model and language presented here.

The proposed conceptual model describes the security context more clearly than the alternatives, and improves the current understanding by introducing social concepts as abstractions. Furthermore, the new policy model provides more intuitive abstractions for ubiquitous social systems and enables a clear and precise analysis of the way in which important contextual information influence access control. Together with the policy language and templates, this facilitates the specification, management and reuse of security policies for ubiquitous security systems. Overall, this thesis provides a comprehensive view of ubiquitous social systems and their security challenges, analyzing them systematically from the perspective of security. In addition, it facilitates context specification and usage, making the design and development of context-aware security systems more straightforward.

Povzetek

Vseprisotni sistemi, ki zajemajo koncepte mobilnosti zaposlenih, opravljanja dela na različnih lokacijah ter vključevanja potrošniških informacijskih aplikacij in naprav v poslovanje podjetij, imajo vse večji družbeni pomen. V takšnem okolju uporabljamo iste mobilne naprave in aplikacije v zasebne in službene namene. Uporaba istih naprav in aplikacij v različne namene briše mejo med zasebnim življenjem in delovnim okoljem, hkrati pa prinaša številne varnostne izzive. Pri reševanju mnogih od njih si lahko pomagamo s kontekstno varnostjo, ki predvideva prilagajanje varnostnih storitev in mehanizmov varnostnemu kontekstu. Žal so obstoječi kontekstni varnostni sistemi le ozko uporabni, večinoma omejeni na posamezne primere. Obstoječi sistemi ne ponujajo jasne in popolne slike o kontekstu, pomembnem za zagotovitev varnosti, kontekstualne podatke pa uporabljajo na poljubni ravni abstrakcije. Nadalje so varnostne politike v takšnih sistemih obremenjene s pravili za razpoznavanje konteksta in aktivnostmi za udejanjenje politik. Varnostni modeli, ki so podlaga za pripravo teh varnostnih politik, upoštevajo kontekst le kot dodatek in ne nudijo potrebnih vzvodov za modeliranje prilagodljivih varnostnih aktivnosti.

Kot znanstveni prispevek k odpravi omenjenih pomanjkljivosti in problemov smo v disertaciji najprej predlagali konceptualni model varnostnega konteksta. Model identificira pomembne koncepte varnostnega konteksta, pri tem pa upošteva tudi družbene vidike. Varnostni kontekst je predstavljen s pomočjo množice konceptov na ustrezni ravni abstrakcije. Na podlagi konceptualnega modela smo predlagali varnostni model in jezik, ki omogočata določitev in analizo enostavnih, vendar izrazno bogatih varnostnih politik, primernih za vseprisotne družbene sisteme. Za zagotovitev prilagodljivosti sistema je v varnostni model vpeljan življenjski cikel objekta. Pripravili smo več predlog varnostnih politik, ki zajemajo ponavljajoče se zahteve v raznolikih primerih uporabe.

Konceptualni model ter varnostni model in jezik smo v disertaciji empirično preverili. Najprej smo pokazali, da lahko z našim konceptualnim modelom predstavimo tudi varnostni kontekst drugih pristopov. Varnostni model smo nato uporabili za prepoznavanje možnih groženj virov in njihovih posledic pri različnih aktivnostih. S pomočjo modela in jezika smo analizirali varnost raznolikih primerov uporabe vseprisotnih družbenih sistemov. Izvedljivost našega pristopa smo potrdili s prototipom kontekstnega varnostnega sistema, ki omogoča specifikacijo in udejanjenje varnostnih politik v skladu z našim modelom in jezikom.

V primerjavi z drugimi rešitvami lahko z našim konceptualnim modelom boljše opišemo varnostni kontekst, model pa z vpeljavo družbenih konceptov izboljša njegovo razumevanje. Naš varnostni model omogoča z vpeljavo jasnejših pojmov vseprisotnih družbenih sistemov bolj natančno analizo, kako pomembne kontekstne informacije vplivajo na nadzor dostopa. Skupaj z razvitimi jezikom in predlogami lajša določitev, upravljanje in ponovno uporabo varnostnih politik za vseprisotne varnostne sisteme. Disertacija na splošno nudi izčrpen vpogled v vseprisotne družbene sisteme in sistematično analizira njihove varnostne izzive. Z lažjo določitvijo in uporabo konteksta omogoča tudi lažje načrtovanje in razvoj kontekstnih varnostnih sistemov.

Contents

List of Figures	xvii
List of Tables	xix
Abbreviations	xxi
Symbols	xxiii
Glossary	xxv
1 Introduction	1
1.1 Problem Statement	1
1.2 Goals and Hypotheses	3
1.3 Scientific Contributions	5
1.4 Thesis Overview	6
2 Background	9
2.1 Security	9
2.1.1 Definition of Security	9
2.1.2 Security Concepts	11
2.1.3 Security Policy	12
2.2 Context and Context-Aware Systems	16
3 Ubiquitous Social Systems	19
3.1 Related Work	19
3.2 Characteristics of Ubiquitous Social Systems	20
3.3 Security Analysis of Ubiquitous Social Systems	21
3.3.1 Challenges	21
3.3.2 Approaches	22
3.4 Use Cases of Ubiquitous Social Systems	23
3.4.1 Use Case 1	24
3.4.2 Use Case 2	25
3.4.3 Use Case 3	25
4 A Conceptual Model of Security Context	27
4.1 Related Work	27
4.2 Definition of Security Context	28
4.3 Definition of the Conceptual Model of Security Context	28
4.3.1 Entity	29
4.3.2 Activity	32
4.3.3 Situation	32
4.3.3.1 Social Concepts	33

4.3.3.2	Properties	34
4.3.4	Summary	35
4.4	Evaluation of the Conceptual Model of Security Context	35
4.4.1	Model Application in Use Cases	35
4.4.1.1	Use Case 1	38
4.4.2	Comparison of the Proposed Model with Existing Approaches	43
4.4.3	Security Requirements of the Security Context	44
5	Security Policy Model and Language for Ubiquitous Social Systems	47
5.1	Related Work	47
5.1.1	General Security Policy Models and Languages	47
5.1.1.1	General Security Policy Models	48
5.1.1.2	General Security Policy Languages	53
5.1.2	Contextual Security Policy Models and Languages	56
5.1.2.1	RBAC-Based Models and Languages	57
5.1.2.2	UCON-Based Models and Languages	58
5.1.2.3	Logic-Based Models and Languages	58
5.1.3	Discussion	60
5.2	Development of the Security Policy Model	61
5.2.1	Description of the Model	62
5.2.1.1	Policy Abstractions	62
5.2.1.2	Policy Rules	66
5.2.2	Model-Based Operation	68
5.2.3	Enforcement of the Model	69
5.3	Definition of the Security Policy Language	70
5.3.1	Syntax of the Language	71
5.3.2	Semantics of the Language	73
5.4	Evaluation of the Policy Model and Language	76
5.4.1	Application of the Model within Ubiquitous Social Systems	76
5.4.1.1	Subject-Object Interactions	77
5.4.1.2	Subject-User Interactions	80
5.4.1.3	Subject-Participant Interactions	80
5.4.1.4	Subject-Observer Interactions	82
5.4.2	Application of the Language within Ubiquitous Social Systems . . .	82
5.4.2.1	Group Policy	83
5.4.2.1.1	Subject-Object Interaction Templates	84
5.4.2.1.1.1	Closed Subject and Object	84
5.4.2.1.1.2	Shared Object with Other Subject	84
5.4.2.1.1.3	Shared Subject with Other Object	85
5.4.2.1.2	Subject-User Interaction Templates	86
5.4.2.1.2.1	Closed Subject and Focus	86
5.4.2.1.2.2	Shared Focus with Other Subject	87
5.4.2.1.2.3	Shared Subject with Other Focus	87
5.4.2.1.3	Subject-Participant Interaction Templates	88
5.4.2.1.3.1	Closed Subject and Participant	88
5.4.2.1.3.2	Shared Subject with Other Participant . . .	89
5.4.2.1.3.3	Shared Participant with Other Subject . . .	89
5.4.2.1.3.4	Shared Subject with Other Participant (In-coming Flow)	90
5.4.2.1.3.5	Shared Participant with Other Subject (In-coming Flow)	90

5.4.2.1.3.6	Open Subject and Participant	91
5.4.2.1.4	Subject-Observer Interaction Templates	91
5.4.2.1.4.1	Closed Subject and Observer	91
5.4.2.1.4.2	Shared Subject with Other Observer	92
5.4.2.1.4.3	Shared Observer with Other Subject	93
5.4.2.1.4.4	Open Subject and Observer	93
5.4.2.1.5	Interaction Templates with Relaxation	93
5.4.2.1.5.1	Shared Subject with Other Focus, with Relaxation	94
5.4.2.1.5.2	Shared Focus with Other Subject, with Relaxation	95
5.4.2.1.5.3	Shared Subject with Other Participant (Outgoing Flow), with Relaxation	96
5.4.2.2	Asset Policy	96
5.4.2.2.1	Subject-User Interaction Templates	96
5.4.2.2.1.1	Shared Subject with Other Focus	96
5.4.2.2.1.2	Shared Focus with Other Subject	97
5.4.2.2.2	Subject-Participant Interaction Templates	98
5.4.2.2.2.1	Shared Subject with Other Participant (Outgoing Flow)	98
5.4.2.3	Global Policy	98
5.4.3	Application of the Model and Language within Use Cases	99
5.4.3.1	Use Case 1	100
6	Prototype	107
6.1	Related Work	107
6.1.1	Android	107
6.1.2	Android Security	109
6.2	Description of the Prototype	112
6.2.1	Architecture of the Prototype	112
6.2.1.1	The Policy Decision Point (PDP)	114
6.2.1.2	The Policy Enforcement Point (PEP)	116
6.2.2	Implementation of the Prototype	117
6.2.2.1	Technical Details	117
6.2.2.2	Relationship with Android	119
6.2.2.3	Policy Evaluation in Datalog	120
6.3	Evaluation of the Prototype	121
7	Conclusions	125
7.1	Discussion	125
7.2	Conclusion	128
7.3	Overall Vision	131
7.4	Future Work	132
Appendix A	Evaluation of the Conceptual Model of Security Context	135
A.1	Use Case 2	136
A.2	Use Case 3	138
Appendix B	Evaluation of the Policy Model and Language	143
B.1	Use Case 2	144
B.2	Use Case 3	146

Appendix C Evaluation of the Prototype	153
References	165
Bibliography	173
Biography	175

List of Figures

Figure 2.1:	Security concepts.	12
Figure 2.2:	Security policy.	14
Figure 2.3:	Adaptation loop mechanism.	16
Figure 4.1:	A conceptual model of security context.	29
Figure 4.2:	Types of resources.	31
Figure 4.3:	The association and setting during activities.	36
Figure 4.4:	Activity involving sending information to the screen.	37
Figure 5.1:	Contextual security policy.	56
Figure 5.2:	Policy abstractions used in the proposed model.	62
Figure 5.3:	Object lifecycle.	64
Figure 5.4:	Assets involved in an activity and their related social groups.	67
Figure 5.5:	Activity lifecycle.	68
Figure 5.6:	Border-crossing analogy.	77
Figure 5.7:	Components used in the graphical representation of interactions.	77
Figure 5.8:	Subject-object interaction during the create activity.	77
Figure 5.9:	Subject-object interaction during the destroy activity.	78
Figure 5.10:	Subject-object interaction during the open activity.	78
Figure 5.11:	Subject-object interaction during the execute activity.	79
Figure 5.12:	Subject-object interaction during the close activity.	79
Figure 5.13:	Subject-object interaction during the read activity.	79
Figure 5.14:	Subject-object interaction during the write activity.	80
Figure 5.15:	Subject-user interaction.	80
Figure 5.16:	Subject-participant interaction during the read activity.	81
Figure 5.17:	Subject-participant interaction during the write activity.	82
Figure 5.18:	Subject-observer interaction.	82
Figure 6.1:	Software stack of Android OS.	108
Figure 6.2:	Prototype architecture.	113
Figure 6.3:	Components of the Policy Decision Point.	116
Figure 6.4:	Flow of the decision making process during activity request.	116
Figure 6.5:	Flow of the decision making process following a context change.	117
Figure 6.6:	Class diagram for the proof-of-concept.	118

List of Tables

Table 2.1:	Threat consequences and possible types of threat actions causing them. . .	12
Table 2.2:	Security services.	13
Table 2.3:	Security mechanisms.	13
Table 4.1:	Description of the types of activity included in the use case analysis. . .	38
Table 4.2:	Description of functionality of applications used in Use Case 1.	38
Table 4.3:	Security context in context-aware security systems.	45
Table 5.1:	Description of the types of activity supported by the proposed policy model.	64
Table 5.2:	Objects and their possible identifiers, according to their state.	70
Table 5.3:	System calls for particular operation on objects.	70
Table 5.4:	System social group policy.	100
Table 5.5:	Work social group policy.	101
Table 5.6:	Home social group policy.	101
Table 5.7:	Public social group policy.	102
Table 6.1:	Duration of decision making in the prototype.	122
Table 6.2:	Duration of certain operations on resources.	123
Table 6.3:	Duration of decision making in the prototype, for varying numbers of facts (objects) and rules (soc. groups).	123
Table A.1:	Description of functionality of applications used in Use Case 2 and Use Case 3.	135
Table B.1:	Group policies of a generic company device.	143
Table B.2:	Group policies of a generic home device.	143
Table B.3:	Work social group policy.	144
Table B.4:	Home social group policy.	144
Table B.5:	Public social group policy.	144
Table B.6:	Work social group policy.	146
Table B.7:	Home social group policy.	147
Table B.8:	Public social group policy.	147
Table C.1:	Some terms and their shortened versions.	153
Table C.2:	Decision making in Use case 1, Scene 1.	154
Table C.3:	Decision making in Use case 1, Scene 2.	154
Table C.4:	Decision making in Use case 1, Scene 3.	155
Table C.5:	Decision making in Use case 1, Scene 4.	156
Table C.6:	Decision making in Use case 2, Scene 1.	158
Table C.7:	Decision making in Use case 2, Scene 2.	159
Table C.8:	Decision making in Use case 2, Scene 3.	160

Table C.9: Decision making in Use case 2, Scene 4.	160
Table C.10: Decision making in Use case 3, Scene 1.	161
Table C.11: Decision making in Use case 3, Scene 2.	161
Table C.12: Decision making in Use case 3, Scene 3.	162
Table C.13: Decision making in Use case 3, Scene 4.	163

Abbreviations

BLP	... Bell-LaPadula model
CA-ACON	... Context-aware activity control service
CA-RBAC	... Context-aware role-based access control model
CASS	... Context-aware security system, based on the models and language proposed here
CM	... Context manager
ConUCON	... Context-aware usage control model
DTE	... Domain type enforcement model
DTEL	... Domain type enforcement language
ER	... Environment roles model
ERBAC	... Event-driven role-based access control model
FAF	... Flexible authorization framework
ICC	... Inter component communication
IPC	... Inter process communication
IT	... Information technology
MDM	... Mobile device management system
OS	... Operating system
PAP	... Policy administration point
PDP	... Policy decision point
PEP	... Policy enforcement point
RBAC	... Role-based access control model
SSL/TLS	... Secure socket layer/Transport layer security protocols
UCON	... Usage control model
UID	... User identifier
VoIP	... Voice over internet protocol

Symbols

A	... set of activities
A_C	... set of continuous activities
C	... set of channels
$cctx$	... current security context of an activity, as a function
$CCtx$	... set of current contexts of all continuous activities
Ctx	... set of security contexts
ctx_a	... security context of an activity a
$CtxPol$	... contextual security policy, as a function
D	... set of data
Dec	... set of security decisions
Dev	... set of devices
E	... set of entities
Ev	... set of events
G	... set of goals
h	... the host, $h \in Dev$
Int	... set of interactions
M	... set of methods
O	... set of objects
O_A	... set of active resources
O_P	... set of passive resource
OP	... set of operations
org	... origin of a resource, as a function
Org	... set of origins of all objects
P	... set of persons
Pol	... security policy as a function
pr	... property, as a function
Pr	... set of properties
PrV	... set of property values
Res	... set of resources
Rol	... set of roles
S	... set of subjects
$SDev$	... set of social devices
SE	... set of social entities
SG	... set of social groups
St	... set of system states
u	... the user, $u \in P$

Glossary

Active resource A resource in an active state, in which its information can be read and modified, in the case of data and channel resources, or in which its functionality can be executed, in the case of an application.

Activity A process in which an entity executes an operation on a resource, while trying to accomplish certain goals.

Application A combination of methods which enables the achievement of complex functionality.

Asset An item that is valued. In ubiquitous social systems, assets are resources and entities.

Asset policy rule A policy rule which applies to a particular social group asset.

Association A social group common to all participants in an activity, which describes them appropriately for that activity.

CASS A context-aware security system applicable to ubiquitous social systems, based on the proposed conceptual model of security context and the security policy model and language.

Channel A pathway through which information is exchanged, either on a single device or between two devices.

Conceptual model A model which identifies the important concepts of a notion and the relationships that exist between them.

Context-aware security system A system which is able to adapt its behavior automatically during runtime, according to the discovered security context, in order to provide more relevant security services.

Context-aware system A system that is able to adapt its behavior automatically during runtime according to the discovered context, in order to provide more relevant services.

Context A set of information that influences the behavior of a system. It arises from activity and includes only those items that are relevant to the particular activity at hand.

Contextual information Any information that can be used to characterize the situation of an entity. An entity can be a person or a device, which can be seen as a composition of its resources.

Contextual security policy model A function which maps a certain event within a system state occurring in a particular security context, as an input, to a certain security decision, as an output.

Continuous activity An activity whose effect on the involved asset continues to last after it is performed.

Data A persistent storage of information, saved for future use.

Device A physical object with computing, communication and storage capabilities.

Entity An actor which performs activities. In ubiquitous social systems, this may be a person or a device.

Event Any event that is potentially harmful for a particular system state.

Focus A social group whose goals are considered primary for the user at a given moment.

Global policy rule A policy rule which applies to assets related to any social group.

Group policy rule A policy rule which applies to all assets related to a particular social group.

Host A mobile device from whose perspective the ubiquitous social system is considered. It hosts a certain context-aware security system for the ubiquitous social system.

Information A fact, idea or item of knowledge, represented (encoded) in a form which is understandable to human beings (persons) or machines (devices) (e.g. text, audio, video or binary).

Method A software component that implements a certain functionality; this is executed by the host after the method's interface (API) is called.

Object A resource on which an activity is performed. Passive and active data, channels and applications can be objects. Subjects can also be objects.

Observer A social entity that is near the host or the participants of the activity and can monitor that an activity is being performed.

Origin A single social group which captures the purpose of creation of a resource. An object inherits its origin during creation from the subject which created it.

Participant A social entity which is involved in an activity through consuming or generating resources at the host.

Passive resource A resource in a passive state, in which it is used for storing information and functionality in the device's memory for future use.

Person A human being who performs activities in order to achieve certain goals.

Policy A security policy not otherwise specified.

Policy abstraction A placeholder in a security policy which is replaced by particular elements from the system state.

Policy rule A statement specified over policy abstractions, which must hold in order for the desired security objectives of a policy to be achieved.

Policy template A partially specified policy. This is a set of policy rules in which one or more constant symbols are left unspecified.

Property A quality that describes a resource or its usage.

Resource An item that can be used for processing information. In ubiquitous social systems, a resource may be data, a channel or an application. A resource may be in a passive or active state.

Security context A set of contextual information which is considered relevant to the process of security, regarding a particular task or activity.

Security decision A decision on whether a security-related action should be performed, in order to implement the security policy.

Security policy A set of rules which prevent the abuse and violation of assets of interest and which achieve certain desired security objectives. This is a function which maps a certain event over a system state, as an input, to a certain security decision, as an output.

Security policy language A formal language designed particularly for specification of security policies.

Security policy model A collection of concepts which represent a set of security policies by abstracting their characteristic details.

Security property A property which is relevant to security. The evaluation of security properties should not be seen as verification that the security services, or their implementation, provide the particular security properties required.

Setting A social group common to all observers of an activity, which describes them appropriately.

Social entity An entity which is a member of a social group. Not all devices are social entities.

Social group A group of entities which has defined goals and roles. Goals shape the activities performed within the group, whereas roles denote the qualifications needed by group members in order to accomplish these activities successfully.

Subject A resource which performs an activity by carrying out an operation. Only active applications can be subjects.

System state A collection of system resources and entities, and the values of their properties.

Ubiquitous social system An environment in which a person uses a single, user-owned mobile device while performing activities within several domains such as professional, personal and public life. It encompasses ubiquitous computing, together with the trends of enterprise mobility and consumerization of IT.

User A person who owns and operates the host.

Chapter 1

Introduction

Ubiquitous computing is an intelligent coalition of the physical and virtual worlds, integrated for the purpose of assisting people in their everyday life [1]. Contemporary mobile devices are bringing us closer to this vision, in the form of a single device capable of supporting multiple functionalities [2]. Indeed, it has been shown that a mobile device can facilitate the interaction between people and objects, and help the user to access ubiquitous services [3]. Today, many people are already using mobile devices extensively for communication, education and entertainment, and enterprises are incorporating these into their practices to increase the efficiency, effectiveness and convenience of their business processes [4, 5]. Since people often became attached to their own mobile devices, they also insist on using them, and their familiar services, in their professional life. Initial positive experiences in the form of lower costs and increased employee productivity and satisfaction have recently caused this to develop into a modern enterprise trend known as the consumerization of IT [6–9].

Ubiquitous environments encompassing the trends of enterprise mobility and consumerization of IT are experiencing an increase in social importance. This means that a user-owned mobile device is used to perform both personal and professional activities, anywhere and at any time. Many of these activities are often performed simultaneously, usually with the same consumer-oriented applications, and this blurs the boundaries between the domains of personal and professional life. As people increasingly work for more than one organization and in complex collaboration patterns [10], these environments will span many administrative domains. These environments are referred to as ubiquitous social systems.

1.1 Problem Statement

Ubiquitous social systems present four main challenges for information security. Firstly, the protection of assets of interest becomes complicated, as users undertake an increased number of interactions with other devices while performing activities. These interactions are often ad hoc, unplanned and involve possible leakages of implicit information, due to modern forms of communication [11–13]. Moreover, they are usually performed using consumer-oriented applications, which may well be malicious [14]. Nevertheless, the protection of assets during these interactions depends to a large extent on the users, who are relatively prone to mistakes and unconcerned about this. Secondly, there is an increased risk to the assets of interest from the high probability of attacks. Attackers are predominantly interested in launching attacks on ubiquitous social systems, since they encompass business domains and are less protected than traditional corporate environments. Thirdly, security provision should involve an understanding of the trade-off between security, performance and efficiency in order to better utilize the limited hardware resources of mobile

devices and to operate with minimal human involvement [11, 15–17]. Finally, enforcement of corporate security policies becomes difficult as enterprises have less control over the devices and applications used in their business processes. Since employees are the owners of the devices, enterprises cannot demand that they use the strongest security outside the enterprise domain or forbid the installation of particular applications on their devices.

Many researchers have proposed context-aware security provisioning as a possible solution for overcoming the challenges of ubiquitous social systems. The notion of context has been widely discussed during recent years in various disciplines such as computer science, linguistics and psychology [18]. Context acts as a set of information that influences the behavior of a system, involving, for example, its properties, users and the environment in which it operates. If a system is able to adapt its behavior automatically during runtime, according to the discovered context, in order to provide more relevant services, then it is called a context-aware system [19, 20]. Existing context-aware security systems provide only limited solutions, since they deal only with targeted use cases, for example smart spaces such as the home [21], workplace [22] or hospital [23–25], mobile workers [26–28] or network communications [29, 30].

Two major shortcomings in the existing context-aware security approaches are identified here. Firstly, these are not based on a clear and complete understanding of the context relevant to security; they usually consider this as being identical to the context in general (e.g. [21, 23, 24, 27–29, 31, 32]), although this arises from activities relevant to security and is therefore specific to the security domain. At the same time, they rarely specify the contextual information which they take into consideration. Due to the variety of possible contextual information, it is therefore difficult to understand what these approaches are aware of and adapt to. In addition, the existing approaches (e.g. [21, 24, 26, 27, 29, 32–34]) utilize only a part of the available contextual information and omit some important aspects, for example the social context. Secondly, the existing context-aware security approaches (e.g. [21, 26, 30, 32, 35]) adapt to an arbitrary level of abstraction of contextual information. This can be cumbersome and error-prone, and may jeopardize the validity of the security service. Since there is a vast amount of information which can be used as context, researchers have pointed out that a more sophisticated approach is for context-aware systems to adapt to contextual information with a higher level of abstraction [36–38]. These information interpret raw sensor data meaningfully using the process of context reasoning. They represent the human perception of reality more effectively and are more stable than sensor data, which are often uncertain and change frequently. However, such abstractions of security context that are relevant in making security decisions are still missing.

Security is usually regulated using security policies. A security policy is a set of statements and rules which restrict how a computer system is used, in order to prevent violation and abuse of the assets of interest and to achieve certain desired security objectives. Security research has shown that security policy models and languages are essential for the specification and analysis of security policies [39, 40]. Both models and languages define certain abstractions (e.g. subject, object and right) and provide a means of specifying certain rules for these abstractions (for example, a subject can access an object only if it has a specific right to this access). Policy models focus mainly on representing the characteristic details of a set of security policies and on enabling an analysis of whether they achieve the desired security objectives. Policy languages, on the other hand, focus mainly on defining the syntax and semantics of policy rules, in order to enable the specification of more expressive security policies and to assure their consistency and completeness.

Existing policy models and languages are not applicable to ubiquitous social systems, for two primary reasons. Firstly, they do not provide sufficient abstraction for the spec-

ification and analysis of security policies containing security context. Traditional policy models and languages such as those in [41–45], miss the security context as an abstraction. This is understandable, since these models are not intended for provisioning context-aware security. However, some of them use certain abstractions that resemble contextual information. For example, UCON [46] uses attributes as abstractions, and can handle several properties of both subjects and objects which are useful in making security decisions. Other approaches, such as those of Ponder [47] and Rei [48], can be extended with domain-specific concepts to handle the security context. However, an appropriate definition of the security context and an analysis of the ways in which it influences access control, as the underlying security service in these models and languages, are both crucial for application to ubiquitous social systems. The policy models and languages used by existing context-aware security approaches include security context as an abstraction, although only as an add-on. These are mainly concerned with methods of incorporating security context into traditional policy models and languages, and lack a comprehensive analysis of how it influences access control. In particular, there are two specific characteristics of the (security) context that are rarely modeled, that is, a change in the context and reactions to this change in context. These are of major importance for providing context-aware behavior. It is therefore unclear whether and how these approaches are able to adapt to context changes.

Furthermore, existing policy models and languages use an arbitrary level of abstraction for contextual information within policies. This depends on whether they support context reasoning; those that support this also use a higher level of security context abstraction (e.g. [22, 27]), whereas the others use only low-level contextual information. However, the former incorporate reasoning rules directly into policies, and this has several consequences. Firstly, it means that policy specification is cumbersome and error-prone, since it requires policy makers to have an exhaustive knowledge of context reasoning. It also discourages context reuse between systems, since the reasoned high-level context is available only for the particular security system for which the policy is specified. Next, it can have an impact on the performance of policy evaluation, as context reasoning is required to be performed during this process. Finally, it can even influence the scalability of the system, as it can trigger unnecessary evaluations for each change in the low-level contextual information.

In summary, the existing context-aware security approaches do not provide directly applicable solutions for ubiquitous social systems. Firstly, they do not cover the diverse use cases of ubiquitous social systems. Secondly, they do not have a common and clear understanding of security context and its nature. Thirdly, they lack a precise specification of what they consider as context, and omit important contextual information, such as social context. Fourthly, existing policy models and languages do not provide appropriate abstractions for the specification and analysis of security policies for ubiquitous social systems. Finally, these policy models and languages burden policies unnecessarily with context reasoning rules and policy enforcement actions.

1.2 Goals and Hypotheses

Ubiquitous social systems are emerging environments that would benefit from context-aware security solutions. However, context-aware security systems are complex, and their development is difficult, since knowledge of several disciplines such as security, context-aware systems, adaptable systems, artificial intelligence and policy-based systems is required. This thesis aims to give a more comprehensive view of context-aware security systems and to facilitate their development, especially for environments such as ubiquitous social systems.

In particular, the main goal of this thesis is to facilitate the specification, management

and reuse of security policies within ubiquitous social systems. This aim encompasses several goals, which lead to the expected contributions of this thesis to the existing literature. Each of these is composed of several sub-goals, as follows:

- Analysis of ubiquitous social system use cases from a security perspective:
 - Determine diverse use cases for ubiquitous social systems;
 - Analyze these use cases from a security perspective and specify security contextual information;
 - Determine possible security relevant system adaptations to context changes to applicable part of the system;
- Definition of a conceptual model of security context:
 - Determine the important concepts within the security context and the relationships between them;
 - Define a conceptual model of security context;
 - Analyze existing approaches in terms of their understanding of security context using the model defined above;
- Development of a security policy language and models for ubiquitous social systems:
 - Develop a security policy model based on the conceptual model of security context;
 - Define a domain-specific language for security policies, based on this security policy model and the conceptual model of the security context;
 - Develop a range of security policy templates which would be suitable for the implementation of the use cases considered;
 - Define a general policy enforcement layer with the mechanisms required for support of adaptable security;
- Design and implementation of a prototype:
 - Design a context-aware security architecture for elements of a ubiquitous social system;
 - Implement a prototype;
- Evaluation of a security context-aware solution for ubiquitous social systems:
 - Evaluate the conceptual model;
 - Evaluate the policy models and language;
 - Evaluate the security policy management and enforcement;
 - Evaluate the performance of the prototype.

The main hypothesis in this thesis is:

Context-aware security based on security policies which utilize higher-level security context abstractions is needed for the protection of ubiquitous social systems.

To verify this hypothesis, the following hypotheses are considered:

1. **Ubiquitous social systems require context-aware security provisioning.**
2. **A conceptual model of security context needs to take into account related social aspects.**
3. **A security policy model based on a conceptual model of security context is expressive enough for the protection of ubiquitous social systems.**
4. **Security policies based on higher-level security context abstractions facilitate policy specification, management and reuse, while providing adequate performance for context-aware security systems.**

1.3 Scientific Contributions

The work in this thesis offers the following original contributions to the existing scientific knowledge.

- **A conceptual model of security context.** A conceptual model of security context is defined which identifies the important concepts that constitute the security context and the relationships that exist between them. This conceptual model describes the security context more clearly than the alternatives, by representing it using only a few high-level abstractions. Previous researchers have already pointed out that higher-level context abstractions make context-aware systems more sophisticated, because they represent human perceptions of reality more effectively and are more stable [36–38]. However, context abstractions relevant to security are still absent, and the proposed conceptual model attempts to fill this gap. Moreover, the conceptual model improves the current understanding of security context by introducing social concepts as abstractions for the first time. Overall, the proposed conceptual model is both expressive and practical enough to describe diverse ubiquitous social system use cases and to make them secure.
- **A security policy model for ubiquitous social systems.** A security policy model is defined for ubiquitous social systems. This identifies the necessary abstractions and rules for the specification and analysis of the security policies applicable within these environments. This policy model gives a primary role to security context within decision making, while other models generally treat the security context as an add-on. In particular, all possible contextual information is defined as abstractions in the proposed policy model, based on the proposed conceptual model of security context. This enables a clearer and more precise analysis of the ways in which important contextual information influences access control, as the underlying security service of the policy model. Furthermore, the proposed policy model provides more intuitive abstractions for ubiquitous social systems; it takes into account the related social aspects and introduces a novel state diagram to represent the lifecycle of objects. The latter also enables the easier provision of context-aware security behavior, as it assists in the detection of context changes and the specification of reactions to these changes. Overall, the proposed policy model facilitates the specification and analysis of expressive security policies which involve security context and are applicable to ubiquitous social systems.

- **A security policy language and templates for ubiquitous social systems.** A security policy language is defined for ubiquitous social systems, which enables policy makers to easily specify the desired requirements for protection of their assets of interest in these environments. Unlike other alternatives, the proposed security policies comprise only a few high-level context abstractions, and are not subject to context reasoning rules and policy enforcement actions. A range of security policy templates for ubiquitous social systems are also defined, which capture the recurring requirements of personal and professional domains within diverse use cases. Together, the proposed policy language and templates facilitate the specification, management and reuse of security policies for ubiquitous security systems.
- **A comprehensive view of security for ubiquitous social systems.** An improved understanding of ubiquitous social systems and their security challenges is offered in this work by systematically analyzing them from the perspective of security. Firstly, the possible threats to the assets of interest are identified during various activities, together with their potential consequences. Following this, diverse ubiquitous social system use cases are analyzed by describing the operation of an applicable context-aware security system. These analyses are more general than others in the literature, since they cover a variety of security requirements from both the personal and professional domains in ubiquitous social systems. The existing approaches mainly deal with specific use cases, which usually address only part of a ubiquitous social system. In addition, the analyses carried out here can assist policy makers in better understanding how ubiquitous social systems can be secured with appropriate security policies, and how to specify these.
- **A novel approach to context-aware security provisioning.** This thesis facilitates the design and development of context-aware security systems by combining knowledge from context-aware systems, adaptable systems and security. In particular, the proposed conceptual model facilitates the neglected task of context specification, by identifying the important concepts that constitute the security context and the relationships that exist between them. This helps system designers and developers to better understand the issues of which their systems are required to be aware and to which they are required to adapt, and assists them in selecting this contextual information from the multitude of possible options. Moreover, the proposed policy model facilitates the task of context usage by providing a means for the specification and analysis of reactive (adaptive) behavior relevant to security. This helps policy makers to more easily represent their desired security requirements for their assets of interest.

1.4 Thesis Overview

This thesis is structured as follows.

Chapter 2 gives an overview of the research areas addressed by this thesis, namely security- and context-aware systems. It defines some of the basic terms in these areas and serves as the background necessary to understand the rest of the thesis.

Chapter 3 introduces ubiquitous social systems, as the main application environments of the work developed in this thesis. It describes their main characteristics and analyzes them from the perspective of security. It also presents typical use cases of ubiquitous social systems. In addition to being useful as examples, these use cases will be essential for the various evaluations in the subsequent chapters.

Chapter 4 gives the core definitions used throughout the thesis. It defines the notion of security context and proposes a conceptual model of security context. This proposed conceptual model is the central part of the thesis, as the remainder of the work is mainly based on this. The model is evaluated by applying it within ubiquitous social system use cases and by comparing it with the existing approaches.

Chapter 5 presents the main practical applications of the current work. It defines a policy model and language for ubiquitous social systems. The policy model enables the analysis of various activities from ubiquitous social systems, while the language enables the specification of a security policy according to these analyses. Both of these utilize the conceptual model of security context. A range of security policy templates are also defined which are applicable in various ubiquitous social system use cases. The proposed policy model is evaluated by using it to identify the possible threats to the assets of interest during various activities, together with their potential consequences. In addition, to evaluate the proposed policy model and language, they are used to analyze diverse ubiquitous social systems use cases from the perspective of security.

Chapter 6 describes a prototype implementation of a context-aware security system applicable to ubiquitous social systems. This prototype enables the specification and enforcement of security policies, according to the proposed policy model and language. It is intended for mobile devices operating on Android.

Chapter 7 concludes the work presented in this thesis, discusses the main contributions that this work offers and identifies directions for future work.

In addition, Appendices A, B and C present the evaluations of the proposed conceptual model of security context, policy model and language, and prototype, respectively. The evaluations from Chapters 4, 5 and 6 are continued in more detail in these appendices, in order to enhance the readability of the thesis.

Chapter 2

Background

This chapter gives an overview of two research areas tackled by this thesis: security and context-aware systems. Several of the basic terms in these areas are defined, and an explanation of how they are related is given.

2.1 Security

Although security is an established field within computer science, universal agreement on many of the terms used in the literature is lacking. Various documents define the same terms slightly differently, and the definitions of terms often overlap. Several references are combined when defining the background terms used in this thesis, namely [39, 40, 49, 50]. References [49] and [50] are standardized documents which focus on security within network- and internet-based systems. Since ubiquitous social systems are themselves such systems, the definitions in these documents seem applicable for the current work. Moreover, these documents provide similar definitions for the majority of the terms. Furthermore, [39] and [40] are textbooks, which cover various topics in this field and are respected sources of learning material. The former provides a comprehensive description of the terms, with reference to several standardized documents, while the latter provides more formal definitions. In the remainder of the thesis, computer security is referred to simply as security.

2.1.1 Definition of Security

Security concerns the protection of certain assets and is usually discussed in the context of information systems.

An *information system* is an organized assembly of resources, which enables the processing of information in order to accomplish a specified set of functions. *Information* is defined as a fact, idea or item of knowledge, represented (encoded) in a form understandable by humans or machines.¹ A *resource* is something that can be used to process information. For example, a resource can be in the form of hardware (e.g. a computer, server, or smartphone), the capacity of hardware (e.g. memory, computing or battery power), software (e.g. an operating system, service or application), network equipment (e.g. a communication link, bridge, or router) or a file (e.g. a document which contains

¹Often, the security literature does not distinguish between the concepts of information and data, and uses these interchangeably. In other cases, for example in [49], data is considered to be a representation of information. In this thesis, a clear distinction is made between these terms. The definition of information used here corresponds to what is usually referred to as data in the literature. The definition of data used in this work is given in the next chapter.

information). Since resources are components of the information system, they are usually referred to as *system resources*.

Each information system has a set of entities and assets. An *entity* is an actor which performs activities. This is usually an individual who uses the functions provided by an information system while performing the activities. An *asset*, on the other hand, is something that is valued. In general, this is a resource which is valued by an entity. Entities are not merely consumers of information systems and their assets; they also produce new assets. For example, a human user can create new files, which need to be further protected. It should be acknowledged that all assets have a particular entity, or group of entities (i.e. an organization) which owns them, especially those which are newly produced. These entities are referred to as *owners* of these assets. Owners can protect their assets from being abused or violated using security.

Definition 2.1 (Security). *Security* is the process of protecting assets of interest in order to attain the objectives of confidentiality, integrity, and availability.

The definition introduces three key objectives, each of which is defined below.

Confidentiality covers two concepts: data confidentiality and privacy.

Definition 2.2 (Data confidentiality). *Data confidentiality* assures that data is not disclosed to system entities unless they have been authorized to have access to that information. This objective refers to data.

Definition 2.3 (Privacy). *Privacy* assures that an entity (typically a human individual) can determine the degree to which it will interact with its environment, including the degree to which that entity is willing to share its personal information with others.

Integrity involves three concepts: data, correctness and source integrity.

Definition 2.4 (Data integrity). *Data integrity* assures that data has not been changed, destroyed, or lost in an unauthorized or accidental manner.

Definition 2.5 (Correctness integrity). *Correctness integrity* assures that the information represented by data is accurate and consistent.

Definition 2.6 (Source integrity). *Source integrity* assures that data is trustworthy (i.e., worthy of reliance or trust), based on the trustworthiness of its sources and the trustworthiness of any procedures used for handling data in the system.

Definition 2.7 (Availability). *Availability* assures that a system or a system resource is being accessible, or usable or operational upon demand, by an authorized system entity.

In addition, authenticity and accountability are often considered as separate objectives of security.

Definition 2.8 (Authenticity). *Authenticity* assures that a system entity or a system resource is genuine and is able to be verified and trusted.

Definition 2.9 (Accountability). *Accountability* assures that the actions of a system entity on a system or a system resource can be traced uniquely to that entity, which can then be held responsible for its actions.

2.1.2 Security Concepts

Several additional concepts are introduced here which will aid in a better understanding of the process of security.

Security concerns ways in which to deal with possible threats to assets. A *threat* is a circumstance or event with the potential to adversely affect an asset. It exists because of vulnerability, which is a flaw or weakness in the design, implementation, operation or management of the information system. Threats may be (i) intentional, resulting from the activity of a malicious entity in an attempt to exploit a vulnerability; (ii) accidental, resulting from a human error or an omission by a system entity; or (iii) environmental, resulting from an environmental disaster, such as fire, flood, earthquake, or explosion. Since environmental threats are the subject of concern of the field of physical security, the remainder of this thesis deals only with intentional and accidental threats. Thus, if not otherwise specified, these are referred to simply as threats.

An *attack* is a deliberate event, carried out in order to realize an intentional threat. If successful, an attack leads to an undesirable violation of security, or threat consequence. The entity carrying out an attack is known as an *attacker*. In order to do this, the attacker must have a motive for abusing an asset, a method or the technical ability to carry out an attack, and an opportunity to appropriately access the targeted system.

Owners can protect their assets by imposing countermeasures. A *countermeasure* is an action, device or procedure which is devised in order to meet or oppose a threat. In general, countermeasures provide three types of functions: (i) prevention, which eliminates the possibility of a security violation taking place; (ii) detection, which identifies that a security violation is in progress or has recently occurred, thus reducing the potential loss from it; and (iii) recovery, which restores the normal operation of the system following a security violation and possibly mitigates the negative effects of this.

It is not normally technically and financially feasible for owners to address all possible threats to system assets. Instead, they need to decide which assets are of greatest interest for protection, and concentrate on protecting these. The notion of risk can help owners in this assessment. *Risk* is the expectation of loss, expressed as the probability that a particular threat will occur for an asset, with the possible cost of its consequences. The former depends on the degree of vulnerability and the effectiveness of the countermeasures in use, whereas the latter is related to the perceived value of the asset for the owner. By protecting assets according to risk, owners can better allocate available countermeasures and security resources so as to minimize total losses.

Fig. 2.1 illustrates the relationship between these newly introduced concepts, essentially providing a general model of security.

Threats can be divided into four categories according to their consequences. Table 2.1 describes each of these, based on [39, 49], together with the possible types of actions (attacks) which may result in each consequence.

Countermeasures can be either security services or mechanisms [39]. A *security service* is a processing or communication service provided by an information system in order to give a specific type of protection to system resources. Table 2.2 (based on [49]) describes some of the more widely used security services. A *security mechanism*, on the other hand, is a method or process that can be used in a system to implement a security service provided by the system. Table 2.3 (based on [39, 49, 50]) describes some of the more widely used security mechanisms.

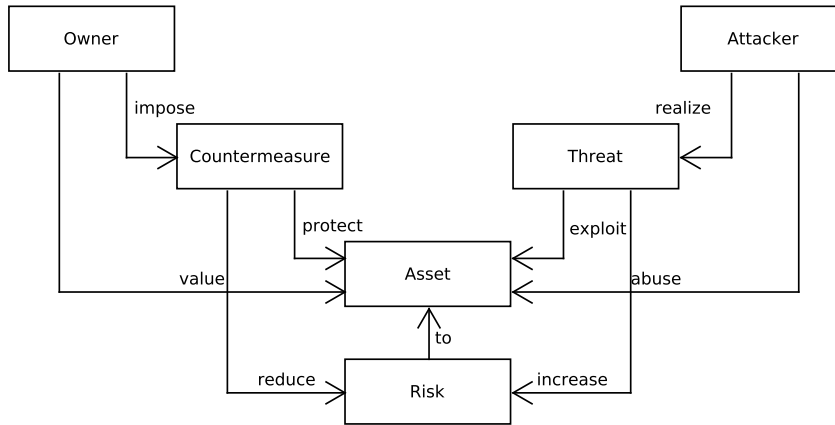


Figure 2.1: Security concepts.

Table 2.1: Threat consequences and possible types of threat actions causing them.

Threat Consequence	Threat Action
<i>Unauthorized disclosure:</i> A circumstance or event whereby an entity gains access to information for which the entity is not authorized.	<i>Exposure:</i> Sensitive data is directly released to an unauthorized entity. <i>Interception:</i> An unauthorized entity directly accesses sensitive data while the data is traveling between authorized sources and destinations. <i>Inference:</i> An unauthorized entity indirectly accesses sensitive data (but not necessarily the data contained in the communication) by reasoning from characteristics or byproducts of communication. <i>Intrusion:</i> An unauthorized entity gains access to sensitive data by circumventing a system's security protections.
<i>Deception:</i> A circumstance or event that may result in an authorized entity receiving false data and believing it to be true.	<i>Masquerade:</i> An unauthorized entity gains access to a system or performs a malicious act by illegitimately posing as an authorized entity. <i>Falsification:</i> False data deceives an authorized entity. <i>Repudiation:</i> An entity deceives another by falsely denying responsibility for an act.
<i>Disruption:</i> A circumstance or event that interrupts or prevents the correct operation of system services and functions.	<i>Incapacitation:</i> A threat action that prevents or interrupts system operation by disabling a system component. <i>Corruption:</i> A threat action that undesirably alters system operation by adversely modifying system functions or data. <i>Obstruction:</i> A threat action that interrupts delivery of system services by hindering system operations.
<i>Usurpation:</i> A circumstance or event that results in control of system services or functions by an unauthorized entity.	<i>Misappropriation:</i> An entity assumes unauthorized logical or physical control of a system resource. <i>Misuse:</i> A threat action that causes a system component to perform a function or service that is detrimental to system security.

2.1.3 Security Policy

Security is regulated according to a set of statements and rules, which are defined in the form of a security policy. The concept of a security policy has been used to describe two terms in the literature: organizational and automated security policies [51].

Definition 2.10. An *organizational security policy* is a set of laws and practices by which an organization protects its assets of interest in order to achieve its desired security objec-

Table 2.2: Security services.

Security service	Description
Access control service	Protects against a system entity using a system resource in a way not authorized by the system's security policy.
Audit service	Records information needed to establish accountability for system events and for the actions of system entities that cause them.
Availability service	Protects a system to ensure its availability.
Data confidentiality service	Protects data against unauthorized disclosure.
Data integrity service	Protects against unauthorized changes to data, including both intentional change or destruction and accidental change or loss, by ensuring that changes to data are detectable.
Data origin authentication service	Verifies the identity of a system entity that is claimed to be the original source of received data.
Non-repudiation service	Provides protection against false denial of involvement in an association (especially a communication association that transfers data).
Peer entity authentication service	Verifies an identity claimed by or for a system entity in an association.
System integrity service	Protects system resources in a verifiable manner against unauthorized or accidental change, loss, or destruction.

Table 2.3: Security mechanisms.

Security mechanism	Description
Specific mechanisms	Mechanisms that provide a specific security service
Encipherment	The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption keys.
Digital signature	Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (e.g., by the recipient).
Access control	A variety of mechanisms that enforce access rights to resources.
Data integrity	A variety of mechanisms used to assure the integrity of a data unit or stream of data units.
Authentication exchange	A mechanism intended to ensure the identity of an entity by means of information exchange.
Traffic padding	The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.
Routing control	Enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.
Checksum	A value that (a) is computed by a function that is dependent on the contents of a data object and (b) is stored or transmitted together with the object, for detecting changes in the data.
Notarization	The use of a trusted third party to assure certain properties of a data exchange.
Pervasive mechanisms	Mechanisms that are not specific to any security service
Trusted Functionality	That which is perceived to be correct with respect to some criteria (e.g., as established by a security policy).
Security Label	The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.
Event Detection	Detection of security-relevant events.
Security Audit Trail	Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.
Security Recovery	Deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

tives.

Definition 2.11. An *automated security policy* is a set of rules which restrict how computer system are used, in order to prevent violation of the organizational security policy.

An organizational security policy prescribes how employees should act while performing various business processes within an organization, thus directly governing their behavior. It is specified informally, in human readable form, and is usually based on standardized documents such as [52]. An automated security policy, on the other hand, manages how computer programs (and processes) operate within the system and how they are used within the business processes of the organization. In return, this restricts the activities of the employees, and thus, indirectly governs their behavior. Automated security policies are usually specified in machine understandable form, in order that they can be enforced automatically by the computer system. The scope of automated security policies is large and covers all the security services which are needed to implement and enforce an organizational security policy. In order for these policies to be specified and analyzed more easily, they are divided into smaller subsets, which are examined separately and in a more formal manner.

The work in this thesis focuses on these subsets of automated security policies. In order to be as clear as possible, in the rest of the thesis, these subsets will be referred to simply as security policies. More precisely, security policy is defined as follows.

Definition 2.12. A *security policy* is a function, $Pol : Ev \times St \rightarrow Dec$, which maps a certain event, $ev \in Ev$, over a system state, $st \in St$, as an input, to a certain security decision, $dec \in Dec$, as an output.

Figure 2.2 illustrates security policy as a function. An input event may be any event that is potentially harmful for the particular system state, which is a collection of system resources and entities and the values of their properties. As may be expected, whether an event is harmful or not depends on the desired security objectives of the policy.² A security decision is a decision on whether a security-related action should be performed in order to implement the security policy.

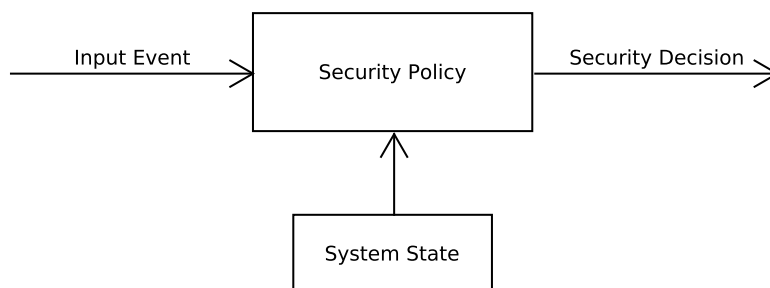


Figure 2.2: Security policy.

Security research has shown that security policy models are essential for the specification and analysis of security policies [39, 40].

Definition 2.13. A *security policy model* is a collection of concepts which represent a set of security policies, by abstracting their characteristic details.

²The security objectives of a security policy are a subset of the security objectives of the organizational security policy, which it implements and enforces.

In fact, a security policy model describes a security policy in terms of a function and relates it to the underlying security services which implement it. Security policy models are concerned with two types of concepts: policy abstractions and policy rules.

Policy abstractions are placeholders in security policies, which are replaced by particular resources, entities or property values from the system state. In practice, they are more strongly related to security services which implement security policies, and abstract how they operate and function. Policy abstractions can be active, passive or specific. An active abstraction, usually referred to as a subject, is an entity or a resource which is capable of performing an operation. It is an abstraction on which a security policy is enforced. A passive abstraction, usually referred to as an object, is an entity or a resource on which operations are performed; it is an abstraction which needs to be protected by a security policy. A specific abstraction is an auxiliary abstraction, which helps in defining how the protection of objects from subjects takes place, according to a particular security service. It can capture the important characteristics of subjects and objects, as well as the relationships existing between them.

Policy rules are statements specified for policy abstractions which must hold in order for the desired security objectives of a policy to be achieved. In fact, it is these statements that define a security policy as a function. Some policy rules are explicitly specified by certain policy makers and apply only for a particular security policy, based on the model. These rules are referred to here as explicit policy rules. In order to enable the specification of explicit policy rules, policy models must provide a means for their representation, in such a way that they can be later evaluated during decision making. Other policy rules are predefined in the model and apply to every security policy it represents; these are referred to as implicit policy rules. These rules are established during the creation of the model and remain constant, meaning that policy makers cannot modify them. Policy models rarely describe how implicit policy rules are represented, and leave this to their particular implementation.

Security policy models (e.g., [41–43, 46]) mainly focus on the analysis of security policies and only provide a means for the specification of simple security policies. In particular, they try to represent the underlying security service of the policy and its operation, which is why they are primarily concerned with policy abstractions. They also identify which details of the security policies are meaningful for security. As a result, this makes security analysis clearer, as it helps security analysts to focus solely on the more important details, omitting those which are unnecessary. At the same time, this makes security analysis more useful and practical, since it applies to all security policies represented by the model.

In order to specify more expressive security policies, security policy languages must be used (e.g., [44, 45, 47, 48]).

Definition 2.14. A *security policy language* is a formal language designed particularly for the specification of security policies. It is a collection of symbols, words and sentences which can represent a set of security policies, implemented in an information system through a security service.

As a formal language, a policy language has its own syntax and semantics. The syntax of a language defines the rules and principles according to which the symbols are formulated. The semantics of the language gives the particular meaning of these symbols. Security policy languages are usually based on more general policy abstractions than policy models, and are mainly focused on policy rules. They enable the specification of several types of policy rules, and also ensure that the resulting security policies will be consistent and complete, despite possible human errors.

2.2 Context and Context-Aware Systems

The notion of context has been widely discussed in recent years by many researchers from various disciplines such as computer science, linguistics, and psychology [18]. Although the understanding of context depends strongly on the specific domain, some characteristics of context appear to be common. *Context* acts as a set of information that influences the behavior of a system, such as a description of the system, its users and the environment in which it operates. However, context arises from activity, and includes only those items that are relevant to the particular activity at hand [53, 54]. Systems can utilize context in various ways [19]. Some systems only present context to an interested user, or make it persistent for the user to retrieve later. These systems track changes in context and update its value, in order to present the current context to the user. Other systems automatically adapt their behavior during runtime in accordance with the discovered context, in order to provide more relevant services. This thesis focuses more on the latter systems, and in particular on those which provide security services. Therefore, in the remainder of this thesis, the usage of the term ‘context-aware security systems’ refers to these systems.

In order to understand how context-aware systems operate, valuable information can be also found in the literature on adaptable systems, for example [55, 56]. Ideally, an adaptable system should embody an adaptation loop mechanism, consisting of the following processes executed at runtime (Fig. 2.3). First, it monitors the sensors and collects raw data (*monitoring*). Following this, it analyzes the received data and detects changes in context (*detecting*). Next, it decides whether these changes require adaptation and determines the most appropriate actions to be taken (*deciding*). Finally, it executes these actions through the actuators (*acting*). These processes form a closed loop, which means that the actions executed by the actuators can further influence the data sensed by the sensors. Context-aware systems place more emphasis on the monitoring and detecting processes, whereas adaptable systems place emphasis on the deciding and acting processes.

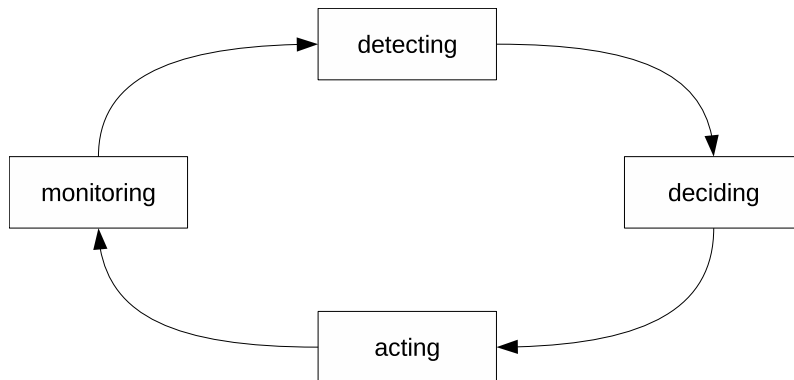


Figure 2.3: Adaptation loop mechanism.

Three tasks present challenges in the design and development of context-aware systems [57]:

- *Context specification*, in which the contextual information relevant to the particular service provided by the system at hand is clarified. This task can be facilitated using a conceptual model of context [58]. A conceptual model of context identifies the important concepts that constitute context and the relationships that exist between them.
- *Context management*, in which relevant contextual information is appropriately han-

dled. This task involves several processes such as acquisition, representation and reasoning about context, as well as detection of context changes.

- *Context usage*, in which the way the system should adapt its behavior according to the discovered context is specified, in order to provide a more appropriate service. One approach to specifying how this is done involves policies [56]. Policies decouple the rules that guide the behavior of a system from its functionality, thus enabling modification during runtime [59, 60].

While context management approaches tend to be generic and applicable to various context-aware systems and types of contextual information, context specification and usage are more specific to the particular system and are domain-dependent. In view of this, only context management is described in more detail below; the other two tasks are extensively discussed in a later section of this thesis.

Context representation and reasoning are the core processes of context management.³ *Context representation* denotes the process of formal representation of contextual information [36, 61–63]. There are several techniques for context representation, such as (i) key-value, which simply maps contextual information and its values in a tabular fashion; (ii) object-oriented, which utilizes the approaches of encapsulation and reusability; (iii) logic-based, which represents contextual information as facts, expressions and rules in a particular logic language; and (iv) ontology-based, which represents contextual information as concepts and relationships using the instrument of ontology.

Context reasoning denotes the process of derivation of new and higher-level contextual information from raw and low-level sensor values [36, 61–63]. Higher-level contextual information is more meaningful, represents human perception of reality more effectively and is more stable than sensor data, which can be uncertain and may change frequently. As a result, context-aware systems that adapt to such information are more sophisticated. Context reasoning depends to a large extent on context representation. Of the above-mentioned representation techniques, ontology-based representation appears to be most acknowledged, as it provides good balance of simple and rigorous representation and a powerful support for reasoning [36, 61].

³These are commonly referred to as context modeling.

Chapter 3

Ubiquitous Social Systems

Ubiquitous social systems are environments which encompass ubiquitous computing, together with the trends of enterprise mobility and consumerization of IT. The concept of a ubiquitous social system is defined and introduced for the first time in this thesis. In this chapter, these environments are described in more detail. Firstly, an overview is given of some of the most notable works regarding each of the areas involved. Following this, the characteristics of ubiquitous social systems are identified and analyzed from the perspective of security. In particular, the main challenges for security and some of the approaches which address these challenges are described. Finally, three typical use cases for ubiquitous social systems are presented.

3.1 Related Work

Ubiquitous computing is an intelligent coalition of the physical and virtual worlds, integrated with the aim of assisting people in their everyday life. This vision dates back to the early 1990s, when Weiser described it in his seminal paper “The Computer of the 21st Century” [1], through the metaphor of profound technologies that “weave themselves into the fabric of everyday life until they are undistinguishable from it”. Ubiquitous computing is about a network of physical objects that are equipped with various tags and sensors and are able to compute, communicate among themselves, and communicate with humans. Consequently, objects become “smart” and are able to “sense” their environment, react autonomously on the changes and have an influence on the physical world [2, 64]. Several systems usually referred to as “smart spaces”, have been developed using environments with smart objects that implement certain aspects of ubiquitous computing [20]. However, these mainly involve a fixed space (e.g. a room) and cannot be deployed on a large scale (e.g. a city). Modern mobile devices can help in overcoming these issues. Their development has caused a shift in their role from terminal devices to personal computers which are capable of executing complex functions and supporting multiple functionalities of ubiquitous computing [2]. A “personal smart space” can be developed on each mobile device, which enables interaction between people and objects and helps the user to access various ubiquitous services [3]. Personal smart spaces act as glue for the various fixed smart spaces and can facilitate their integration. As a result, this can bring us closer to the vision of ubiquitous computing.

Mobile devices provide enormous possibilities in terms of usage. This primarily involves communication and entertainment; however other uses include remote access to data, mobile working, education, collaboration, healthcare etc. Because of their versatility, they have quickly become useful and popular gadgets. Enterprises have also started to incorporate mobile devices into their practices, a trend referred to as enterprise mobility.

Enterprise mobility mainly centers around remote access to company data and services from outside company premises, which facilitates collaboration between employees, communication with customers, enterprise resource planning, customer relationship management and office applications [65]. As a result, this enables employees to perform their work regardless of their location, giving them more flexibility in managing their working hours. It also enables enterprises to improve service delivery and to increase the efficiency, effectiveness and convenience of their business processes [4, 5]. As people become attached to their own mobile devices, they also insist on using them, together with their familiar services, in their professional life. Initial positive experiences in the form of lower costs and increased employee productivity and satisfaction have recently caused this to develop into the modern enterprise trend known as the consumerization of IT [6–9].

3.2 Characteristics of Ubiquitous Social Systems

A ubiquitous social system is a ubiquitous environment which encompasses the trends of enterprise mobility and consumerization of IT. In this environment, an individual uses a single, user-owned mobile device while performing activities within the domains of both professional and personal life. Ubiquitous social systems can be more precisely described using the following characteristics.

A ubiquitous social system is a ubiquitous environment which encompasses the trends of enterprise mobility and consumerization of IT. In this environment, a person uses a single, user-owned mobile device while performing activities within the domains of professional and personal life. Ubiquitous social systems are described more precisely through the following characteristics.

- Users often perform activities within different domains simultaneously. This involves activities from professional and personal life, as well as public activities. As people increasingly work for more than one organization and with complex patterns of collaboration, these environments also span many administrative domains.
- Users use their own mobile devices when performing the various activities. Although they may be obliged to obey a certain security policy, for example specified by their employer, they are still free to use their mobile device as they wish. Among the many things users can do with their devices is the installation of various applications.
- Users perform activities within the different domains with the same applications, which are usually consumer-oriented. This means that companies adopting ubiquitous social systems allow their employees to use applications created for personal usage while performing their professional activities.
- Users perform these activities in various surroundings, which are often different from the one in which the activity is natively performed. For example, professional activities can be performed at home or in public, while personal activities can be performed at work.

As a result, ubiquitous social systems blur the boundaries between the domains of professional, personal and public life. This does not mean that the domains themselves become combined, but that the activities performed within them cannot be easily separated, since the traditional means of separation, such as the location where activities are performed, time when they are performed, and the applications and devices used during their performance, no longer apply. For comparison, in enterprise mobility, users perform only their professional activities using company-owned mobile devices. Although this also

involves performing activities in various surroundings, these activities are mainly carried out using business-oriented applications.

3.3 Security Analysis of Ubiquitous Social Systems

The characteristics of ubiquitous social systems amplify the existing threats to ubiquitous computing, enterprise mobility and consumerization of IT.¹ This makes them attractive targets for attacks, especially with respect to assets related to business domains (e.g. enterprise data, processes and networks). The main challenges for security in ubiquitous social systems are presented below, and some of the existing approaches applicable to these challenges are described.

3.3.1 Challenges

Ubiquitous social systems present four main challenges for security, as described below.

Firstly, the protection of assets becomes complicated. Users carry out an increased number of interactions with their devices while performing activities. These interactions are often ad hoc, unplanned, and use various sensor data or audio-visual forms of communication with other people. Modern forms of interaction cause leakage of implicit information and can result in unauthorized disclosure [11–13]. The fact that these activities are usually performed simultaneously further amplifies this threat. Furthermore, these activities are usually performed using consumer-oriented applications, which can easily be malicious. Mobile malware is on the rise, and mobile applications routinely require extensive permissions during installation [14]. A mobile application with suitable permissions (e.g. permissions for both data and network access) can leak confidential information from the device, or can modify and delete important information from it. This can result in unauthorized disclosure or disruption, respectively. Moreover, a malicious application may inspect corporate networks and systems and even launch various attacks on them, causing disruption or usurpation. In addition, the protection of assets becomes complicated as it depends to a large extent on the user. Users are more prone to (accidental) mistakes and can easily become confused about the purpose of an activity they perform or whether they are allowed to perform an activity within a certain environment. Therefore, they may expose sensitive data to unauthorized entities or inadvertently corrupt it, resulting in unauthorized disclosure and disruption.

Secondly, there is an increased risk to the assets of interest, as there is a high probability of attack. Attackers are predominantly interested in launching attacks on ubiquitous social systems, since they encompass business domains and are less protected than traditional corporate environments. Moreover, attackers are highly motivated to create malware, as such applications can affect both professional and personal life simultaneously after being installed. For example, an application with permissions for storage access in Android can utilize these for any data, regardless of whether it is for personal or business use. In addition, an application with permission to accessing a sensor can utilize this whenever it wants, regardless of whether the user is at work or at home. Thus, if an attacker succeeds in deceiving the user into installing a malicious application intended for personal usage, which is often easier, s/he can automatically gain access to business assets on or from the same mobile device.

¹These areas are not analyzed separately here; the reader is referred instead to [7, 15, 17], which provide a good overview of the security issues. However, these analyses are taken into account when describing the security challenges of a ubiquitous social system.

Thirdly, security provision should understand the trade-off between security, performance and efficiency in order to utilize the limited hardware resources of mobile devices more effectively [11, 15–17]. The domains of professional and personal life do not require the same level of security protection. For example, an enterprise may require stronger security for its assets than the user, who uses certain applications for personal activities. Existing consumer-oriented applications provide only a single set of security mechanisms; for example, they may not provide appropriate protection for certain domain's assets, providing either too much or not enough protection. Countermeasures are needed which can adapt to the different requirements for the domains of professional and personal life, and which can achieve this with minimal human involvement.

Fourthly, the enforcement of security policies becomes difficult. This is especially true for business domains, since enterprises have less control over the devices and applications used in their business processes. Since the employees are the owners of the devices, enterprises cannot demand that they use the strongest possible security outside the enterprise domain, or forbid them to install various applications on their devices. Any application installed on the mobile device can affect enterprise assets, even though it is not intended to be used for the purpose of that enterprise. In addition, many of the consumer-oriented applications used for enterprise purposes might not be capable of providing appropriate protection.

3.3.2 Approaches

Several alternative approaches to context-aware security are described here which seem interesting in terms of the security challenges identified above. However, these are only partially applicable to ubiquitous social systems.

Mobile device management (MDM) is a security system developed specifically in view of the trends towards enterprise mobility and consumerization of IT. It is widely popular nowadays, and many enterprises require their employees to install it prior to allowing them to operate mobile devices at work. MDM systems mainly address the risks of device loss or theft; however, other applications such as access control or secure communication are also possible [66]. They are usually integrated into the operating system and are provided by mobile device vendors. Their aim is primarily to enable the enforcement of corporate policies, and they are therefore intended for business domains. In particular, they offer some of the following mechanisms: password authentication, restriction of installation of applications, remote storage erasure, device deactivation, etc. MDM systems do not differentiate between the domains of professional and personal life in which the device is used, but enforce corporate policies for all assets accessible from the device. This means that the business domain is given priority over the other domains. For example, if the device's storage is erased remotely, both professional and private data will be lost; this is generally unacceptable for the users.

Virtualization provides computing environments called virtual machines which are isolated from each other and operate independently on the same hardware [66, 67]. Virtualization can be achieved at various levels of the operating system. For example, it can be achieved above the kernel layer, in which case two virtual machines share the same kernel; alternatively, it can be achieved at all OS layers, including the kernel. Virtualization approaches have been widely used in traditional computers, in order to increase security and reduce the cost of deployment of applications. However, their development for mobile devices has lagged behind that of personal computing; this is mainly due to their architecture, which was not designed with virtualization in mind and which suffers from low performance. Because of this, approaches such as those described in [67–69] try to achieve virtualization above the middleware layer, or only for certain aspects of the

operating system (e.g. memory). These approaches are specifically designed for mobile devices and are thus lightweight in performance. However, drawbacks common to all virtualization approaches are that their configuration is hardcoded, only one virtual machine can be active at a given moment, and switching between virtual machines requires user interaction.

Compartmentation refers to the problem of controlling how information flows horizontally (i.e. “across” departments) rather than vertically (i.e. “down” a hierarchy) [70]. Relevant applications range from healthcare to national intelligence, and include cases where the confidentiality and integrity of corporate data or the privacy of individual customers’, citizens’ or patients’ data is important. Approaches for controlling vertical information flow, such as those described in [42, 71], are also applicable to compartmentation at the same time. However, these approaches are strict, and are not practical in many environments. For example, if they are applied in an operating system environment, the operating system becomes an isolation mechanism, rather than a sharing mechanism. Domain type enforcement [72, 73] attempts to address these issues. It is designed to ensure the integrity of a system and to provide compartmentation, but only from the aspect of a single domain, whether professional, personal or public life. However, it is not clear how labeling can be achieved in the case of more than one domain; applications and data need to be labeled in a uniform manner, as it is necessary to be able to relate them to a particular domain. In a similar way, a distinction needs to be made between a single type of data which originates from different domains.

3.4 Use Cases of Ubiquitous Social Systems

In this section, typical use cases of ubiquitous social systems are presented. In addition to being useful as examples, these use cases will be essential for the various evaluations in the subsequent chapters. The use cases include general and simple scenarios originating from the area of enterprise mobility. Similar use cases have been described in several works [4, 65, 74] and are encountered in common practice. These use cases are further enriched with the characteristics of IT consumerization, which are a major influence on security. In addition, they take into account the security guidelines for enterprise mobility, as presented in standardized security documents [52].

Each use case centers around a single protagonist, referred to here as the user. A user owns a mobile device and uses it for both personal and work purposes. Usually, she uses the same consumer-oriented applications for both of these purposes. Furthermore, it is assumed here that a context-aware security system is installed on each mobile device. This security system enables the specification and enforcement of security policies consisting of security context, and is responsible for protection of the resources stored or used via the mobile device. Each user is required by her employer to install the company’s security policy onto the security system, in order to be allowed to use her mobile device at work. In this way, the company is assured that its resources are protected according to its requirements. In similar manner, each user specifies a security policy for protection of their personal assets of interest. In addition, it is assumed here that the security system can identify the purpose for which the user uses an application; for example, it can identify whether the user uses a *camera* application for personal or for work purposes. According to the security policies, the security system generally only allows the user to operate applications for a single purpose at any given moment.

3.4.1 Use Case 1

Alice is working as a finance manager at the ACME1 company. ACME1 is a traditional company, which tries to follow modern technological trends by allowing its employees to use their own mobile devices for work purposes. However, it is still conservative in this regard, and requires its employees to use their devices exclusively for work purposes while at work. This means that, while at work, employees can operate only apps approved by company policy. Employees may have other apps installed on their devices, but may not use them while working. For example, Alice cannot use intelligent assistant applications at work, since such applications send audio recordings to remote servers and can potentially leak confidential company conversations. In addition, ACME1 employees are not allowed to access work documents outside of the company's premises.

Scene 1. Alice is in her office and starts her working day. She is going to prepare a finance report for the past week. The *Office* app has been her favorite word processing application for years. Following her recommendation, ACME1 has approved this app for work purposes. After starting the application, Alice creates a new document for the report and starts writing. During this time, the *Office* app collects usage statistics and tries to send them to a remote server, without Alice being aware of this. Such activity should not be allowed, as the app may also leak the confidential company document that Alice is writing. After a couple of hours, Alice finishes the report and closes the document. She decides to check a personal document with the same app, but she is unable to open it. ACME1 allows her to open only work documents while using *Office* for work purposes.

Scene 2. Alice is taking a coffee break with her colleague Adam in the company's kitchenette. She asks him to send her some of the previous finance reports, in order to compare them with the one she has just finished. They use the *FileManager* app to do this, for example through the "send via wireless" option. The transmission is unencrypted, since both Alice and Adam are located inside the company premises. In order to be able to read the documents, Alice needs to install an appropriate reader application. She connects to the access point located in her office, selects one of the applications approved by ACME1, and installs a new application on her mobile device, which will be used for work purposes.

Scene 3. Alice is in her office and wants to call home. Since ACME1 prohibits this, she is not able to use the *Voip* app for personal purposes while at the office. The conversation is important to her, so she decides to leave the company premises in order to make the call. She logs out on the access control device at the entrance and leaves the building. According to the company policy, all applications she has been using during work are then closed automatically. Outside, Alice is able to use the *Voip* app for personal purposes, and she calls home.

Scene 4. Alice is on the way home and uses the *Assistant* app. She finds this intelligent assistant application very useful, especially while walking through the city, because she can operate it using voice commands. When she passes the restaurant where she usually has lunch, she asks the *Assistant* app about the menu for the following day. The app promptly answers her question, after consulting a certain service on a remote server. During this, the *Assistant* app is also able to leak a company document to the server, since it was given permission to access the entire device storage during installation time. Such activity should be denied. In front of her apartment, Alice uses her phone to unlock the door. As soon as she enters her home, she starts her favorite song using the *Player* app, which automatically connects to the music system. Alice is still thinking about the results from the report she created earlier that day. She decides to check these again, but is not able to open the report. According to company policy, employees can access work documents on their mobile devices only while they are on company premises.

3.4.2 Use Case 2

Carol is a computer science researcher at the ACME2 company. ACME2 is an innovative company which considers its employees' wellbeing to be crucial to its success. It therefore not only allows its employees to use their own devices for work purposes, but allows them to operate their devices as they wish. Nevertheless, it still imposes some restrictions; for example, Carol can chat with her friends while at work, but cannot make voice calls to them, as this could potentially leak confidential company conversations. ACME2 provides two networks for its employees, one intended for work and the other for other purposes. In this way, it can keep apps which have not been approved for work purposes out of the corporate network.

Scene 1. Carol has just arrived home from her office. She is excited, as she has finally achieved promising results from her experiments earlier in the day. She decides to check these again, but this time in a more relaxing environment. She opens the document with the results using the *Office* app and goes through it carefully. She is able to do this, since ACME2 allows her to use this app for work purposes while at home. Afterwards, she calls to her dog and takes a photo of it using the *Camera* application.

Scene 2. Carol is at the office, preparing for a business trip. She has arranged a project meeting the following day at an associate's site, and has already booked a flight for herself and her colleague Clark for later that day. She still needs to arrange transportation to the airport and decides to try a new and popular city transportation service. The service requires customers to submit a trip request using an application called *Cab*; the request is then routed to the drivers, who use their own cars for transportation. More specifically, customers provide the desired source and destination of the trip to the app, as well as an optional arrival time, and the app notifies them of their specific pick up time. ACME2 does not restrict its employees in terms of methods of travel on their business trips; however, it has not approved the *Cab* app for work purposes. Thus, prior to Carol using this app, all work apps need to be closed, in compliance with company policy. Following this, Carol opens the *Cab* app and submits a request with the required trip information, after which she receives her pick-up time.

Scene 3. Carol is in the cab, travelling with her colleague Clark to the airport. Although she generally does not share personal information with her colleagues, she decides to make an exception this time and shows Clark some photos of her new pet. She opens the *Camera* app on her mobile device and explicitly allows it to show a couple of personal photos to her colleague on the screen. Carol and Clark are looking at the selected photos and are talking about her pet.

Scene 4. Carol and Clark are at the airport, reviewing the slides for the meeting using their mobile devices. Carol sends Clark the latest version of the slides over a wireless channel, using a file manager application. Usually, the transmission of unencrypted company documents between colleagues is allowed on company premises, as long as only company employees are present. However, at this moment many passengers are present in the hall, and some of them may be trying to capture an unencrypted communication. A secure channel between Carol's and Clark's mobile devices must therefore be established prior to the transmission.

3.4.3 Use Case 3

Bob is a commercial manager at the ACME3 company. ACME3 is a modern company, which tries to be as "virtual" as possible. It lacks premises or equipment in a traditional sense, and all its servers and systems are cloud-based. Most of its employees are mobile workers and are required to be always available. Bob enjoys working outside of an office, as

this offers him more freedom and gives him the ability to manage his work more efficiently. However, he does not enjoy being obligated to receive work mails or calls at any time of the day, especially when he is at home. For company meetings, ACME3 usually rents conference rooms located in some public venue.

Scene 1. Bob is working somewhere in the city. He has arranged a meeting for later that day, so he checks the details of the meeting from an older email. He sees that the meeting is taking place at a public venue which has recently opened in the city. Bob reads his work emails using his favorite *Email* app, installed on his device for work purposes. In compliance with company policy, the *Email* app connects to the company mail server using a secure channel.

Scene 2. Bob is in his car, driving towards the public venue where the company meeting is being held. He mounts his mobile device on the docking station and opens an application called *Traffic*. This is a navigation app with advanced features, which is able to inform its users about traffic jams throughout the city and find alternative routes to a destination if necessary. Although Bob finds this app very useful, ACME3 has not yet approved it. Therefore, Bob cannot use apps intended for work purposes while using the *Traffic* app. Nevertheless, it is assumed here that some apps for work purposes, such as *Voip*, can operate in the background, without directly communicating with Bob, for example through the screen or the microphone. Bob is already near the venue when a certain *Notifier* app alerts him that a colleague has just called him using the *Voip* app. Bob stops the *Traffic* app and calls his colleague back using the *Voip* app. Since he is driving, he turns on the hands-free option. He arrives at the public venue's parking place, still continuing the conversation. As he gets out of his car, he needs to turn off the hands-free option and use the in-ear speaker instead. In this way, people nearby will be prevented from hearing the whole conversation.

Scene 3. Bob is in a conference room in the public venue where the meeting is being held. He is about to present his slides to his colleagues, using his mobile device. He activates the slideshow from the *Office* app, and this activates transmission to the projector. Since there are people in the public venue, outside the conference room, who are not members of the ACME3 company, a secure channel with the projector is first established. During the presentation, a security guard suddenly enters the conference room, as part of his regular patrol. Since he is not employed at ACME3, he is not allowed to see the confidential company data on the screen. The presentation should therefore be terminated automatically.

Scene 4. Bob is in front of his house, coming back from work. He uses his mobile device to unlock the door. After entering, he opens the *Video* app and starts watching a movie. The movie is streamed from his mobile device to the TV set by the same app. While he is at home, several applications for work purposes are still operating in the background, although without directly communicating with him. An example of such an application is the *Email* app, which continues to receive incoming emails. In the middle of the movie, the *Notifier* app signals Bob that he has received a new work email through the *Email* app. He pauses the movie and opens this app to read the mail.

Chapter 4

A Conceptual Model of Security Context

Security context has not been as widely discussed as context in general, although many context-aware security approaches have been proposed in the past. This chapter focuses on providing an explicit and clear definition of the concept of security context. Firstly, a review is presented of related works which define and describe the security context; following this, a working definition of security context is proposed. Next, a conceptual model of security context is proposed. This conceptual model identifies important concepts of security context at different levels of abstraction. The applicability of the model is evaluated by applying it to the use cases of ubiquitous social systems and comparing it with existing approaches.

4.1 Related Work

Very few works have explicitly defined security context. Johnson [75] combined the acknowledged definitions of context and security, and defined the security context as any information which characterizes the situation of an entity and has security implications. As an entity this author considers the user, the user's computing device and the surrounding environment. Mostefaoui [34] defines the security context as a set of information from the user's and application's environments that are relevant for the security process, describing it further as the state of these environments requiring security interventions. In addition, Bandinelli et al. [30] are concerned only with contextual information which originates from the communication between entities and which is useful for providing end-to-end security.

Existing definitions of the security context are incomplete and do not represent the full nature of the context. They specify information about only part of the relevant entities, namely the user and the particular device that hosts the context-aware system. This is insufficient for ubiquitous social systems, for which contextual information about other surrounding entities (people and devices) also needs to be taken into consideration. Moreover, current definitions describe only the static nature of the context, omitting the concept of activities which guide the creation of the security context.

A few works have also tried to describe the security context more precisely using taxonomies. Johnson [75] classifies the security context along two dimensions—the relevant entity and the affected security objective—and provides simple guidelines for identifying the relevant security context. Evesti and Panssar-Syväniemi [76] define concepts important to the security context and classify them into three groups: (i) the situation context, which contains concepts describing the usage of applications; (ii) the digital context, which contains concepts describing the surrounding environment; and (iii) the physical con-

text, which contains concepts describing the execution platform. Cuppens and Cuppens-Boulahia [25] recognize several types of security context in their taxonomy: the temporal; the spatial; the user-declared, which is related to the user's intention; and the prerequisite and the provisional contexts, which are related to the preconditions and obligations for activities. In addition, Bandinelli et al. [30] divide the possible security context into several categories depending on their origin: user, device, communication and application.

Current taxonomies do not appear to be very helpful for the specification of security context. Although their categories describe important concepts within the security context, they do not provide a means of relating these concepts to security services. Moreover, they rarely take into account the associated social aspects. Only a few of them [25, 76], try to encompass social concepts such as the role of the user and environment and the objective of the user, and this is done in an arbitrary way, without any attempt to define these concepts in a consistent manner.

4.2 Definition of Security Context

The definition of security context given in this work combines the notions of security and context. It is based on one of the most widely accepted definitions of context, that given by Day et al. [54]. It also emphasizes the dynamic nature of context and the role of activities. For clarity, no explicit definition of security is involved. Security is defined as the protection of resources in order to attain objectives of confidentiality, integrity and availability [39, 40]. The definition of the security context is given below.

Definition 4.1. *Security context* is a set of contextual information considered relevant for the process of security, regarding a particular task or activity.

Definition 4.2. *Contextual information* is any information that can be used to characterize the situation of a person or a device, which can be seen as a composition of its resources.

In an analogous way to systems in general [19], security systems can utilize the security context using two approaches. Systems which can present only information about the security situation of the resources of interest, depending on the current security context, can be referred to as passive context-aware security systems. On the other hand, systems that automatically adapt their behavior during runtime in accordance with the discovered security context, in order to provide a more relevant service, are called active context-aware security systems. In the rest of this paper, the term 'context-aware security systems' is used to refer only to active systems.

4.3 Definition of the Conceptual Model of Security Context

A conceptual model of security context is proposed here in order to define and describe this concept more precisely. The conceptual model identifies both the important concepts that constitute the security context and the relationships between them (Fig. 4.1). Many of these concepts are derived directly from the motivation scenario in Section 3.4. For example, there are persons (e.g. Alice and Bob) who use devices (e.g. their mobile devices) to perform various activities (e.g. share work documents) which are more precisely described in terms of the persons' intentions, social relationships and surrounding environment (e.g. sharing work documents with a colleague at a public event while working).

The conceptual model below was introduced in [77]. A refined version, with clarified definitions of the identified concepts, is presented here. The concepts involved are separated

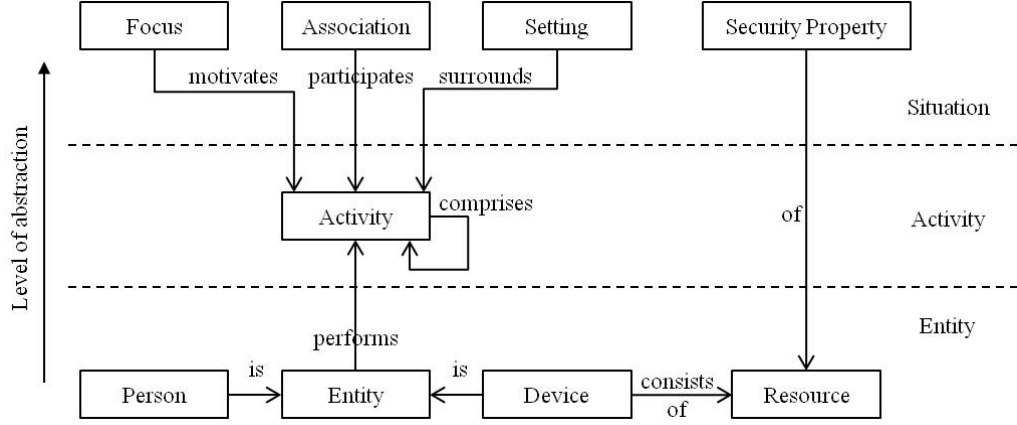


Figure 4.1: A conceptual model of security context.

into three layers: entity, activity and situation (Fig. 4.1). Each of these has a different level of abstraction from the perspective of a context-aware system; concepts from lower layers are used to derive concepts from higher layers, and thus need to be determined first. In practice, this represents the way in which a system becomes aware of the context in which it operates, through the process of context reasoning.

4.3.1 Entity

Definition 4.3. An *entity* is an actor which performs an activity.

The model centers around two types of entities: person and device.

Definition 4.4. A *person* is a human which performs activities in order to achieve certain goals.

Social scientists have observed that during their lives, people do not act simply as individuals in an undefined manner, but take part in various social groups (communities) in order to achieve their goals more easily [78, 79]. The common goals and interests of these groups shape the activities performed within them. Group members divide these activities among themselves according to the qualifications needed to accomplish them successfully, and this is represented as the formation of roles within the social group. These roles and the distribution of activities mean that social groups are highly organized, and depending on these goals and roles, social groups can be categorized into social domains. Examples of social domains include family, friends, workplace, education, health care, a research department, a city, a country etc., while examples of social groups include their specific realization. The social domain of a research department, for example, is defined by roles such as the head of the department, researcher, student or developer, and common goals such as solving challenging scientific problems, proving the feasibility of proposed solutions, etc. Since people are often part of a single social group within a social domain, social groups are referred to in the remainder of this thesis by their social domains, in order to clarify these aspects as far as possible. For example, rather than referring to a particular family as a social group, it is referred to as a family group from the perspective of a particular person.

Definition 4.5. A *device* is a physical object with computing, communication and storage capabilities.

Devices perform activities for people and help them to be more efficient and productive. Some devices perform important activities which are useful for a number of people within a social group. For example, a printer makes a persistent representation of documents on paper for all the members of a research department social group; a network access point enables the members of a family to connect to the Internet. Therefore, such devices can also be considered as being members of social groups, and can have assigned roles that stem directly from the purpose of their use. These devices and the people involved are referred to as *social entities*. Other examples of devices which are social entities include payment terminals, access control devices, attendance devices, information devices etc. On the other hand, a smart mobile device such as a smartphone or a tablet performs activities and is useful only to a single person. It cannot therefore be considered a member of a social group; instead, it is more like a surrogate, which represents the person operating it within the social groups of which s/he is a member. From the perspective of a ubiquitous social system, the mobile device hosting the system is referred to here as the host, and the person operating this device as the user.

The set of all entities, $E = P \cup Dev$, is denoted as the union of the set of persons P and the set of devices Dev . Two entities are of particular importance for this discussion: the host, $h \in Dev$, and the user, $u \in P$. An entity which can be a member of a social group is referred to as a social entity, $se \in SE$, $SE = P \cup SDev$, where $SDev \subseteq Dev$. Let G be a set of possible goals that can result from activities and Rol be a set of possible roles that can represent the qualifications needed to successfully accomplish activities. Then, the concepts of social domain and social group can be defined as follows.

Definition 4.6. A *social domain*, sd , is defined as a tuple, $sd = (gls, rls)$, of specific goals, $gls \subseteq G$, the achievement of which is a reason for social entities to organize themselves (in the form of social groups), and a set of specific roles, $rls \subseteq Rol$, according to which activities that result in goals gls are divided between those entities.

Definition 4.7. A *social group*, $sg \subseteq SE \times Rol$, is a set of tuples, (se, rol) , of social entities se and their assigned role rol , such that all roles are part of a single social domain sd .

The sets of all social domains and groups are represented as SD and SG .

Furthermore, a device can be seen as a composition of resources.

Definition 4.8. A *resource* is something that can be used for processing information.

It is assumed here that each resource is related to only a single social group which is assigned to it upon its creation. This social group is referred to as the origin of the resource. The *origin* denotes the purpose of the resource, i.e. that the resource is intended to be used for achieving the goals of a particular social group. In addition, origin can be used as a label in order to refer to resources more conveniently. For example, a resource whose origin is the work social group can be referred to as a work resource.

There are three types of resources in the proposed model: method, data and channel¹. If the set of methods is denoted as M , the set of data as D , and the set of channels as C , then the set of all resources can be denoted as their union, $Res = D \cup M \cup C$. Each of the resource types are defined and described below.

Definition 4.9. A *method* represents a software component which implements a certain functionality for information. It can be executed through its interface (API).

Methods are usually combined in *applications*. In practice, applications need to be executed first, in order for their methods to be available.

¹Hardware or their capacities are not treated as resources. In a way, these are included in the concept of device.

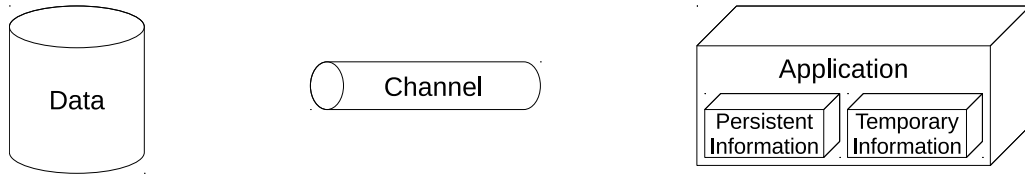


Figure 4.2: Types of resources.

Definition 4.10. An *application* is a software component which enables the achievement of complex functionalities for information.

Applications can be understood as being composed of two information containers. One of these keeps the specific instructions which guide the performance of the particular functionalities (persistent information), whereas the other keeps additional information needed during the execution of the application (temporary information). The latter is saved in the memory of the device only during the execution of the application. An example of an application at operating system level is an executable file or its execution in a form of a process. In addition, *security mechanisms* are recognized here as methods and applications whose functionality is related to the protection of other resources.

Definition 4.11. *Data* is a persistent storage of information, saved for future use.

Data is understood here as a single information container (Fig. 4.2). Generally, the concept of data refers to information in general, including information transferred through channels or contained within applications during execution. However, the concept of data used here is more specific, and does not refer to such information. An example of data at operating system level is a file or a directory. A person can access and consequently consume information from this data only through a specific application, which can interpret the encoding of the information. For example, a person can read text data using an office application, or listen to audio data using a music player application. Data is a shared information storage and can be accessed by different applications, even at the same time.

Definition 4.12. A *channel* is a pathway through which information is exchanged, either on a single device or between two devices.

A channel can also be understood as a single information container. Channels are built using a particular component through which information is exchanged in form of (encoded) messages. Channels are used for communication between the user and any other device with an application at the host, or between two applications at the host. Channels are temporary containers, meaning that messages are kept only until they are transferred (Fig. 4.2). An example of a channel at operating system level is a socket, a named pipe, or a special file used for exchange through a sensor. Three types of channels are identified here: sensors, networks and internal channels. Sensor channels are built using various device sensors (e.g. touch screen, microphone, camera or speakers). These channels are used for user-application and device-application communication. Network channels are constructed using available network components in the devices (e.g. Wi-Fi or Bluetooth cards). These channels are used only for device-application communication. Internal channels take place inside the host; these channels are used for application-application communication.

Entities can be described using various items of contextual information, which can be used for their recognition and sensing within the ubiquitous social system. This information can be acquired from both physical and virtual sensors [20]. Physical sensors are hardware components such as GPS, camera, microphone, accelerometer and many others, which are

already present in contemporary mobile devices; virtual sensors are software components which provide contextual information taken from various sources such as applications, protocols, social networks etc., either from the host or from an external context provider. Persons are usually described in terms of their fingerprints, voice, face, age, gender, social relationships or political or religious views, whereas devices are usually described in terms of their manufacturer, component IDs, operating system, available services and protocols etc. Persons can also be described in terms of their mobile devices, since it is these devices through which people access ubiquitous social systems. In addition, resources can be described using contextual information such as owner, origin, time of creation or modification, structure, protocol, security mechanisms used etc.

Finally, the existence of three special social groups is assumed in the proposed model: public, empty and system. A *public* social group is a general social group which consists of all social entities, whereas an *empty* social group is a social group which does not contain any members. These social groups are important for a precise determination of the social concepts introduced later in the model. In contrast, a *system* social group is a particular social group which consists of a single social entity, i.e. the user. This social group is especially important for relating the resources from the underlying operating system to the host, such as kernel applications, driver applications, configuration data etc. It is considered here that all these resources have the system social group assigned as their origin.

4.3.2 Activity

Definition 4.13. An *activity* is a process in which an entity executes an operation on a resource, in the pursuit of certain goals.

An activity is performed either explicitly by the user, using an application while interacting with the host, or implicitly by the host itself. An activity whose goals are related to the protection of resources is called a *security activity*.

Some activities are more complex than others and comprise several activities that are performed consecutively. An activity is represented in the proposed model as a tuple of tuples (app, op, obj) , where $app \subseteq M$ is the application used to perform the activity, $op \in M$ is the method that implements the operation of the activity, and $obj \subseteq Res$ are the resources upon which the activity is performed, i.e. the objects of the activity. The set of all activities is denoted as A . For clarity, in the remainder of this thesis activities are represented with their description in words. For example, an activity described as “Office app reads data d_1 ” is represented as a tuple $(Office, read - data, d_1)$.

As an example of a complex activity, consider an activity described as “Office presents data d_1 ”, which involves several other activities: “Office reads data d_1 ”, “Office opens channel c_1 ”, and “Office shares screen through channel c_1 ”. Similarly, a complex security activity is “ app_1 establishes confidential channel c_1 ”, which is comprised of several (security) activities that are executed during the SSL protocol handshake.

4.3.3 Situation

The concept of situation interprets an activity more effectively and gives it more meaning. A *situation* describes the state of the relevant entities and the relationship between them [36, 37]. Several concepts suitable for representing a situation are defined. These are separated into two groups: social concepts and properties.

4.3.3.1 Social Concepts

Based on the notion of social groups, three social concepts can be defined: focus, association and setting.

Focus describes the motivation of the user while performing an activity. As explained earlier, people perform activities in order to achieve certain goals. Since each goal is from a particular social group, each activity a person performs is related with a particular social group. However, people often consecutively perform activities related with different social groups in ubiquitous social systems. Some goals related with these activities are more important than others at a given moment and are therefore referred to as primary.

Definition 4.14. *Focus* is a social group, $focus \in SG$, whose goals are considered primary for the user at a given moment.

Focus describes the primary intention and orientation of the user and motivates the user's behavior. The focus of the user is the focus of each activity the user performs at a given moment.

An *association* describes the participants of an activity. A *participant* is a social entity which is involved in an activity through consuming or generating resources in the host. In the former case, participants consume applications and data through some of the output channels of the host (e.g. a screen or speaker), whereas in the latter case, participants generate data through the host's input channels (e.g. touch or microphone). Since for some activities, the number of participants can be large, it is impractical to represent these individually. Instead, the notion of association is introduced for this purpose.

Definition 4.15. An *association* is a social group which is common for all participants in an activity, which describes them appropriately.

In order to define an association more precisely, the following notation is used. Let $par_a \subseteq SE$ be the set of participants in an activity a and let $se_g \subseteq SG$ be the set of social groups of which a social entity se is part. Also, let $common : 2^{SE} \rightarrow 2^{SG}$ be a function that maps an arbitrary set of social entities $B \subseteq SE$ to the intersection of social groups of which they are all part, $common(B) = \{sg \in SG \mid \forall se \in B (sg \in se_g)\}$. Then, the association of an activity a , $assoc_a \in SG$, can be defined as a common social group of a non-empty set of participants in that activity, $assoc_a \in common(par_a)$, $par_a \neq \emptyset$. In the case of a single common social group, the association is uniquely determined. However, in the case of more than one common social group, the association is determined as that which most suitably describes the participants. For activities with a single participant, the association is considered to be the focus of the participant, $assoc_a = focus_{par_a}$. If the participant is the user, then the focus is determined by the host itself. Otherwise, it needs to be somehow provided by the participant (or by the participants mobile device if the participant is a person)². For activities which involve several participants, their association can be determined as the smallest common social group, i.e., the social group with the lowest number of members. Since this social group is the most specific of the common social groups, it is considered the most suitable for describing the participants. In addition, if there are no participants in an activity, the association for this is considered to be empty, $assoc_a = empty$. It is assumed that empty is a specific social group which does not contain any members.

Different activities have different entities as participants. Persons can participate in an activity by directly interacting with the host through a sensor channel. By default, it is assumed that the user is the participant in any such activity. Other persons can also

²The existence of a certain protocol for achieving this can be imagined.

participate in activities indirectly, using their mobile devices. In this case, the person's mobile device further interacts with the host via a network channel. On the other hand, devices always participate in activities by directly interacting with the host, either through a sensor or via a network channel.

A *setting* describes the observers of an activity. Activities are performed within various environments, which can change dynamically in ubiquitous social systems. In addition to participants, these environments can involve many other social entities, referred to as observers. An *observer* is a social entity which is nearby the host or the participants in the activity and which can monitor the activity being performed. The concept of 'nearby' means here that the observer is either physically located in the vicinity of the host or the participants, or is part of a channel between them and takes part in enabling their communication during the activity. The aspect which distinguishes observers from participants is the ability to access the host's resources during an activity; only participants can access these resources, while observers cannot.³ In a similar way to participants, observers are more easily described as a group than individually.

Definition 4.16. A *setting* is a social group common for all observers of an activity, which describes them appropriately.

In order to define setting more precisely, the same notation is used as for association. Let us further denote the set of observers of an activity a as $obs_a \subseteq SE \setminus \{u\}$. The setting of an activity a , $sett_a \in SG$, is defined as a social group common to the participants and a non-empty set of observers of that activity $sett_a \in common(par_a \cup obs_a)$, $obs_a \neq \emptyset$. In the case of a single common social group, the setting is uniquely determined. However, in the case of more than one common social groups, the setting is determined as that which most suitably describes the observers. In a similar ways to association, setting is determined in this case as the smallest common social group, i.e., the social group with the lowest number of members. In addition, if there are no observers for an activity, it is considered that the setting for this is the empty social group, $sett_a = empty$.

4.3.3.2 Properties

A *property* is a quality that describes a resource or its usage. A property pr is defined as a function that maps a resource $r \in Res$ to a certain property value, $prv \in PrV$. In its simplest form, it is a function $pr : Res \rightarrow \{0, 1\}$. In this case, a value of 1 denotes that the resource has the property, or that the property is achieved, whereas 0 denotes the opposite. The set of all properties is denoted as PR .

Of all possible properties, the security properties of resources are of particular importance for this discussion.

Definition 4.17. A *security property* is a quality that describes a resource or its usage, with respect to the objectives of security.

Generally recognized security properties are confidentiality, *conf*, integrity, *int* and availability, *av* [40]. In addition to contextual information, the values of security properties of resources should be understood as an indicator of the way in which a security system operates. However, their evaluation should not be seen as verification that security services, or their implementation, provide the particular security properties. For an activity a , the current values of the security properties of all resources that are of interest and are

³However, observers can easily become participants in an activity if they intercept the communication between the host and the participants. For example, an observer can become closer and intercept a communication taking place via a sensor channel, or can understand an unencrypted network communication passing through it.

involved in a comprise a tuple, $sprop_a = (pr_1(r_1), pr_1(r_2), \dots, pr_n(r_m))$, where $pr_i \in PR$ and $r_j \in Res$.

4.3.4 Summary

In summary, the security context of a particular activity a is defined as follows.

Definition 4.18. The *security context* of an activity a is a tuple, $ctx_a = (a, focus, assoc_a, sett_a, sprop_a)$, consisting of the activity itself, the focus of the user, $focus$, the social concepts that characterize a —association, $assoc_a$, and setting, $sett_a$ —and the current values of the security properties of resources that are currently involved in a .

The set of security context of all activities A is denoted as Ctx .

4.4 Evaluation of the Conceptual Model of Security Context

The applicability of the proposed model is demonstrated here in an empirical manner. Firstly, it is used to analyze the use cases from Section 3.4. Then, it is compared with the existing approaches, and particularly their understanding of the security context. In addition to verifying the ability of the model to represent the security context, this comparison is also an evaluation of the understanding and specification of the security context within the existing approaches. Finally, the security requirements of the security context are presented.

4.4.1 Model Application in Use Cases

Before using the proposed conceptual model of security context in analyzing the use cases, several assumptions are made. Firstly, it is assumed that the main protagonists in the use cases, that is Alice, Bob and Carol, are the users, and that their mobile devices are the hosts of the ubiquitous social system in each case. It is assumed that each user is a member of four social groups, referred to as work, home, public and system. It is further assumed that hosts retain information about the structure of each of these social groups, and in particular about the members of these groups and their roles. As explained above, the public and system social groups are same for all of these; they contain all the entities and the specific user, respectively. In contrast, work and home social groups differ, and their members are described below for each of the use cases.

Furthermore, it is assumed that each social group has its own set of applications, which are exploited by the user to achieve the goals of the group. This means that if the user wants to use an application within a particular social group, the application first needs to be installed for the purposes of that social group. If the user wants to use the functionality of an application in several social groups, this application needs to be installed for each of these social groups.⁴ This results in several applications with the same functionality being installed on the host. For example, if the user wants to use a particular *Launcher* app for starting applications in the work, home and public social groups, this app needs to be installed three times. Following installation, there will be three applications on the host, *Launcher_W*, *Launcher_H*, and *Launcher_P*, which provide the same functionality.

Next, it is assumed that each host runs a context-aware security system which is based on the proposed model of security context. For convenience, this system is referred to as the

⁴The entity installing applications is not addressed as part of the proposed model. Clearly, one approach is that the user would be responsible for this. However, it can easily be imagined that a more automatic approach is possible, which could take place during the initialisation of the social group.

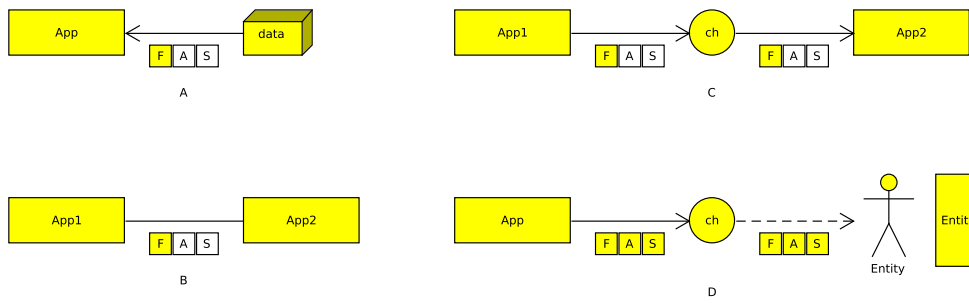


Figure 4.3: The association and setting during activities. The yellow color indicates that the focus (F), association (A) and setting (S) are a specific social group (e.g. work social group) and that the resources originate from this social group. The white color denotes that the association and setting are empty.

CASS. In terms of the proposed model, the CASS is an application which originates from the system social group. The CASS enables the specification and enforcement of security policies applicable to ubiquitous social systems. It monitors each resource and intercepts each request for an activity. It first examines the security context of the requested activity and then decides whether to allow or deny this, on the basis of the specified security policies. It is assumed that a security policy specified at the CASS for each social group (e.g. the company (work) policy, home policy, public policy).⁵ For some activities, the CASS continues to monitor their security context; if the context becomes inappropriate according to these policies, the CASS terminates the activity.

Then, it is assumed that the association and setting for activities that take place within the host are empty. This applies to activities involving data (Fig. 4.3 a) and application objects (Fig. 4.3 b), as well as internal channels (Fig. 4.3 c) which are used for communication between two applications. This assumption is intuitive, since these activities have no participants and observers; no social entity is capable of directly finding out that they have been performed. In fact, it is considered that only activities involving sensors and network channels have an association and a setting which is other than empty (Fig. 4.3 d).

It is also assumed that system applications are used for the purposes of all the social groups within the ubiquitous social systems. System applications are part of the host's operating system and are specifically installed and executed in order to enable other, user installed, applications to use the host's hardware in a convenient manner. This is especially notable in activities which involve network (e.g., WiFi or 3G) and sensor (e.g., screen or touch) channels. As an example, consider the case of the *Launcher_W* work application attempting to send information to the screen through the channel *scrCh* (Fig. 4.4 a). The screen channel *scrCh* is in fact a channel to the *ScreenDriver* system application, which is responsible for writing information to the screen component of the host. In this case, the *Launcher_W* app writes information to the screen channel, and the *ScreenDriver* reads this information and displays it on the screen (Fig. 4.4 b).

Finally, it is assumed here that the CASS is limited to reasoning only about the properties of confidentiality and integrity of network channels. For example, these properties can be achieved using the SSL/TLS protocol, that is, by establishing a secure channel. This simplified assumption is made in order to show the applicability of security properties as part of the security context. Nevertheless, it should be noted that a system cannot be

⁵Security policies are addressed in the next chapter.

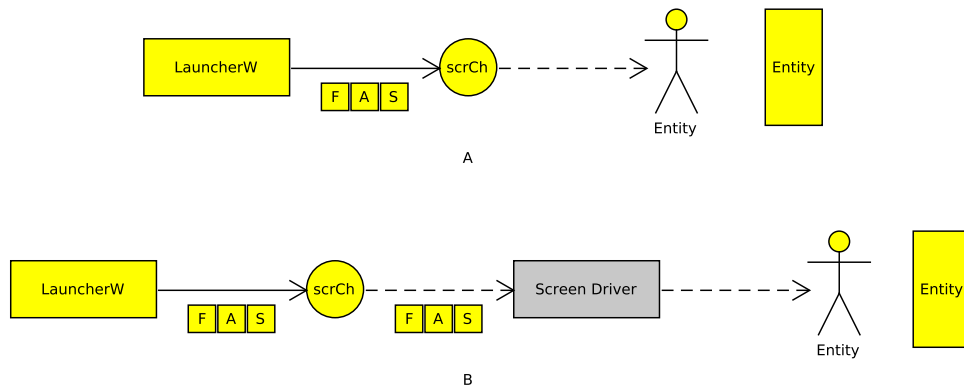


Figure 4.4: Activity involving sending information to the screen. The yellow color indicates that the focus (F), association (A) and setting (S) are a specific social group (e.g., the work social group) and that the resources and entities originate from this social group. The gray color indicates that the resource originates from the system social group.

secured solely by ensuring these security properties. For example, in order to achieve confidentiality and integrity for a channel with the SSL/TLS protocol, the authenticity of the user and participant needs to be confirmed in advance. In addition, it is assumed that confidentiality and integrity cannot be achieved for sensor channels (e.g. speaker, microphone or screen channel), since security mechanisms (e.g. network protocols, authentication and access control mechanisms) do not apply when securing these types of physical phenomena (e.g. voice data or images).⁶ It is therefore considered that these properties have a value of “-1”.

Below, the proposed conceptual model of security context is used to practically analyze activities and their security contexts. In order to facilitate the readability of the thesis, only Use Case 1 from Section 3.4 is analyzed here; the remainder of the use cases are analyzed in Appendix A. During the analysis presented here, one possible operation of the context-aware security system (CASS) is described which utilizes the proposed model. In particular, the way in which the CASS reasons about the security context of activities and uses it to control them is described. An attempt is made here to give a representative example; therefore, activities and security contexts that repeat or are similar to those already described are not discussed in further detail. Appendix C lists all the activities performed during the use cases and their security contexts.⁷

Only four types of activities are described during the following analysis: create, read, write and execute. The meaning of these activities is given in Table 4.1. Note that Appendix C lists several further types of activity which are not included in this analysis. These are defined in the next chapter, within the proposed policy model. They require a deeper explanation which is beyond the scope of the current chapter; these types of activities and the way in which their context is reasoned is therefore addressed in the next chapter. However, Appendix C is useful for a fuller understanding of this evaluation, since it lists the chronological order of activities according to the use cases and their security context.

In addition, in the analysis below, applications are labeled according to the social group for which they are installed. Thus, applications for work, home and public social groups

⁶In these cases, simply terminating an activity protects the information flowing through the channels.

⁷In practice, it lists all activity requests during the use cases.

Table 4.1: Description of the types of activity included in the use case analysis. A description is given for the type of resource on which each activity is performed.

Type of activity	Description
create	Data, channel, application: A new resource is created.
read	Data: Information is read from a resource. Channel: Information is received at a resource.
write	Data: Information is written to a resource. Channel: Information is sent through a resource.
execute	Application: A resource is started and its functionality is available.

are labeled as W , H and P , respectively. Note that applications for the system social group are not labeled. Table 4.2 describes the functionality of the applications used within the use case analyzed.

Table 4.2: Description of functionality of applications used in Use Case 1.

Application	Functionality
Launcher	Starts an application following a user request.
Installer	Installs an application from an app store.
Network Manager	Connects to a device from a network infrastructure, through the host's wireless capabilities (e.g., WiFi, 3G, etc).
NFC Manager	Connects to a device through the host's NFC capability.
Office	Enables word processing of various office documents.
File Manager	Enables the management of files and folders stored in the host's memory.
VoIP	Enables voice communication between users over the Internet.
Assistant	Answers questions, makes recommendations and performs actions following a user request.
Player	Plays audio files.

4.4.1.1 Use Case 1

Alice's work and home social groups are defined as follows. Her work social group represents her employer, i.e. the ACME1 company. This social group consists of all persons employed by the company and the various devices owned by it. This includes Alice, Adam, access points, routers, printers, company servers, an access control device at the entrance of the company premises etc. Alice has the role of finance manager within this social group. The notation $work = ((Alice, finance\ manager), (Adam, accountant), (AP_{W1}, access\ point), (R_{W1}, router), (P_{W1}, printer), (S_{W1}, server), \dots)$ is used here. Her home social group represents her apartment; this social group consists of Alice, her daughter and all their home appliances, such as her TV set, access point, door locking system, refrigerator etc. Alice has the role of resident within this social group. The notation $home = ((Alice, resident), (AP_{H1}, access\ point), (TV_{H1}, TV\ set), (RF_{H1}, refrigerator), (DL_{H1}, door\ lock), \dots)$ is used here.

Scene 1. Alice is in her office and her focus is her work social group. The CASS determines this from Alice's previous activities. The ACME1 company has an electronic access control system installed at the entrance to its premises, and all its employees must authenticate themselves in order to enter or leave the building. In addition to logging their

movements, the access control system serves as a time monitoring service at the same time: while the employees are located within the building, they are considered as being at work. Alice authenticates herself using her mobile device when she enters the ACME1 premises in the morning. The CASS captures this activity and sets her focus to her work social group. This social group stays as Alice's focus for all the activities she performs in this scene.

Alice wants to prepare a report using the *Office_W* app. This is a work application, which is installed on her device solely for the purposes of the work social group. Alice starts this application using the *Launcher_W* app as follows. Firstly, the *Launcher_W* app sends a list of available apps to the screen through a channel *scr_{W1}*⁸, which initiates an activity a_1 : *Launcher_W writes scr_{W1}*. Next, Alice taps the *Office_W* app icon on the screen, in order to start this application; this initiates an activity of the *Launcher_W* receiving an input from a touch channel, *tch_{W1}*⁹, a_2 : *Launcher_W reads tch_{W1}*. Finally, the *Launcher_W* app executes the *Office_W* app, which initiates an activity a_3 : *Launcher_W executes Office_W*. The CASS needs to examine the security context for these activities before allowing or denying them. This may be done in the following way.

Here Alice is considered to be the only participant in activities a_1 and a_2 ¹⁰. According to the proposed definition, the CASS sets the association for these activities as the same social group as their focus, i.e., the work social group, $sett_{a_1} = work$ and $sett_{a_2} = work$. Furthermore, the CASS recognizes several entities as observers of a_1 and a_2 after scanning the environment for wireless signals. Two of these entities are devices (the access point and the printer in Alice's office) and the rest of them are persons (her colleagues in the nearby offices, who are recognized by their mobile devices). All of these entities have two social groups in common with Alice: the work and public social groups. Since the work social group has a lower number of members and is more specific than the public group, the CASS sets this as the setting of these activities, $sett_{a_1} = work$ and $sett_{a_2} = work$.

On the other hand, activity a_3 is considered as being performed within the host and as having no participants and observers. As a result, the CASS determines its association and setting as the empty social group, $assoc_{a_3} = empty$ and $sett_{a_3} = empty$.

Activities a_1 and a_2 are accompanied by two additional activities, which are performed by system applications. As explained above, certain system applications enable other, user installed, applications to use the host's hardware in a convenient manner. These additional activities describe the operation of these system applications. Activity a_1 is accompanied by an activity of the *ScreenDriver* reading the screen channel, a_{1a} : *ScreenDriver reads scr_{W1}*, which takes places after a_1 . In contrast, activity a_2 is accompanied by an activity of the *TouchDriver* writing to the touch channel, a_{2a} : *TouchDriver writes tch_{W1}*, which takes place before a_2 . Each activity has the same security context as a_1 and a_2 , respectively.

Since all of these activities are allowed in accordance with company policy, the *Office_W* app is started. Alice communicates with the *Office_W* app through a screen, *scr_{W7}*, and a touch channel, *tch_{W7}*, in the same way as with the *Launcher_W* app. However, the *Office_W* app first needs to create these channels. It is considered here that this activity of creating a channel also takes place inside the host, in the same way as a_3 , which means that it also lacks both participants and observers. As a result, its association and setting are both set as the empty social group. The new resources are considered as being created for the same purposes as the application which created them, i.e., the *scr_{W7}* and *tch_{W7}* channels are work resources. The CASS allows these activities in accordance with company policy.

⁸The screen channel, *scr_{W1}*, is assumed to have been created previously in the device.

⁹The touch channel, *tch_{W1}*, is assumed to have been created previously in the device.

¹⁰As discussed in a later section, a particular handshake can be used so that the CASS can acknowledge that another person is a participant in activities involving sensor channels.

Alice prepares the report as a new data resource, doc_{W1} . This data is created by the $Office_W$, through the activity $Office_W \text{ creates } doc_{W1}$ ¹¹. This activity is considered as being performed within the host and therefore has no participants or observers. As a result, the CASS determines its association and setting as the empty social group.

The process of report preparation is actually a series of four activities performed consecutively. The $Office_W$ app first reads the information contained in the report, a_5 : $Office_W \text{ reads } doc_{W1}$ and then sends this to the screen, a_6 : $Office_W \text{ writes } scr_{W7}$. Alice reads this information and generates new information by tapping on the virtual keyboard on the screen. The $Office_W$ app receives this information via the touch channel, a_7 : $Office_W \text{ reads } tch_{W7}$, and consequently adds this to the report, a_8 : $Office_W \text{ writes } doc_{W1}$. After finishing the report, Alice taps on the screen to close the document doc_{W1} ; this is received by $Office_W$ app as an activity in the same way as a_7 . This triggers the actual activity of closing the document, a_9 : $Office_W \text{ closes } doc_{W7}$. Activities a_6 and a_8 are similar to a_2 and a_1 , respectively; the CASS therefore sets the association and setting of the former to be the same social group as he latter. On the other hand, activities a_5 , a_7 and a_9 are considered as being performed within the host, and their associations and settings are therefore set as the empty social group. The CASS allows all of these activities in accordance with company policy.

While working on the report, Alice does not notice that the $Office_W$ app is attempting to communicate with a remote server, by opening a network channel ch_{W7} to it, a_{10} : $Office_W \text{ opens } ch_{W7}$. The CASS examines the security context of this activity in the following way. The only participant in a_{10} is the remote server, which is identified by its MAC and IP address, acknowledged during the establishment of ch_{W7} . Since the CASS does not recognize this server, it is identified by default only as a member of the public social group. Therefore, the CASS sets the association of this activity to the public social group, $assoc_{a_{10}} = public$. Furthermore, the CASS scans the environment in order to identify the observers of the activity. Several devices are observers by virtue of being part of the channel ch_{W7} . These are the access points and routers that enable the delivery of messages between the host and the participant. Of these, it can be assumed that two or three are members of the work social group, as they are part of the company's network infrastructure. All the other devices are unknown and are thus considered only as members of the public social group. In addition, several entities near the host can also observe this activity, since the first device involved in the channel ch_{W7} is the access point in Alice's office and the host communicates with this using a wireless transmission. These entities are Alice's colleagues in nearby offices and the printer located within her office, all of which are members of the work social group. Since the public social group is the single common social group of all the participants and observers, the CASS determines this as the setting of this activity, $sett_{a_{10}} = public$. In accordance with company policy, the CASS does not allow this activity to be performed, since the $Office_W$ app may leak information from the report doc_{W1} to the remote server.

Finally, when Alice tries to open a home document, doc_{H1} , with the $Office_W$ app, she is not able to do this. The work social group does not allow its apps to read documents other than work documents, and the home social group does not allow its documents to be read by applications other than home applications.

Scene 2. Alice is taking a coffee break with her colleague Adam in the company's kitchenette. Her work social group is still her focus and applies to each of the activities that are performed during this scene.

Alice asks Adam to send her some documents over a wireless channel. In order to receive them, Alice starts the $FileManager_W$ app through the $Launcher_W$ app, in the

¹¹Channels scr_{W1} , scr_{W2} , tch_{W7} and tch_{W7} are created in a similar manner.

same way as the *Office_W* app in Scene 1. The *FileManager_W* app further creates screen and touch channels in order to communicate with Alice, in the same way as the *Office_W* app. Alice remains the only participant in these activities and thus their association is also determined to be the work social group. However, since these activities are taking place in a different environment, their observers are different. The CASS identifies as observers all of Alice's colleagues who are also in the kitchenette, as well as the surrounding devices with networking capabilities, such as the freezer and the microwave oven. Although this set of observers is different from the one in the previous scene, these activities still have both the work and public social groups in common with their participant. As a result, the setting is again determined as being the work social group. The activity of executing the *FileManager_W* app remains unaffected, as it takes place within the host. Thus, the CASS determines its association and setting as being the empty social group. All of these activities are allowed in accordance with company policy.

Alice accepts the documents sent by Adam by tapping on the screen, at which point the *FileManager_W* app opens a channel, ch_{W8} , to Adam's mobile device, a_{11} : *FileManager_W* opens ch_{W8} . Alice's colleague Adam is the single participant in this activity and she has two social groups in common with him: the work and public social groups. It can be assumed that there is a specific protocol through which Alice's and Adam's mobile devices exchange the social groups in their focuses. Since Adam's focus is the work social group, the CASS sets this group as the association of a_{11} . The observers of this activity are the same as those of the activities involving a sensor channel, described previously. As a result, the CASS also identifies the setting as the work social group. After the establishment of the ch_{W8} channel, *FileManager_W* performs several read and write activities through this, during which the actual transfer of the documents takes place. The association and setting of these activities is the same as for a_{11} . *FileManager_W* also creates a new data document to which it writes the received information. The association and the setting of these activities is the empty social group. Since the security context of these activities is appropriate according to company policy, they are all allowed.

In order to be able to connect to the company app store, Alice first needs to connect her mobile device using a company's access point. For this purpose she starts the *NetworkManager_W* app from the *Launcher_W* app, which automatically opens a network channel to the nearest access point, a_{12} : *NetworkManager_W* opens ch_{W3} . This company device is a member only of two social groups (work and public) and is the only participant in this activity. It is assumed that the focus of this company device is known in advance and is the work social group. Thus, the CASS easily identifies this social group as the association of a_{12} . In addition, it determines the setting of this activity to be the work social group social groups, determining this in a similar manner as for a_{11} .

Once connected to the company network, Alice starts *Installer_W* in order to install an appropriate application for viewing the received documents. The *Installer_W* app consequently opens a connection to the server hosting the company app store. This activity is similar to a_{12} , and the CASS therefore determines its association and setting as the work social group. This is because the company server, as a single participant, is also member of the work and the public social group and has the work social group in its focus, in the same way as the company access point. The observers of this activity consist of the observers of a_{12} , together with a few additional company routers, which are also only members of the work and public social groups.

All activities occurring through a network channel are accompanied by similar activities performed by a *NetworkDriver* system application. These activities describe the operation of the system application which enables other, user installed, applications to use the host's network hardware elements in a convenient manner.

Scene 3. Alice is in her office and her focus is her work social group. However, she now wants to call home, so she tries to use the *Launcher_H* app to do this. The CASS recognizes this as a change in Alice's focus, from the work to the home social group¹². As a result, it closes all work applications (e.g., the *Office_W* and *FileManager_W* apps), as well as the data and channel objects opened previously (e.g., screen and touch channels). Although Alice is able to start the *Launcher_H* app, this application is not able to use the screen and touch channels. This is because the setting of these activities is the work social group, and activities related to social groups other than work are forbidden in a work setting, in accordance with company policy. Alice has no other choice but to open the *Launcher_W* app, which brings her focus back to the work social group. At this point, the CASS also closes the *Launcher_H* app.

Activities related to the home social group are allowed in public by both home and public policies; Alice therefore leaves the company premises in order to call home using her mobile device. On her way out of the building, she logs onto the electronic access control system at the exit. For this purpose, she uses the *NFCManager_W* app, which establishes a channel with the access control system and exchanges some information with it. The access control system is the only participant in these activities, while the company devices and employees who are within the hall of the building at the same time are considered to be observers. Since they have two social groups in common (work and public) and their focus is the work social group, the association and setting of these activities are determined as the work social group. The CASS allows these activities and changes Alice's focus to her public social group after they are performed.

As soon as Alice is outside of the company premises, she tries to use the *Launcher_H* app to start the *Voip_H* app, which is also intended for home purposes. This changes her focus to her home social group. This focus applies to all her activities for the rest of the scene.

In order to be able to make calls with the *Voip_H* app, Alice needs to use the 3G connectivity of her device. She starts *NetworkManager_H* responsible for the home social group; this consequently establishes a connection with her mobile operator network and exchanges certain information. The participant in these activities is a base station which is part of the mobile operator's network. Since this entity is unknown to the CASS, it is identified by default only as a member of the public social group. The CASS therefore sets the association of this activity as the public social group. The observers of this activity are the surrounding entities, which are all members of the public social group. As a result, the CASS also determines the setting as the public social group. The CASS allows these activities in accordance with the relaxation in the home policy, which allows the *NetworkManager_H* app to communicate with any device. This relaxation is necessary so that network connectivity is possible anywhere.

The *Voip_H* app establishes four channels upon starting, via the screen *scr_{H5}*, the touch sensor *tch_{H5}*, the microphone *mic_{H5}*, and the ear-speakers *spk_{H5}*. These are all considered to be established within the host, and the CASS sets the association and setting of these activities as the empty social group. Furthermore, *Voip_H* establishes a network channel, *ch_{H5}*, with Alice's daughter's mobile device, which is the only participant in this activity. It is assumed here that her daughter has the home social group as her focus, which the CASS acknowledges after a certain protocol is used for exchange through *ch_{H5}*. As a result, the CASS determines the association as the home social group. The CASS determines the

¹²It can be assumed that the focus may change after the performance of an activity. Usually, such activities are related to the launching applications. When the user wants to perform activities related to a social group (e.g., home), she will try to start certain apps using the launching app installed for that social group (e.g., *Launcher_H*). The CASS identifies the start of this launching app (which takes place through the *Launcher* system app) as a change in the user's focus.

setting of this activity as the public social group, since the channel spans many devices from the mobile operator's network. In addition, the CASS determines that the confidentiality and integrity of ch_{H5} has been achieved, since the $Voip_H$ app establishes a secure channel with the participant's mobile device.

Alice's conversation is carried out as a series of four activities performed in sequence. The $Voip_H$ app first reads the information received through the network channel, a_{13} : $Voip_H$ reads ch_{H5} , and then emits this to Alice through the ear-speakers, a_{14} : $Voip_H$ writes spk_{H5} . Alice receives this information and generates new information by speaking into the microphone. The $Voip_H$ app receives this information as an activity, a_{15} : $Voip_H$ reads mic_{H5} , and transfers this to the participant via the network channel, a_{16} : $Voip_H$ writes ch_{H5} . Since Alice is the sole participant in activities a_{14} and a_{15} , their association is determined as the home social group; since her daughter is the sole participant in activities a_{13} and a_{16} , their association is also determined as the home social group. The observers of a_{14} and a_{15} are the same entities observing each of her activities involving a sensor channel, performed outside the company premises. Thus, the CASS determines their setting as the public social group. On the other hand, the observers of a_{13} and a_{16} are also composed of these entities, together with several additional devices which are part of the channel ch_{H5} . These are routers which enable the delivery of the messages between Alice and her daughter. Since the CASS does not know these entities in advance, it automatically assigns them as being members of the public social group. After it computes the common social group between the home member, as a participant, and these devices, as observers, it determines the setting of these activities as the public social group.

Scene 4. Alice is on her way home and her focus is the public social group. The CASS can determine this as soon as Alice leaves the company building, in the same way as explained in Scene 3. There are also alternative ways for CASS to determine this, for example using her location as obtained from the GPS sensor or many unfamiliar devices in her vicinity, and sensed by their wireless signals. This focus applies to most of the activities performed in this scene.

Alice is allowed to operate with the $Assistant_P$ app via the screen and touch channels since this is a public app and the association and setting of all these activities is determined as being the public social group. Furthermore, the $Assistant_P$ app is allowed to communicate with a particular remote server, from which it gets information about the restaurant's menu, since the association and setting of these activities is also the public social group. However, the app is not allowed to open a work document, since ACME1 company policy prohibits this.

Alice can open the door to her home only with the $NFCManager_H$ app, which is intended for home purposes. She starts this app manually, using the $Launcher_H$ app, which changes her focus to the home social group¹³.

While at home, Alice is able to open the $Player_H$ app and listen to music, since this app is intended for home purposes. However, she cannot use the $Office_W$ app for work purposes, since the ACME1 company prohibits this. In similar way to that in Scene 2, Alice can start the $Launcher_W$ app, which will change her focus to work. However, this app cannot operate with screen and touch channels, so Alice is forced to open the $Launcher_H$ app again.

4.4.2 Comparison of the Proposed Model with Existing Approaches

The proposed model is compared with various existing approaches to context-aware and adaptable security. In particular, the security context that the existing approaches take into

¹³As shown in Appendix C, her focus can also change automatically, after an unsuccessful attempt by the public $NFCManager_P$ app to unlock the door.

consideration is mapped to the proposed model. Their characteristics and goals are very similar, and the differences lie mainly in emphasis;—the former are primarily concerned with a diversity of context and ways in which to utilize this, whereas the latter mainly deal with the adaptation of their behavior. Their adaptable behavior is also specified here, which in fact constitutes the security service that these approaches provide. The results are presented in Table 4.3.

This comparison highlights several deficiencies in the existing approaches, as described below:

- The existing approaches show a serious lack of context specification. Due to the variety of possible contextual information, it is difficult to understand which aspects these systems are aware of and are able to adapt to. As a result, their applicability becomes unclear.
- The existing approaches mainly adapt to low-level contextual information, acquired directly from sensors. This makes the definition of policies cumbersome and error-prone, due to the granularity, uncertainty and frequent changes of this information. It also creates an additional burden for policy evaluation and can compromise its performance, as it leads to performing context reasoning during this process.
- Very few approaches [23, 25, 31, 76, 80, 81] take contextual information from the social domain into consideration. These mainly use the concept of social relationships. However, this is done only arbitrarily and for specific use cases, without any attempt to define this concept in a consistent manner. These approaches use only social contextual information to support and describe activities, despite the existence of other possibilities such as the classification of assets or grouping of security policies, as discussed further in the next chapter.
- The existing approaches are principally concerned only with the security properties achieved. They evaluate security properties on the basis of the characteristics of the security mechanisms which provide these properties, such as their strength or performance. An exception to this rule is [33], which tries to evaluate the specific security properties achieved by a set of security activities.
- The existing approaches generally try to implicitly balance the trade-offs between security and other properties. Except for [35, 82, 83], none of the approaches track any other properties accurately.

4.4.3 Security Requirements of the Security Context

The correct operation of any context-aware system depends on how accurately the contextual information represents reality [88, 89]. Since the contextual information directly influences the integrity of context-aware systems, manipulation of this is an attractive attack vector for these systems. This applies especially to context-aware security systems, whose application is critical and can have serious consequences for its users. Any incorrect operation of these systems can compromise them and cause an (easier) exploitation of the resources of interest they are protecting.

Contextual information needs to be appropriately secured and verified prior to its use in context-aware security systems, in the same manner as data in general. In particular, three properties of the integrity of data need to be assured [49]: (i) data integrity, which is the property that the values contained in the data have not been changed in an unauthorized manner; (ii) source integrity, which is the property that the data is trustworthy, based

Table 4.3: Security context in context-aware security systems. A checkmark (✓) denotes that the system takes into consideration this type of contextual information. Please note that a mark in the Device column denotes that the system takes into consideration contextual information about the device as a whole. (Assoc - association, SP - security property, OP - other property, AC - access control, A - authentication, SC - security channel, IM - identity management, Sev.Serv. - several services.)

Ref.	Name	Context											Adaptable Behaviour
		Entity		Activity			Social		Property				
		Person	Device	Resource	Simple	Complex	Focus	Assoc	Setting	SP	OP		
[21]	Env. roles	✓	✓	✓	✓							AC	
[22]	Cerberus	✓	✓		✓	✓				✓		A, AC	
[26]	Shrink-wrap. sec.			✓	✓					✓		AC	
[27]	Proteus	✓		✓	✓	✓						AC	
[31]	Mob. Soc. ecosys.	✓		✓	✓			✓				AC	
[35]	Progressive auth.	✓	✓	✓	✓					✓	✓	A	
[76]	Smart Space Arc.	✓	✓	✓	✓		✓	✓				Sev.Serv.	
[33]	Serenity	✓	✓	✓	✓	✓				✓		Sev.Serv.	
[29]	Adapt. PKI			✓	✓					✓		SC	
[30]	CASec for NGN	✓	✓	✓	✓							SC	
[82]	Adapt. protocols		✓	✓	✓					✓	✓	SC	
[23]	CA-RBAC		✓	✓	✓			✓				AC	
[32]	ConUCON	✓	✓	✓	✓							AC	
[80]	Daidalos	✓	✓		✓	✓		✓				IM	
[81]	Persist	✓	✓		✓	✓		✓		✓		IM	
[24]	CRAAC		✓	✓	✓							AC	
[25]	OrBAC	✓	✓	✓	✓	✓	✓					AC	
[84]	Auth. Confidence	✓			✓					✓		A	
[85]	CS Adapt. Auth.		✓	✓	✓					✓		A	
[83]	Auto. Sec. FW	✓	✓	✓	✓	✓				✓	✓	A, AC, SC	
[34]	CoDiS	✓	✓	✓	✓							Sev.Serv.	
[28]	Data UCON	✓		✓	✓	✓						AC	
[86]	RelBAC	✓	✓	✓	✓					✓		AC	
[87]	Chameleon	✓	✓	✓	✓					✓		AC	

on its source; and (iii) correctness integrity, which is the property that the underlying information represented by the data is accurate and consistent. For this purpose, standard security mechanisms for data security can be used [50]. In addition, confidence in the correctness of contextual information can be built on the basis of its quality, which is usually represented using the associated quality parameters [88, 89].

Chapter 5

Security Policy Model and Language for Ubiquitous Social Systems

In order to provide context-aware security services, context-aware security systems must employ security policies which include contextual information. These security policies are referred to here as contextual security policies. Existing policy models and languages do not provide abstractions which are sufficient and appropriate for the specification and analysis of such policies. In view of this, a policy model and language for contextual security policies is proposed in this chapter. However, a survey of some of the most representative security policy models and languages is first given, in order to identify their components as a background to the current work of designing a security policy model and language. At the same time, this survey highlights the gaps in these approaches, due to which these solutions are limited for ubiquitous social systems.

5.1 Related Work

In this section, some of the most representative security policy models and languages are surveyed. The survey includes security policy models and languages defined for security policies in general, as well as for contextual security policies. To make things as clear as possible, the former are simply referred to as general, while the latter are referred to as contextual security policy models and languages. The former are surveyed in Section 5.1.1, whereas the latter are surveyed in Section 5.1.2. In addition, Section 5.1.3 summarizes the findings from the survey.

5.1.1 General Security Policy Models and Languages

General purpose security policy models and languages are mostly related with access control. They usually act upon a single input event—a request for access, and produce two security decisions as output—access is either allowed or denied. They mainly use authorization policy rules, which are statements that grant approval to subjects to access objects, and thus to perform a particular operation over them. Some policy models and languages also use a negative form of these rules, which deny approval to subjects to access objects. These rules are simply referred to as positive and negative authorization policy rules.

Furthermore, general security policy models and languages mainly use the concept of right as a specific abstraction. Right is an approval which is granted to a subject by an authorization rule. A right states that the subject may access an object in a particular mode, which corresponds to a particular operation that can be performed over the object. Actually, a right denotes that the subject may perform the particular operation over the

object. Security policy models and languages usually deal with rights for the following three types of operations: (i) direct operations over objects, such as read, write or execute an object, (ii) administrative operations over objects, such as create or destroy an object, (iii) administrative operations over policy rules, such as delegate or revoke rights from a subject.

In addition, general security policy models and languages try to provide two essential features for the specified security policy: completeness and consistency. Completeness means that the security policy produces a security decision for each request for an activity as an input. This requires authorization rules to be specified for each possible access that a subject can perform over an object. However, this is a tedious task. Policy makers can easily omit a rule for certain accesses by accident. A more practical approach is to have a special, so called, default rule in the security policy. This rule will apply in situations when the security policy will not produce a security decision on the basis of the authorization rules, explicitly specified by the policy makers. Furthermore, consistency means that the security policy produces a single security decision for each request for an activity as an input. Although this seems easy in case of only one type of authorization policy rules, it gets difficult when both of the types are supported. For example, policy makers can accidentally allow and deny a certain operation of a subject over an object, both at the same time. This results in security policy producing useless security decision for requests regarding this activity. In order to deal with such situation, a special, so called, conflict resolution rule is needed. This rule will specify what security decision should be made when an activity is both allowed and denied on the basis of the authorized rules explicitly specified by the policy makers.

5.1.1.1 General Security Policy Models

Early security policy models have been oriented towards confining the users in a multi-user environment, so that they can separate them between each other. In order to facilitate policy specification and reuse, security policy models employ higher abstractions instead of directly specifying access rights for particular objects to particular subjects. These abstractions group subjects and/or objects according to their characteristics.

Security policy models usually represent authorization policy rules simply as relations between policy abstractions. These relations are stored in the system (e.g., in a database) and are later searched during evaluation.

Access control matrix (ACM) model. This policy model was initially proposed by Lampson [90] and was later refined by Graham and Denning [41]. It is one of the first security policy models that have been defined. Its name stems from the fact that the policy rules in the model are conceptually stored in form of a matrix, which is referred to as access control matrix.

The ACM model considers only subject, object and right as policy abstractions. A subject is a combination of a process, which is defined as a program in execution, and a domain, which represents a protection environment in which a process operates (e.g., supervisor or user). An object is a data (e.g., a file), a hardware (e.g., a disk device, a memory segment or a memory page), or a software resource (e.g., an instruction or a process). Since subjects need to be also protected, they are considered as objects, too. The model supports all three types of rights, and identifies several typical rights for each type: (i) rights for direct access of objects: read, write and execute for files, block and wakeup for processes, seek for disk drive devices, etc. (ii) rights for administration of objects: create, destroy, or owner of an object, and (iii) rights for administration of policy rules: grant, delete or transfer right to a subject.

Explicit policy rules in the ACM model are represented as relations between subjects,

objects and rights. Each subject is assigned a right for a particular operation over a particular object. In other words, rights are assigned directly to subjects. Conceptually, policy rules are stored in a matrix, in which subjects are arranged in one dimension and objects in another. Each entry in the matrix contains the rights a particular subject has over a particular object. However, since such matrix can easily grow large in size, a more practical approach is to store the rules in lists. Two types of such lists are possible: (i) access control lists, which are attached to each object and contain the rights each subject has for that object, and (ii) capabilities, which are attached to each subject and contain the rights they have for each object. Furthermore, the ACM model builds upon one implicit policy rule, which states that a subject can perform a particular operation over an object, only if it has an appropriate right for that operation over that object. A security decision is made each time a subject requests to access an object, by simply looking through the matrix (or the lists). If the subject has a right for the requested operation over the object, then it is allowed to access the object in that mode. Otherwise, it is denied. For example, if a subject S has a right to perform *read* operation over an object O , then it is allowed to read it. Also, if S has *owner* right for O or *control* right for another subject S' , then it can delete a right S' has over O .

Bell-LaPadula (BLP) model. In [42], Bell and LaPadula defined one of the first formal models in the realm of security. The model was intended for the military environments.

The BLP model also defines subject, object and right as abstractions. It makes no restrictions to what can be considered as subject and object, and leaves the precise notion of these abstractions to be determined for the particular system where the model is applied. It assumes that objects are organized in hierarchical structures over the parent-child relation. It operates only with rights for direct access to object, in particular rights for read and write operation. Furthermore, the model introduces an additional abstraction for classification of subjects and objects, called security level. It defines security level as a combination of two types of labels: security classification and security category. Security classification represents a confidentiality level of an information, that is, it states how sensitive an information is and how important is to keep that information confidential. When attached to an object, it describes that the object contains information with the particular confidentiality level, whereas when attached to a subject, it describes that the subject is cleared to access such confidential information¹. On the other hand, a security category represents a kind of information, for example, related with a certain department or mission in a military environment. When attached to an object, it describes that the object contains information with of particular kind, whereas when attached to a subject, it describes that the subject need to access this kind of information in order to perform its functionality. While subjects and objects can have multiple categories in their security level, they can have only a single classification.

Explicit policy rules in the BLP model are represented as relations between policy abstractions. The model defines two relations: labeling and ordering. The labeling relation is defined over subjects or objects, on the one hand, and security levels, on the other. With this relation, a security classification and category are attached to each subject and object. Furthermore, the ordering relation is defined over security classifications or over security categories. The model supports security classifications arranged in a linear order, meaning that the higher a classification is, the more sensitive is the information it labels, and, thus, the greater is the need for keeping that information confidential. An example of upward ordered security classifications, which comes from military environments, includes: *unclassified*, *confidential*, *secret* and *top secret*. On the other hand, the

¹A security classification attached to a subject is usually referred to as security clearance.

model supports security categories arranged in a lattice structure, under the operation *subsetof*. For example, the security category (*crypto*) is subset of the security category (*crypto, foreign*). Furthermore, the BLP model establishes three implicit policy rules. They use an auxiliary relation, called *dominate*, which is defined over security levels and combines the both ordering relations over security classifications and categories. For example, $(class_1, catSet_1) \text{ dominates } (class_2, catSet_2)$, if and only if $class_1 \geq class_2$ and $catSet_1 \supseteq catSet_2$. The three implicit policy rules are the following.

- The first principle, referred to as *simple security property*, states that a subject can read an object only if its security level dominates the security level of the object.
- The second principle, referred to as **-property*, states that a subject can write an object only if the object's security level dominates its security level.
- The third principle, referred to as *ds-property*, states that a subject can access an object in a particular mode only if it has the particular right for this mode according to the ACM model, as long as this is allowed according to first two principles. For this purpose, the model supports an additional assignment relation over subjects, objects and rights.

While the simple and *-property are mandatory and apply always, the ds-property is optional and applies only if an access control matrix is specified. Finally, a security decision in the BLP model is made when a subject requests to perform an operation over an object. If all the three rules hold, then the subject is allowed to perform the requested operation. Otherwise, it is denied.

Role-based access control (RBAC) model. The Role-based access control (RBAC) model is proposed as a standard in [43].

The RBAC model considers as subject the user, which is defined as a human being, whereas it considers as object an information container, such as a file, a directory, a database table or a database row, or an exhaustible system resource, such as a CPU cycle, a disk space, or a printer. The precise meaning of object, however, depends on the particular system where the model is implemented. At the same time, this determines the rights which are considered by the model. For example, in a file system, the RBAC model can control rights for read, write, and execute operations, whereas in a database management system, rights for insert, delete, append, and update operations. Furthermore, the model introduces the concepts of role and session as abstractions. A role represents a job function within an organization, which has certain authority and responsibility. For example, in a software development company possible roles are tester, developer, team leader, manager, etc. Roles are assigned to subjects and one subject can have several roles, which correspond to all its responsibilities in the organization. However, not all of them are active all the time. A session represents the roles which are active for a subject at a given moment. A subject can have one or more sessions, whereas a session can have only a single subject.

Furthermore, the RBAC model represents explicit policy rules as relations between the model abstractions. It defines two such relations: role-right and subject-role assignment relations. The former is defined over roles, rights and objects, whereas the latter is defined over subjects and roles. With these two relations, rights for objects are first assigned to roles, and then, by assigning subjects to roles, they are assigned to subjects. In this way, subjects are authorized indirectly, through the concept of roles. Such mediation adds flexibility and granularity to the authorization process in the model. Moreover, it fits nicely with the principle of least privileges, as it enables each employee (subject) in an organization to be authorized to access objects according to the authority and responsibility of its job function (role).

Finally, the RBAC model establishes the following implicit policy rules. A subject creates sessions and assigns them roles. A role which is assigned to a subject's active session is called an active role, and the rights assigned to it are called active rights. In this way, subjects are not statically authorized, but their active roles, and thus active rights, are determined from their active sessions. This brings dynamism to the authorization process, enabling roles to be activated and deactivated during runtime. A subject can perform a particular operation over an object, only if it has an active right for that operation over that object. A security decision is made each time a subject requests to access an object, by searching through the set of active rights for the subject. If the subject has a right for the requested access mode over the object, then it is allowed to access the object in that mode. Otherwise, it is denied.

Lattice model. Lattice model takes into account the possible flows of information which can occur as a result of a collection of given access rights [71].

It model considers as subject a process, whereas it considers as object any information container. Objects can be of different level of granularity, for example files, memory segments, or program variables, which are all data resources, or the user, which can also be considered as an object. The model further introduces security class as abstraction, which is intended to encompass the concepts of security classifications and categories, similar to the BLP model. Security classes are bounded to subject and objects, either statically, in which case they remain constant, or dynamically, in which case they vary with the subject's/object's contents. The security class of an object is determined by the information it stores, whereas the security class of a subject is determined by the security clearance of the user which operates it or by the history of security classes to which the subject has had access. Depending on the objects granularity, the model deals with different operations. For example, in case of coarse grained objects, such as files, the model deals with read and write as operations, whereas in case of fine grained objects, such as program variables, the model deals with various programming statements (e.g., assignment or conditional statement) as operations. The model translates all operations into flows of information.

The Lattice model represents three types of explicit policy rules, all of which are represented as relations between model abstractions. The first type of explicit rules is a relation between a subject/object and a security class, and assigns security classes to subjects and objects. The second type of explicit rules is a relation between a pair of security classes and specifies flows between which security classes are allowed. For example, a rule $A \rightarrow B$ specifies that flow from security class A to security class B is allowed. The third type of explicit rules is also a relation between a pair of security classes and it specifies the updated security class of a subject (an object), which occurs as a result a combination of its security class with the security class of an object (a subject), during an information flow from the object (the subject) to it. For example, if a subject of security class A reads an object of security class B , which generates an information flow from the object to the subject, then these rules specify the resulting security class of the object, after the read operation. According to these three rules, a lattice structure can be defined over the security classes, according to the function used for combining the classes in the third rule.

Finally, the Lattice model establishes a single implicit policy rule, which states that a sequence of operations are allowed only if they give rise to an information flow which is allowed according to the second explicit policy rule. If subjects and objects have a static security class, then flows are simply allowed or denied. If a subject or an object has a dynamic security class, then it is appropriately updated after the information flow, according to the third explicit rule.

Domain type enforcement (DTE) model. Domain type enforcement model was first introduced by Boebert and Kain in [91] and was later refined by Badger et al. in [72,

73]. The latter version is reviewed here, since it is the basis for creation of the Domain type enforcement language (DTEL), reviewed in the next section.

The DTE model considers as subject a software program (application), whereas it considers as an object a data resource. It defines domain and type as specific abstractions, in addition to the widely used notion of right. Both of these abstractions are attributes useful for the access control service. Domains are associated with subjects, whereas types are associated with objects. In such setting, the model is concerned not only with controlling accesses over objects, but also with controlling accesses over subject. Although this essentially makes subjects to be considered also as objects, this is not the case, as they are both associated with different attributes. The model deals with rights for direct operations over objects (e.g., read, write or execute), as well as for administrative operations over both subjects and objects (e.g., signal or create).

The DTE model represents explicit policy rules as follows: (i) rules for subject-to-object operations are represented as relations between domains, types and rights, whereas (ii) rules for subject-to-subject operations are represented as relations between domains, domains, and rights. This means that rights are not assigned directly to subjects and for particular objects, but this is done through the newly introduced abstractions, which act as mediators. Conceptually, policy rules are stored in two tables (matrices), one for each of the possible rules. In the first table—the one storing the (i) rules—domains are arranged in one dimension and types are arranged in the other. In the second table—the one storing the (ii) rules—domains are arranged on both of the dimensions. Each entry in both of the tables, contains the rights a particular domain has over a particular type, i.e., domain. Furthermore, the DTE model builds upon two implicit policy rules. The first one states that a subject can perform a particular operation over an object, only if its domain has an appropriate right for that operation over that object's type. The second one states that a subject can perform a particular operation over another subject, only if its domain has an appropriate right for that operation over that other subject's domain. A security decision is made each time a subject requests to access an object, by simply looking through both of the tables. If the subject's domain has a right for the requested operation over the particular type, i.e., domain, then it is allowed to access the object, i.e., subject. Otherwise, it is denied.

Usage control (UCON) model. In [46], Park and Sandhu propose a model which enhances the notion of access control. The model does not only strive to control whether subjects can access objects, but also how subjects use objects while they have right to access them. Hence, the name of the model is Usage Control (UCON).

The UCON model considers as subject the user, which is a human being, whereas it considers as an object a system resource (e.g., data or software). It adopts the notion of right, same as other policy models, and takes into account only rights for direct usage of objects, such as read, write or execute. The model introduces attributes as additional abstraction. It defines attributes as properties or capabilities of subjects and objects which are useful for making security decisions. It acknowledges two types of attributes: immutable, which do not change during subjects' activity over objects, and mutable, which can be modified as a consequence of such activities. Many of the abstractions defined by the previous models, such as, subject's security clearance, role or group membership, or object's security class or role permissions, can be considered as immutable attributes. On the other hand, attributes like subjects' credit or usage logs, or objects' price or number of accesses (e.g., the first access of an object is free, but the next ones should be charged on the subject, each time with higher price) can all be considered as mutable attributes.

The UCON model only describes what explicit policy rules it supports, without explaining how they are represented or evaluated. Authorization rules are not specified directly

over subjects and objects, but over their attributes. This adds great flexibility in policy specification and substantially shortens the number of rules which need to be specified. In addition to them, the UCON model further supports, so called, obligations and conditions. These are essentially forms of authorization rules, for granting rights to subjects. However, they not include attributes. Obligations are specified over previously performed (history) activities (e.g., whether a certain commercial has been watched), whereas conditions are specified over various environment variables (e.g., current time and location of the system, system load, security status of the system, etc)². In addition, the model supports specification of optional update procedures, which denote modification over specific subject or object attributes.

The UCON model establishes several possible groups of implicit policy rules over policy abstractions and describes them as different (sub-)models. These group of implicit rules are either applicable alone or can be combined together. Each group is made of only one type of explicit policy rules (authorizations, obligations and conditions) which are consulted either prior allowing and access or during an ongoing access. Thus, for example, there are groups of rules for pre-authorization, on-obligations, pre-conditions, etc. In addition, these group of rules also take into account the optional update procedures. Except pre-usage and on-usage, update procedures can also be post-usage, which means they should be performed after a subject has finished accessing an object.

5.1.1.2 General Security Policy Languages

Security policy languages seek for more expressive policy rules which can be easily reused in different security policies. In particular, they try to provide expressiveness by supporting some of the following three features. First, they try to support variables in policy rules. Variables are strings which represent any element from a set of policy abstractions. Variables enable specification of policy rules which apply for several subjects or objects at once, instead of specification of policy rules for each subject and object separately. This substantially shortens the number of policy rules that need to be specified by policy makers, thus facilitating policy specification³. Second, they try to support explicit specification of both positive and negative authorization policy rules. Since this potentially threatens the consistency of the security policy, it also requires conflict resolution rules. Security policy languages support explicit specification of these rules, too, which adds to the expressiveness of the language. Third, they try to support explicit specification of default rules. For example, this enables specification of default policy rule for open policy, in addition to the one for close policy. This default rule states that a subject can perform any operation over an object, unless this operation is prohibited by (negative) authorization rules.

With respect to policy abstractions, security policy languages can be either general or specific. General languages strive to support specification of security policies based on various security policy models. By default, they use fewer and more general abstractions. However, they usually provide possibility of defining additional ones, in order to complement the lack of specific abstraction. Thus, if policy makers need abstractions not supported by the language by default, they have to define them in advance. On the contrary, specific languages are defined for a particular security policy model and adopt the same policy abstractions defined by the model. Policy makers can directly use these languages for specification of policies according to the particular model.

Woo and Lam's language. Woo and Lam [44] were among the first to propose a logic language for specification of policy rules. Their approach is oriented towards access

²Environment information can be seen as a policy abstraction, however they are not defined as such.

³In order to achieve similar effect, policy models introduce additional policy abstractions.

control in distributed systems. Woo and Lam’s policy language adopts the widely accepted abstractions of subject, object and right. It considers as subject the user, whereas it considers as object a data resource, such as a file, a directory or a database tables. It deals only with rights for direct operation over objects, for example read, write and execute. Furthermore, the approach uses an additional abstraction of group, which denotes a set of subjects with same authorization privileges (e.g., developer). Although the language leaves room for using arbitrary abstractions in policy rules, in form of propositional predicates, it does not provide precise description how this should be done.

The syntax and semantics of Woo and Lam’s policy language is based on first-order logic. They provide little restrictions to first-order logic, which makes the language fairly expressive. However, this can be a drawback, as it can produce policy rules whose evaluation may not be decidable. The language represents explicit policy rules as logic formulas composed of three parts—prerequisite, assumption and consequent. The meaning of a rule is that: given the assumption, if a prerequisite holds, then the consequent follows. The language supports both positive and negative authorization rules. It further gives possibility for expressing conflict resolution and default rules. In addition, the language supports specification of exceptions from granted rights. Exceptions are expressed in form of special, so called, virtual rights, which are granted to subjects same as the other rights. Also, the authors mention that the language supports composition of policy rules, depending on administrative domain which specifies the rules. However, it stays unclear how this is achieved, as there is no specific parameter in the language predicates to represent the administrative domain.

Domain type enforcement language (DTEL). Together with the refined DTE model, Badger et al. also proposed a policy language in [72, 73]. As might be expected, DTEL is a specific policy language and thus adopts the same policy abstraction defined in the DTE model. Its syntax consists of several keywords, used for defining domains and types, and for assigning them to the subjects and objects. Subjects and objects are usually represented as directories from the system’s file hierarchy, which means that several subjects or objects are denoted at once. In addition, a special, initial domain is specified to denote the domain of the first subject creation during the system starting. On the other hand, the DTEL defines its own semantics, which is not based on logic, and develops its own interpreter for evaluation of policy rules. It represents policy rules as statements. It supports only positive policy rules, as a result of which, conflict resolution rules do not need to be specified.

Flexible Authorization Framework (FAF). Jajodia et al. proposed a security policy language in [45], suitable for access control in general. The FAF language defines the widely accepted subject, object and right as abstractions, and introduces group and role as additional ones. Group is defined as a set of users, whereas role is defined as set of rights over objects. The language considers as a subject the user, while it considers as an object a data resource (e.g., a file, a directory or a database table) or a role. It deals with rights for direct usage over objects (e.g., read, write or execute) and administration of policy rules (e.g., activate or deactivate role). In addition, the FAF language enables definition of arbitrary abstractions through the notion of relations, which can be of different arity.

The FAF’s policy language defines syntax and semantics is based on Datalog with negation. Datalog is a general logic programming language based of first-order logic [92]. It is equipped with several restrictions, which guarantee that the evaluation of the produced rules will be decidable and efficient. The FAF language represents explicit policy rules as logic formulas in the form of Horn clauses, composed of two parts—prerequisite and consequence. The meaning of a rule is that: if a prerequisite holds, then the consequent

follows. The language supports both positive and negative authorization rules. It provides specific predicates for handling conflict resolution and default rules and investigates several types of such rules. These rules can be easily specified differently for particular groups and roles. In addition, the language supports using history activities in policy rules, similar as obligations in UCON. History activities are specified through special predicates.

Ponder. Ponder is a policy language proposed by Sloman et al. in [47]. Its purpose is not only security, but policy management in distributed systems in general. The language adopts the basic abstractions of subject, object and right. As a subject it considers a user or a software resource, whereas as an object it considers any type of resource. Also, it considers subjects as objects. It deals with rights for all three types of operations: for direct usage and administration of abstractions and policy rules. Furthermore, the language introduces domain as a specific abstraction. Domain is a set of objects to which same policies apply. Domains partition the objects in a system and provide means for grouping them according to various characteristics, such as geographical boundaries, object type, responsibility, authority, etc. In addition, Ponder supports definition of additional abstractions, in a form of various constraints within the policy rules. However, these abstractions are arbitrary and are not structured by the language.

Ponder defines its own semantics which is not based on logic, as the other general policy languages. It follows the object-oriented programming paradigm and uses its own interpreter developed specifically for evaluation of its policy rules. Ponder defines policy rules as programming objects and supports both positive and negative authorization policy rules. It handles conflict resolution and default rules as additional, so called, meta policies. In order to facilitate policy specification and reusability, Ponder enables grouping of policies. Policies can be grouped according to many criteria, such as reference to: same object, same application, role of a subject in an organization or same department. In addition to authorization rules, Ponder supports, so called, obligation policy rules, which can be used for managing security services other than access control. However, they does not provide any means for modeling of these security services.

Rei. Rei is a policy language proposed by Kagal et al. in [48]. Its purpose is not only security, but policy management in Semantic Web in general. Rei uses the abstractions of subject and object, however it stays quiet about what these abstractions can represent. It supports definition of various operations, called actions in the language, even complex ones which combine several other operations. Such combinations are for example, sequence of activities, choice between two activities, repetition of an activity, as well as any combination of them. Rei does not use the concept of right and controls these operations directly into the policy rules⁴.

Rei is based on first order logic (Prolog). It also uses ontologies, which gives possibility of defining various properties of and relations between subjects, objects and actions. However, this is done in arbitrary manner, as these new concepts are not defined as policy abstraction from start. These properties and relations can be used for specification of various conditions withing the policy rules. Rei supports both positive and negative authorizations, as well as their administration through delegation and revoking. Possible conflicts of authorizations are handled with so called meta-rules, for example, according to their priority and precedence. In addition, Rei supports positive and negative obligations, similar as Ponder, which can be used for managing security services other than access control. However, it still does not provide any means for modeling these security services.

⁴Nevertheless, Rei uses 'right' as keyword, which is used for specifying an authorization rule.

5.1.2 Contextual Security Policy Models and Languages

A contextual security policy is more precisely defined as a function, $CtxPol : Ev \times Ctx \times St \rightarrow Dec$, which maps a certain event, $ev \in Ev$, over a system state, $st \in St$, occurring in a particular security context, $ctx \in Ctx$, as an input, to a certain security decision, $dec \in Dec$, as an output (Fig. 5.1). It extends the notion of security policy, given in Chapter 2, by introducing security context as an additional input element; security context is elaborated in Chapter 4. According to its definition, a security context corresponds only with a particular activity, which in this case, is represented by the input event. For example, a request for an activity as an input event represents an activity that is desired to be performed, and the security context accompanying this request denotes its security context. The notions of the other elements of the contextual security policy remain same as in Chapter 2.

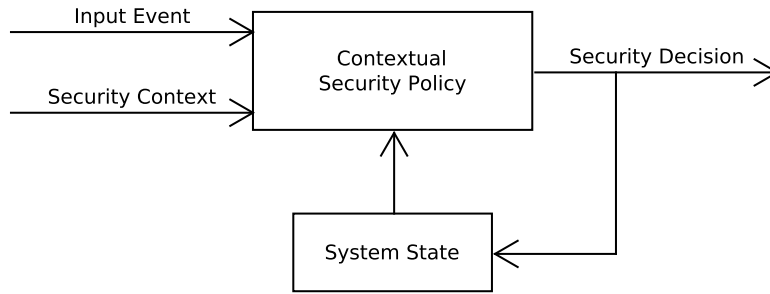


Figure 5.1: Contextual security policy.

Except for the additional input element, contextual security policies have two more specific characteristics with respect to security policies in general. First, they take into account change of security context as a specific input event. Security context of activities can change during their performance and an activity allowed in one security context might not be allowed in another one, surrounding it later. This requires security decisions regarding such an activity to be made also after it has been allowed, upon each change of its security context. If the security context of that activity is not appropriate anymore, certain adaptations need to be performed in the system. This brings us to the second specific characteristic of contextual security policies—they also produce reactions as security decisions. Reactions are activities which occurs as answer to changes of the security context of activities and ensure that the system continues to operate securely upon these changes. For example, a reaction can terminate an activity which has been previously allowed, or automatically perform additional security activities, without assistance of the user. Such security decisions can change the system state, and possibly the security context of some activities, which can trigger additional reactions in return.

Because of their specific characteristics, contextual security policies applicable in ubiquitous social systems require appropriate security policy models and languages. Such models and languages should: (i) include security context into their policy abstractions and rules, (ii) provide means for detection of changes in security context, and (iii) support policy rules for reactive behavior. In addition, since ubiquitous social systems can span over many administrative domains, these models and languages should be able to combine policy rules from each of these domains.

Many approaches take one of the security policy models presented Section 5.1.1 as

their basis and enhance it with support for context-awareness. For example, they extend these models with additional policy abstraction for context or with additional policy rules to fit context changes. On the other hand, other approaches define contextual security policy models from scratch. Nevertheless, most of the contextual security policy models also provide policy language for specification of policy rules. This is because policy rules cannot be simply represented as relations between policy abstractions.

5.1.2.1 RBAC-Based Models and Languages

Several approaches take the RBAC security policy model as their basis. These approaches adopt the policy abstractions defined by the RBAC model, namely subject, object, right, role and session. They incorporate security context in RBAC by extending the explicit policy rules with additional conditions, which contain contextual information. These contextual conditions act as constraints to the relations representing the explicit rules, meaning that the relations hold only if the conditions are satisfied. Different approaches constraint different relations.

Environment roles (ER). Covington et al. [21] proposed an extension for RBAC which captures contextual conditions as an additional abstraction, called environment role. Environment role represents a set of security contexts defined by certain condition. For example, the environment role *business hours* represents the following context condition: *time of day is between 08:00 and 17:00*. Environment roles are assigned to subjects in the same manner as subject roles. However, they can be assigned only when contextual conditions by which they are defined are true. Environment roles model security context arbitrarily, as they only represent security contexts, but tell nothing about what security context can be. Furthermore, the ER model adds environment roles to the RBAC's subject-role explicit policy rule. This results in a rule defined by a subject, right and set of environment roles, which states that the particular role is assigned to the subject only when the contextual conditions defined by the set of environment roles are true. Finally, the ER model establishes the following additional implicit rules for environment roles—an environment role is active only if the contextual condition which defines it is true. Environment roles are activated and deactivated automatically as the contextual information contained in the condition change. Therefore, a subject can perform a particular operation over an object, only if it has an active role, which has the appropriate right for that operation over that object, and the corresponding environment roles for this role are active. Otherwise, it is denied.

Context-aware RBAC (CA-RBAC). Kulkarni and Tripathi proposed a policy model for context-aware RBAC [23]. The CA-RBAC model adds the contextual conditions to the RBAC's explicit policy rules, making the relations representing these rules constrained. For example, a constrained subject-role relation denotes that a subject can have a role assigned if a contextual condition is true. Similarly, constrained role-right relation denotes that a role can have a right assigned if a contextual condition is true. Also, the model introduces the notion of abstract objects, which represents a set of objects for whom a certain contextual condition applies. Abstract objects can be used in role-right assignments, same as regular objects. For example, a role can be assigned *read* right for an abstract object representing all objects which are created on a particular date. Furthermore, the CA-RBAC model introduces several additional implicit policy rules. According to this model, a role is active for a subject only if the corresponding contextual condition in the subject-role is true. Similarly, a right is active for a role only if the corresponding contextual condition in the role-right relation is true. Roles and rights are automatically activated and deactivated as the contextual information contained in their contextual conditions change. Also, abstract objects are instantiated automatically with

particular objects. Therefore, a subject can perform a particular operation over an object only if it has an active right for that operation over the object, which depends on the active roles and instantiated objects. Otherwise, it is denied.

Event-Driven RBAC (ERBAC). Bonatti et al. propose another extension of the RBAC model called Event-Driven RBAC (ERBAC) [93]. This model extends the notion of rights and roles by using contextual conditions. It introduces the notion of parametrized right, as a right which denotes that an operation can be performed over an object only if certain contextual conditions hold. Also, it considers a role to have two states depending on contextual conditions, either enabled or disabled. The model uses explicit policy rules for specification of which contextual conditions apply for particular parametrized rights and for enabling of particular roles. Furthermore, ERBAC mainly considers location and time as security context and extensively models how they are represented and used in contextual conditions. It also gives possibility of defining other contextual information, through the general concept of event. However, this is done only in arbitrary manner. Finally, ERBAC introduces the following implicit policy rules. Parametrized rights are activated and roles are enabled (disabled) automatically as the contextual information contained in their contextual conditions change. Only enabled roles can be activated, that is, assigned in subject's active sessions. Also, only active rights can be exercised over objects. Therefore, a subject can perform a particular operation over an object only if it has an active right for that operation over the object. Otherwise, it is denied.

5.1.2.2 UCON-Based Models and Languages

Context-Aware Usage Control (ConUCON). Bai et al. [32] propose a Context-Aware Usage Control (ConUCON) model. The model adopts the policy abstractions defined in UCON, namely: subject, object, right and attributes. However, it considers application as subjects, which is different than UCON, where subjects are users. The model also considers two attributes particularly important for security: confidentiality and integrity label. These labels are attached to subjects and data objects and denote their security levels according to the BLP [42] and Biba [94] model. Furthermore, the model introduces an additional abstraction for representing security context, called simply context. Context is an environmental or system property and can be of several types, such as CPU rate, battery, location and time. The model takes location and time as especially important context types, and models them more precisely. Finally, the ConUCON model adopts the policy rules defined by the UCON model. Same as UCON, it does not explain how explicit policy rules are represented or evaluated, but only describes them. It defines some of the authorization policy rules solely over confidentiality and integrity labels, similar to the BLP and Biba models. However, these policy rules are specified as explicit, opposite from BLP and Biba, where they are implicit. As expected, they apply only for read and write operations over data objects.

5.1.2.3 Logic-Based Models and Languages

Cerberus. Al-Muhtadi et al. [22] propose a context-aware security system for smart spaces called Cerberus. Cerberus defines a logic language for specification of security policies, which is based on Prolog. The language underlies on a policy model which is not precisely defined. Only a subject and object are clearly defined as policy abstractions. It considers as a subject the user, whereas it considers as an object any kind of resource (e.g., hardware, software, data or network). The language does not consider the abstraction of right and aims at controlling whether the subject can access an object in general, that is, without specifying about the particular operation during the access (e.g., read, write,

execute, etc.). Furthermore, the language enables definition of various contextual information as policy abstractions. Contextual information can be defined in form of predicates, whose name denotes the type of contextual information and whose parameters denote a particular element of that type. For example, *Location(Bob, entering, roomA)* denotes a particular element of a contextual information *Location*. The language also supports definition of (high-level) contextual information as a composition of other (low-level) contextual information. For example, subject's confidence value, as a high-level contextual information, can be defined as a composition of two contextual information, namely the security mechanism used for authentication of the subject and the confidence level in that mechanism. Finally, Cerberus represents explicit policy rules as Prolog rules. It supports only simple authorization rules which define whether a subject can access an object in general, without specifying about the particular operation during the access. It relies on Prolog for achieving completeness of the security policy. Since it does not support negative authorization, consistency is not an issue for it.

OrBAC. Cuppens and Cuppens-Boulahia [25] propose a contextual policy model called OrBAC. The model is based on RBAC policy model and adopts its basic abstractions of subject, object and role. It uses action instead of right as an abstraction. OrBAC introduces several additional abstractions, namely: organization, view, activity and context. Organization is an organized group of subjects, within which they have assigned some roles. An organization can be seen as an entity that is responsible for managing a security policy. View and activity act as a grouping mechanisms for objects and actions, respectively. They represent objects, that is, actions, for which the same policy rules apply. In a way, they act as roles for objects and actions. Context is used for expressing different types of extra conditions that control actions of subjects over objects. OrBAC uses predicates for definition of various contexts, such as time, location, previous activities, environment, system architecture and purpose. Furthermore, the authors also propose a security policy language for OrBAC, which is based on Datalog. As expected, the language builds upon the same policy abstraction as the model. It represents the explicit policy rules as Datalog rules. It supports both positive and negative authorizations and enables definition of conflict resolution and default policy rules. In addition, it supports positive and negative obligation policy rules.

Proteus. Toninelli et al. [27] proposed a contextual policy language called Proteus. The language is not based on any traditional security policy model. It centers around the notion of subject, object, operation and context. As subject, it considers the user, whereas as object it considers any kind of resource. It supports all three types of operations, although it does not state this explicitly. Contexts act as intermediaries between the entities requesting access to resources and the set of operations that can be performed on these resources. A context is associated with a single operation which is allowed to be performed over a particular object. An object, on the other hand, can be associated with more than one context. Furthermore, Proteus is based on description logic and logic programming. It supports positive and negative authorizations, as well as conflict resolution and default rules. In Proteus, several contexts can be active for an object in a given moment, which consequently means that a subject can perform several operation over it. When a subject tries to perform an operation, it is checked whether this is allowed according to the active protection contexts. As contextual information change, so does the active protection contexts and the set of allowed operations over objects.

CoDiS. Mostéfaoui [34] proposed a contextual security policy model called CoDiS. CoDiS model centers around subject, security actions and context as policy abstractions. As subject it considers the user, whereas as security action it considers an executable method that aims to provide some security service. As context it considers information

such as time, location, device and resources characteristics and environment. Interestingly, the model considers operations over objects also as context. However, it lacks of detailed description about what kind of operations and objects it deals with. Furthermore, the CoDiS model represents policy rules as so called contextual graphs. Contextual graph is defined as an acyclic graph, with unique input and output, and a set of nodes connected by oriented arcs, in a parallel organization. A node can denote a security action or a contextual condition, during which the current value of a particular contextual information is checked. A policy evaluation, in this case, is a path through the graph, from input to output, which depends on the current values of contextual information. The model does not support specification of default and conflict resolution policy rules.

5.1.3 Discussion

General security policy models and languages are silent on the subject of security context. This is understandable, since many of these date back to the early days of computer security, when more basic matters needed to be discussed such as which resources should be protected, from whom and how. Nevertheless, certain general policy models and languages may be able to support security context; for example, the UCON model uses attributes as policy abstractions and enables various environment conditions to be defined as policy rules. Both attributes and environmental conditions are essentially contextual information. Also, the models proposed by Ponder and Rei can be extended using domain-specific concepts which can handle security context. However, none of these approaches consider a change in the security context as a particular input element in the security policy; they also still require an appropriate analysis and specification of what constitutes security context and how it influences their underlying security service, i.e. access control.

On the other hand, existing contextual security models and languages consider security context as an add-on. These are primarily concerned with how to incorporate security context into general policy models, even though these are not intuitive for representing contextual security policies. Instead, security context must be introduced as a primary abstraction and modeled in terms of how it influences access control as a security service. These approaches are also mainly concerned solely with access control, and rarely support information flow control as part of the model. Although some approaches, such as [95], use the security context in controlling information flow, this is done arbitrarily, without relating security labels to the security context.

Furthermore, existing contextual security models and languages rarely include a reaction as a security decision. Consequently, it is unclear whether and how these approaches adapt to context changes. While it seems straightforward to assume the results of a reaction in case of data objects, this becomes difficult in the case of application objects. For data objects, it can safely be assumed that after a change in context, any future access request by a subject over an object will be denied, as a reaction to this change. However, when the context changes, applications that have been previously executed and are running as subjects need to be terminated. Existing approaches do not address this requirement or how it can be achieved. Although some policy languages enable the specification of reactive behavior in the form of obligatory actions (activities), this is done only arbitrarily. The lack of modeling of reactions therefore makes policy enforcement more difficult. The enforcement of reactions is possible only if it is clearly known in advance what these reactions are and how they depend on context changes as input.

Finally, existing contextual security models and languages mainly take low-level contextual information into account. This is impractical from the aspect of system design, since such contextual information changes frequently, resulting in the generation of an enormous number of input events to the system. This can compromise the performance

of the security system and can even lead to attacks on its availability (denial of service attacks). Some approaches (e.g. [22, 25, 27]) do support contextual information at a higher level of abstraction; however, they incorporate the context reasoning rules into the security policy. This is unnecessary from the aspect of policy specification, as it requires security administrators to be comprehensively familiar with context reasoning, and can further influence the correctness of the security policies. Moreover, this can even compromise the performance of the security system, since context reasoning will be performed during security decision making.

5.2 Development of the Security Policy Model

In order to address the issues discussed above, a new security policy model is proposed here for ubiquitous social systems. The purpose of this model is to represent a range of contextual security policies which are applicable in such environments and to enable their security analysis. The model centers around the notion of security context, as described in Section 4.2, and is based primarily on the conceptual model of security context defined in Section 4.3.

The security objective of the policies represented by this model is access control. However, a slightly different perspective towards access control is used in this model, whereby activity rather than access is considered the fundamental unit of control. Hence, this service is referred to as activity control, since the term ‘activity’ corresponds better to the concept of security context used here. According to the definitions of security context, activity is an essential part of security context, since security context rises from activity and cannot exist without it. The term ‘access’ is related more to the operation within the activity than to the activity itself; as such, it does not give an impression of an organized performance from which security context arises. In addition, it suggests only a single interaction between the assets of interest (i.e. between the subject and object), whereas several possible interactions are addressed during a single activity in the proposed model. In order to emphasize the concept of activity in access control, this security service is referred to as activity control in the proposed model. Consequently, the model is referred to as the context-aware activity control (CA-ACON) model.

Furthermore, the term ‘activity’ has connotations of duration, which is especially important for context awareness. Traditional access control models consider access as being permanent; once access is granted to an application, it exists as long as the application exercises it. However, the effects of different activities on the assets involved vary. For example, an application can open a channel once, and then send and receive packets of information many times afterwards. In the case of this activity, the application continues to access the channel for some time, and the security context for this activity needs to be constantly monitored. On the other hand, in the case of consecutive send and receive activities, the application accesses the channel only briefly and the security context of these activities needs to be checked only at the time they take place.

The proposed model is a form of originator-based control. It enables each social group in ubiquitous social systems to protect its own assets of interest. A resource that is assigned to a particular social group, or an entity that is defined as its member, is considered here as an asset of interest. Entities are also considered as assets, since these are responsible for achieving the goals of the social group. A social group can protect its assets by controlling their interactions with other assets during activities, especially if these assets are related to another social group. Several interactions are identified between assets during a single activity and each of them analyzed from the perspective of security. At the same time, the proposed model supports discretionary-based control; it empowers the user to specify

certain relaxations from social group requirements. These relaxations refer to a particular asset, and also need to be allowed by the particular social group to which these assets are related.

The proposed policy model has its roots in two of the author's previous works [96] and [97]. The former describes a decentralized access control mechanism for complex environments, such as P2P networks. It enables the specification of fine-grained security policies, based on location, time, service quality, sharing ratio etc., which essentially constitute contextual information. This work helped in a practical exploration of access control in complex and decentralized environments, and demonstrated the importance of security context to security decisions and the need to involve reactive security behaviors into the policy model. The latter describes a model for contextual security policies applicable in environments such as ubiquitous social systems. This model is based solely on security properties as the security context, and attempts to relate these to security metrics. This work identified the benefits of using high-level context abstractions in security policies and the necessity of including social context in security decisions.

The CA-ACON model was introduced by the author in [98]; this is described in detail below. Firstly, policy abstractions and policy rules are defined as its main components; following this, its operation is described. Finally, the ways in which this can be enforced are presented.

5.2.1 Description of the Model

5.2.1.1 Policy Abstractions

The following abstractions are defined in the proposed policy model: activity, subject, object, origin, the security context of the activity, continuous activities and current security contexts. Figure 5.2 illustrates these policy abstractions and the relationships between them. Since continuous activities and current security contexts form sets of activities and their security contexts, respectively, they are represented using these abstractions in the figure. Each policy abstraction is described in detail below.

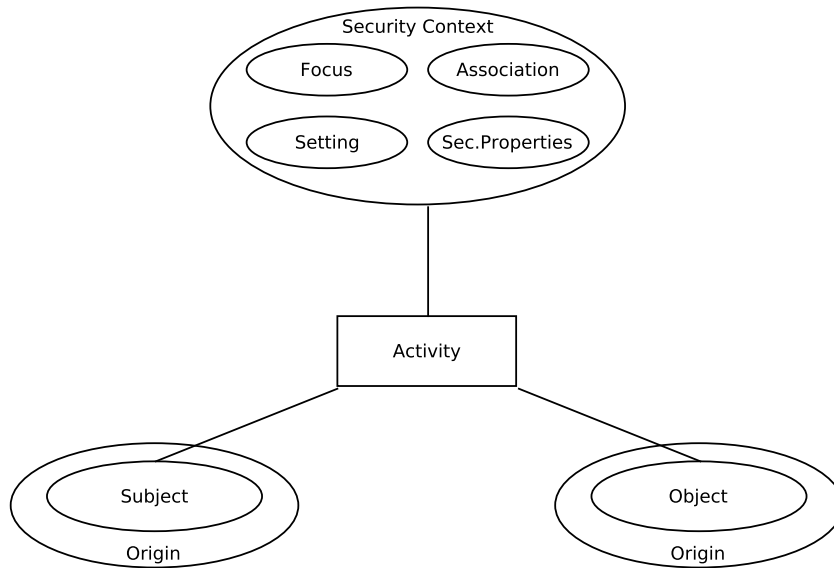


Figure 5.2: Policy abstractions used in the proposed model.

Activity, subject and object. Activity is the single unit of control in the proposed policy model, as described above. The same definition of activity is used as in Section 4.3; however, a restriction is imposed on this concept. Only simple activities are addressed here, and complex activities are treated as sequences of simpler ones, which need to be controlled separately. This simplifies the proposed policy model in terms of fewer components; however, an appropriate security analysis is required prior to the specification of the policy rules (see Section 5.4.1). As a result, the set of all simple activities is denoted as A' , which is a subset of the set of all activities A defined in Section 4.3, $A' \subset A$.

The subject and object are actors involved in an activity.

Definition 5.1. A *subject* is a resource which performs an activity by carrying out an operation, whereas an *object* is the resource on which an activity is performed.

According to the definition of activity, a subject is an application, while an object may be any resource, i.e. data, a channel or an application. It follows that applications can be both subjects and objects, and that subjects are also objects. The sets of all subjects and objects are denoted as S and O .

Resources can reside in two states: active and passive. These states differ in terms of the accessibility of the information contained within the resource and the functionality it provides. A passive resource is used to store information and functionalities in the device's memory for future use. This is analogous to a closed container of which the contents are not accessible directly. An example of a passive application in Linux is an executable file, whereas an example of passive data or channel is a regular file. In order for these contents to be accessed and utilized, the resource needs to be activated. An active resource is analogous to an open container, from which information can be read and modified, in case of data or a channel, or whose functionality can be executed, in the case of an application. An example of an active application in Linux is a process, whereas an example of active data or channel is a file descriptor.

With the help of object states, a subject and object can be defined more precisely, as follows.

Definition 5.2. A *subject* is an application in an active state, whereas an *object* is data, a channel or an application, in either a passive or an active state.

It follows that active applications can act both as subjects and objects, whereas passive applications can act only as objects. Thus, the sets of passive and active resources are denoted as O_P and O_A , and these form the set of objects, $O = O_P + O_A$. In the remainder of this thesis, the concepts of resource and object are used interchangeably.

Seven types of activities can be performed on objects, depending on the operation carried out during the activity. These are: *create*, *open*, *execute*, *read*, *write*, *close* and *destroy*. These types are sufficient to describe and control all the activities performed in ubiquitous social systems. Each of these is performed on a particular state of the object and has a particular effect on it which results in state transitions. Table 5.1 describes the meaning of each of these types of activity. These possible types of activity, together with the possible object states and state transitions, compose the lifecycle of an object. Fig. 5.3 illustrates this lifecycle as a state diagram.

The object lifecycle can be described as follows. Each object o begins its lifecycle when a certain subject s performs a 'create' activity, $(s, create, o)$. This adds a new element to the set of passive objects, $O_P = O_P + \{o\}$. In order to be directly used, the object o must be activated. Two activities are used for activating an object: an open activity, which applies to data and channel objects, and an execute activity, which applies to applications. This is due to the activation of the latter having different security implications than activation

Table 5.1: Description of the types of activity supported by the proposed policy model. A description is given for each resource type on which an activity is performed.

Type of activity	Description
create	Data, channel, application: A new resource is created.
open	Data, channel, application: A resource can be read or written.
execute	Application: A resource is started and its functionality is available.
read	Data: Information is read from a resource. Channel: Information is received by a resource.
write	Data: Information is written to a resource. Channel: Information is sent via a resource.
close	Data, channel: A resource can no longer be read or written. Application: A resource's functionality is no longer available.
destroy	Data, channel, application: A resource is destroyed and no longer exists.

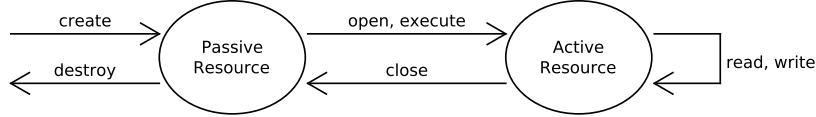


Figure 5.3: Object lifecycle.

of the former; different types of activities are required in order to control these separately. This is discussed in more detail in Section 5.4.1. Both open $(s, open, o)$, and execute $(s, execute, o)$ activities move o from the set of passive objects to the set of active objects, $O_P = O_P - \{o\}$, $O_A = O_A + \{o\}$. While in an active state, a data or channel object can be read or written to. During a read activity, $(s, read, o)$, information from o is transferred into the subject s ; during a write activity, $(s, write, o)$, information from s is transferred into the object o , thereby modifying it. Read and write activities can be performed numerous times on an object without changing its state. On the other hand, an application object in an active state becomes capable of providing its functionality; that is, it becomes a subject. This object o can therefore be added to the set of subjects, $S = S + \{o\}$. Active application objects (subjects) can perform activities on other objects. For example, o can create a new object $(o, create, o_1)$, and open it, $(o, open, o_1)$, in order to write to it, $(o, write, o_1)$. An object o stays in an active state as long until it is deactivated with a close activity, $(s, close, o)$. This activity moves o back to the set of passive objects, $O_A = O_A - \{o\}$, $O_P = O_P + \{o\}$. It also removes o from the set of subjects, $S = S + \{o\}$, in the case that o is an application. Transitions between the passive and active states of an object can be performed many times during its lifecycle. Finally, a destroy activity, $(s, destroy, o)$, destroys the object and removes it from the set of passive objects, $O_P = O_P - \{o\}$.

Security context of activity. Each activity is surrounded by a particular security context, which can change with the performance of other activities. Security context is discussed extensively above in Chapter 4, as a tuple, $ctx_a = (a, focus, assoc_a, sett_a, sprop_a)$, consisting of an activity a , the focus of the user, $focus$, the social concepts that characterize a : association, $assoc_a$, and setting, $sett_a$, and the values of the security properties of the resources that are involved in a , $sprop_a$. A further function is defined as the *current security context*, $cctx : A \rightarrow Ctx$, which maps an activity to its security context at a given

moment. An important observation should be made here about the materialization of security context; by definition, security context surrounds an activity during its performance, that is, while the subject accesses the object by carrying out the operation. However, for the purpose of activity control, the security context of an activity needs to be examined prior to it taking place, so that the activity can be allowed or denied accordingly. For example, this occurs when an activity request is received as an input event. In this case, the security context of the requested activity is in fact the expected security context, which will surround the activity if it is performed immediately. This security context has not yet been materialized, since the activity has not yet been performed; it will be experienced as soon as the activity is allowed and started.

Origin of subject and object. Subjects and objects are more effectively described and classified in terms of their origin.

Definition 5.3. An *origin* is a single social group to which a resource can be related and which captures the purpose of creation of that resource.

For an application, the origin represents the social group whose goals it is intended to achieve; for data or a channel, the origin represents the social group whose information it is intended to contain. Resources inherit their origin when created, from the subject which created them. For each social group, it is assumed that there is an application from which each resource related to that social group inherits its origin. This means that each resource of a social group is created either directly by this application, or indirectly by another application created by it. Origin is defined here as a function, $org : Res \rightarrow SG$, which maps a resource to a social group. The origins of all subjects and objects compose the set of all origins, Org , which is defined as a set of tuples $(o, org(o))$, where $o \in O$. Since each subject is an object, this definition implies that Org also includes the origins of all subjects.

Continuous activities and their security context. Each activity is performed when the subject carries out the operation of the activity on the object. It is assumed that once an activity is performed, the effects of that activity on the assets involved are manifested.

Definition 5.4. A *continuous activity* is an activity whose effect on the asset involved endures after it is performed.

Only *open* and *execute* activities are considered to be continuous.⁵ This is because these activities move the object into an active state, where its functionality becomes available to the user in the case of applications, or where its information becomes available to the subject in the case of data and channels. For example, an open activity, $(s, open, o)$, makes a data object o available to the subject s for reading and writing during a number of activities in the future. This differs from a single read activity, $(s, read, o)$, during which information is exchanged between the subject and the object only as it is being performed, and not subsequently.

The security context of a continuous activity is assumed to be changeable after it is performed. As a result, the security context of these activities needs to be controlled after the activity is allowed, for each context change. In order to be able to detect context changes, all continuous activities which are currently being performed are examined, and their security context constantly monitored. The *current continuous activities* $A_C \subset A$ are

⁵The *destroy* activity is not considered to be continuous, since the involved object is destroyed after its execution. It cannot be said that the effect of this activity on the involved object exists since the object no longer exists.

defined as the set of all continuous activities whose effect on an object endures. Furthermore, the *current security contexts* $CCtx$ are defined as a set of tuples $(a, cctx(a))$ of the current security context of each continuous activity $a \in A_C$. The elements in the current security contexts $CCtx$ are constantly updated in order to ensure that values are relevant.

5.2.1.2 Policy Rules

Policy rules guide the process of taking security decisions regarding the activities performed in ubiquitous social systems. The proposed model builds upon the following explicit and implicit policy rules.

Explicit policy rules. Explicit policy rules in the proposed model are defined in terms of authorization. For activity control, this means that the rules specify whether a particular activity is allowed to be performed, with respect to its security context and the system state. As described earlier, the proposed policy model is a form of originator-based control which includes support for discretionary control. This requires the model to combine policy rules specified from several social groups, as well as from the user. In order to facilitate this process, interaction can be defined as a finer-grained abstraction for control than activity. An interaction simply denotes an ability or a means of communication between two assets involved an activity, which may have security implications. Thus, rather than specifying explicit policy rules through activities, the proposed model uses these rules to specify whether a particular interaction between two assets during an activity is allowed.

During a single activity, parallel interactions occur between several of the assets involved in that activity. Assets may be involved in an activity in the following roles: a resource can act as subject or object, whereas an entity can act as a user, participant or observer of the activity.⁶ However, not all possible interactions are relevant for security considerations; only those which involve the subject of activity, that is, subject-object, subject-user, subject-participant and subject-observer interactions are assumed to be relevant. The reason for this is twofold. Firstly, the subject performs activities and hence drives each of the interactions. Any interaction involving the object either (i) goes through the subject, for example in the case of data, since the user/participants/observers cannot access data directly; or (ii) is negligible, for example in the case of channel and application, since channels only serve as pathways for communication between subjects and users/participants/observers, and the purpose of applications is to be used as subjects.

Secondly, the subject is usually the only target for which security policies can be enforced. Thus, any discussion of direct interactions between the user, the participants and the observers is futile, as these cannot be enforced. All interactions can be easily translated into interactions between social groups by mapping the assets involved to the social groups to which they are related. This is achieved as follows (Fig. 5.4): (i) the subject and object are related to the social group in their origin; (ii) the user is related to the social group in his/her focus, i.e. the focus of the activity; (iii) the participants are related to the social group in the association; and (iv) the observers are related to the social group in the setting of the activity. The set of all interactions is denoted as *Int*.

According to their application, explicit policy rules can be of three types: group, asset and global.

Group policy rules apply to all assets related to a particular social group. They express the security requirements of a particular social group regarding a specific interaction during an activity. This type of policy rule can be specified for each possible type of interaction during an activity. A set of group policy rules, specified by a single social group, is referred

⁶The entity representing the user also acts as participant in the activity; that is, the user is one of the participants in an activity. When this entity acts as the user, it initiates the performance of the activity.

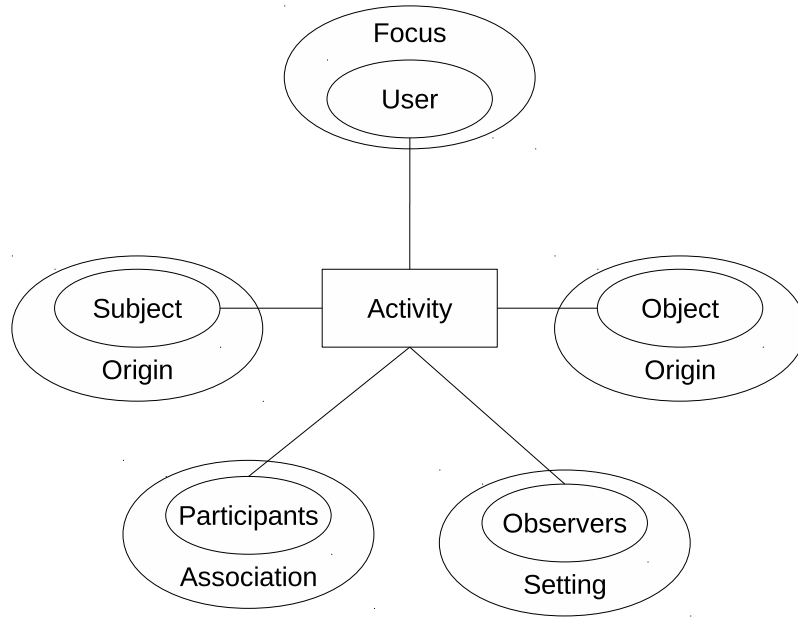


Figure 5.4: Assets involved in an activity and their related social groups.

to here as a group policy. A group policy is taken into account during each decision regarding an activity which involves an asset related to the social group that specifies the policy. Group policies are specified by social group administrators.

Asset policy rules apply to a particular asset of a social group. They are intended to express relaxations from group policies regarding a particular interaction during activities. Asset policies are specified explicitly by the user. They are taken into account only during decisions regarding an activity involving the particular asset for which they are specified. However, in order for this to happen, the relaxations they express must first be allowed by the group policy, which is specified by the social group from which the asset originates. Although asset policies can be specified for each possible type of interaction during an activity, only asset policies for subject-user and subject-participant interactions are considered here as being practical.

Global policy rules apply to assets related to any social group. They are intended to express default security requirements regarding each interaction during any activity. Global policies are taken into account during decisions on each activity; they combine group policies specified by the social groups whose assets are involved in the activity, in order to evaluate them in a single decision. In other policy models, these policies are usually specified by system administrators. However, in case of ubiquitous social systems there is no single administrator, since they act as a combination of various administrative domains. Global policies are therefore predefined in the proposed policy model.

Implicit policy rules. Three implicit policy rules are established in the proposed model.

Basic policy rule: An activity is allowed to be performed only if all interactions involved in this activity are allowed by the social groups whose assets are involved in the activity. This means that an activity is allowed if this follows from each group policy specified by the social groups concerned, as well as from the applicable asset policies, if specified.

Contextual policy rule: If the security context of a continuous activity changes while it is being performed, then the basic policy rule for this activity must be re-evaluated. If the activity is allowed according to the basic policy rule, then it continues to be performed

as before; otherwise, the effect of this activity needs to be terminated, according to the termination policy rule.

Termination policy rule: The effects of a continuous activity are terminated by closing the object on which this activity is being performed. If there are other continuous activities involving the same object, these also need to be terminated. These ‘close’ activities are performed by the context-aware security system as a subject.

5.2.2 Model-Based Operation

The operation of a context-aware activity control (CA-ACON) service is described here, which implements security policies based on the proposed model. As stated earlier, CA-ACON controls whether or not a particular activity is allowed to be performed with respect to its security context. Moreover, it monitors activities after they are allowed, and adapts to changes in their security context if necessary.

Fig. 5.5 illustrates the lifecycle of an activity controlled by CA-ACON. As can be seen, it represents a state diagram whose transitions between states occur as result of the possible inputs and outputs in the proposed model. This state diagram is described in more detail below.

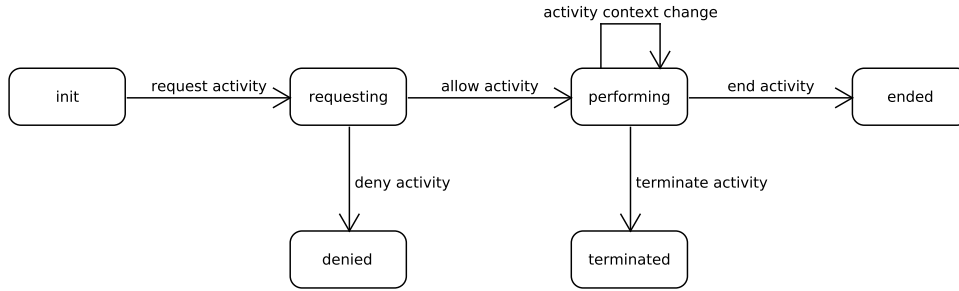


Figure 5.5: Activity lifecycle.

Each activity begins its life cycle in an initial state. For a particular activity $a = (s, op, o)$, $a \in A$, this happens when a subject $s \in S$ tries to perform an operation $op \in OP$ on an object $o \in O$. As a result of this, a request for this activity, $req(s, op, o)$, is issued to the CA-ACON. The CA-ACON accepts each request, but continues to process only those which are valid. A request is valid if the operation op corresponds to the state of the object o , according to the object lifecycle (Fig. 5.3). In the case that $req(s, op, o)$ is valid, the CA-ACON moves the activity a into ‘requesting’ state.

While a is in a requesting state, the CA-ACON makes a security decision regarding this request. During this process, the CA-ACON evaluates the explicit and implicit policy rules specified as part of the security policy. In particular, it attempts to verify whether the security decision $allow(s, op, o)$ follows from these rules. If holds, then the CA-ACON moves the activity a into a ‘performing’ state. Otherwise, it generates a security decision $deny(s, op, o)$ and moves the activity into a ‘denied’ state.

A denied state of an activity a indicates that a will not be performed. The CA-ACON simply drops the received request $req(s, op, o)$ and terminates its operation regarding a . This means that the denied state is a final state in the lifecycle of an activity.

On the other hand, the performing state of an activity a indicates that a is being performed. The CA-ACON simply lets the subject s to carry out the operation op on the object o , which results in the manifestation of the effects of a on s and o . For example,

the object o is moved from a passive to an active state; if o is an application, it is also moved into the set of subjects S . Activities stay in the performing state according to their duration. Ephemeral activities stay only briefly, after which they move to an 'ended' state; on the contrary, continuous activities stay for a longer period of time, since they have a prolonged effect on the subject and object involved. The CA-ACON constantly monitors the security context of all continuous activities, A_C , since their security context can change during their performance. It detects these changes with the help of the values saved in the set of current security contexts, $CCtx$. Upon detection of a context change for an activity, the CA-ACON makes another security decision regarding this.

Following a change in the security context of an activity, $chg(s, op, o)$, decision making is carried out as follows. The CA-ACON evaluates the explicit and implicit policy rules in the same way as before with respect to the (new) activity context. However, it interprets the evaluation outcome slightly differently. If $allow(s, op, o)$ follows from the policy rules, the CA-ACON does not produce any security decision, but simply leaves the activity a in a performing state. On the other hand, if $allow(s, op, o)$ does not follow from the policy rules, the CA-ACON produces a reaction that a needs to be terminated as a security decision. This moves the activity into a 'terminated' state.

A terminated state of an activity a indicates that a is no longer allowed to be performed and that its effects on s and o will be terminated. This decision is enforced with the execution of an opposite activity as a reaction to a . The terminated state applies only to a continuous activity; it is a final state after which the activity and the operation of the CA-ACON are finished.

Finally, the ended state of an activity a indicates that a and its effects on the subject and object involved are finished. Ephemeral activities move to this state automatically, immediately after they have been performed, while continuous activities move to this state only when an activity with the opposite effect on the subject and object involved is requested, allowed and performed. For example, for the activity $(s, open, o)$ the 'opposite' activity, according to the object lifecycle, would be $(s, close, o)$.

5.2.3 Enforcement of the Model

An enforcement layer for the proposed policy model is described here. It represents how subjects and objects, as well as possible operations on them, can be represented using operating system concepts. The enforcement layer described here is based on Linux, since this is a popular open-source operating system. However, similar enforcement layers can also be defined for other operating systems, since they use similar concepts. The selection of policy abstractions used here is a general one which is independent of a specific implementation, and which can be applied within environments other than operating systems.

Table 5.2 shows how can objects be represented according to their state. As can be seen, all data, channel and application objects in a passive state are files, and can be represented by their file names. Data and channel objects in an active state are file descriptors and can be represented by their IDs, whereas application objects in an active state are processes and can be represented by their process ID. The latter also applies to subjects, since subjects are in fact active applications.

Furthermore, Table 5.3 shows how operations on particular objects can be represented using system calls. This set of system calls can be assumed to be sufficient for monitoring all inputs and enforcing all outputs from the proposed policy model. As expected, open, read and write operations do not have system call representation for applications, as these do not apply to them. For the same reasons, the execute operation does not have a system call representation for data and channel objects.

Table 5.2: Objects and their possible identifiers, according to their state.

Resource	State	Identifier
Data	passive	file name
Channel	passive	file name
Application	passive	file name
Data	active	file descriptor ID
Channel	active	file descriptor ID
Application	active	process ID

Table 5.3: System calls for particular operation on objects.

Operation/Object	App	Data	Channel
create	chmod()	creat()	creat(), socket()
open		open()	connect(), bind(), open()
exec	exec()		
read		read()	read(), recv(), recvfrom()
write		write()	write(), send(), sendto()
close	kill(), exit()	close()	close()
destroy	rm(), chmod()	rm()	rm()

5.3 Definition of the Security Policy Language

Existing policy languages are not applicable to ubiquitous social systems, since they do not provide the appropriate policy abstractions and rules. Even though some of them are capable of being extended, it is more intuitive and straightforward to define a new policy language. In the rest of this section, a logic language is defined for the specification of security policies for ubiquitous social systems.

A logic language describes an abstract, simplified view of some selected part of the world, called the domain of discourse, by specifying the concepts involved and the relationships that exist between them regarding a particular purpose. The domain of discourse for the proposed language is derived from the concepts defined in the proposed security policy model and conceptual model of security context. In particular, it consists of the following finite sets: a set of subjects S , a set of objects O , a set of operations OP , a set of social groups SG , a set of property values PrV , a set of security contexts Ctx and a set of interactions during an activity Int . Formally, the proposed language is a dialect of Datalog [92]. This means that it is a first-order logic language with its own syntax, which conforms to the syntax of Datalog, but which utilizes the semantics of Datalog. This enables the specification of more concise policy rules, the correctness and consistency of which is guaranteed by Datalog. As discussed in [99], many other security policy languages are also based on Datalog, for example the Flexible Authorization Framework [45], Cassandra [100] and Protune [101].

The syntax of the proposed policy language is defined below and its semantics described with respect to Datalog and first-order logic. Following this, the language is used to specify the various security requirements for ubiquitous social systems.

5.3.1 Syntax of the Language

The syntax of a logic language determines which collection of symbols constitutes legal expressions within it. Since the proposed language is a dialect of Datalog, it supports two types of legal expressions: terms and rules. A term represents an object from the domain of discourse and is either a constant or a variable symbol. A rule, on the other hand, expresses a statement about terms and takes the form $L_0 \Leftarrow L_1 \wedge L_2 \wedge \dots \wedge L_n$, where each L_i , $0 \leq i \leq n$ is called a literal. Each literal is of the form $p_i(t_1, \dots, t_{m_i})$, such that p_i denotes a predicate symbol and t_j denotes a term symbol. The left-hand side of a rule is usually called the head, while the right-hand side is called the body of the rule. Furthermore, the terms of a literal are usually called arguments of the literal's predicate. The constant, variable and predicate symbols form the alphabet of the language, together with the logic symbols for connecting the literals within rules.

The following alphabet is used in the proposed language.

Constant symbols. A finite set of constant symbols are defined for each set from the domain of discourse. These are the sets $\mathcal{S}$, $\mathcal{O}$, $\mathcal{OP}$, $\mathcal{CTX}$, $\mathcal{SG}$, $\mathcal{PRV}$, $\mathcal{INT}$, which contain constant symbols for each element in the sets S , O , OP , Ctx , SG , PrV , and Int , respectively. Each constant symbol explicitly denotes a particular element from the related set of the domain of discourse. For example, the symbol $s_1 \in \mathcal{S}$ denotes a particular subject from the set of subjects S . Each constant symbol is represented as a string beginning with a small letter.

Variable symbols. A finite set of variable symbols is defined ranging over each set from the domain of discourse. In particular, these are $\mathcal{V}_S$, $\mathcal{V}_O$, $\mathcal{V}_{OP}$, $\mathcal{V}_{CTX}$, $\mathcal{V}_{SG}$, $\mathcal{V}_{PRV}$, $\mathcal{V}_{INT}$, which range over the sets S , O , OP , Ctx , SG , PrV and Int respectively. Each variable symbol denotes any element from the related set of the domain of discourse. For example, the symbol $X_S \in \mathcal{V}_S$ denotes any subject from the set of subjects S . Each variable symbol is represented as a string beginning with a capital letter.

Predicate symbols. The following terminology is used to define predicate symbols. A constant and a variable symbol which represent an element from the same set of the domain of discourse are referred to as terms of that set. For example, the constant s_1 and the variable X_S which represent an element from the set of subjects S are referred to as subject terms. In addition, object and group terms are collectively referred to as asset terms⁷.

The following predicate symbols are defined in the proposed language.

- *allow* is a ternary predicate. Its first argument is a subject term, its second is an operation term and its third is an object term. This predicate represents an activity which is allowed to be performed, as identified by the subject, operation and object term.
- *gallow* is a five-term predicate. Its first argument is a social group term, its second argument is an interaction term, and the remaining three arguments are the same as for the *allow* predicate: subject, operation and object terms. This predicate represents an interaction during an activity which is allowed to be performed by the social group.
- *uallow* is also a five-term predicate. Its first argument is an asset term, and the remaining four arguments are the same as in the *gallow* predicate. It represents an interaction during an activity which is allowed to be performed by the user.

⁷The asset terms denote all the possible assets which can be involved in an activity. A social group in this case denotes the user, participants or observers as asset, whereas object terms denote also the subjects, since subjects are objects.

- *org* is a binary predicate. Its first argument is a social group term, and its second argument is an object term. This predicate represents the origin social group of an object.
- *cctx* is a four-term predicate. Its first argument is a context term, and the remaining arguments are subject, operation and object terms, respectively. This predicate represents the current context of an activity.
- *foc*, *asc* and *set* are binary predicates. Their first argument is a social group term, and their second argument is a context term. They represent that a social group is the focus, association or setting element of a given context, respectively.
- *confO*, *intO* and *avO* are also binary predicates. Their first argument is a security property value term, and their second argument is an object term. They represent that a value of a security property is the confidentiality, integrity and availability of the object, respectively.
- “=” is a special binary predicate. Its first argument is a variable, and its second argument is a constant; both of these refer to elements of the same set from the domain of discourse. This predicate represents that two elements, denoted by the variable and the constant, are the same. The infix notation is used for this predicate, unlike the other predicates, for which the prefix notation is used.

Logic symbols. In addition, the following symbols are used as logic symbols: “ $\leftarrow$ ”, “ $,$ ” and “ $-$ ”. The first symbol represents the standard logical connective for implication, the second for conjunction and the third for negation. These are used for building the rules in the proposed language.

There are three types of rules that can be expressed in this language: group, asset and global rules. These constitute group, asset and global policies, respectively.

Group rules. A group rule contains the *gallow* predicate in its head and any combination of *org*, *cctx*, *foc*, *asc*, *set*, = and the security property predicates in its body. It states the necessary conditions for allowing a particular interaction during an activity with respect to a particular social group. It is specified for a particular operation, which is identified by a constant, and for any subject and object, which are represented by a variable. This means that it applies to a set of activities rather than to a single activity. The particular social group which specifies a group rule, as well as the particular interaction, are represented by constants in the predicate. Consider the following group rule as an example:

$$\begin{aligned} gallow(work, sp, S, read, O) \leftarrow & org(work, S), \\ & cctx(C_2, S, read, O), asc(work, C_2). \end{aligned}$$

This rule states the conditions for allowing the subject-participant interaction identified by the interaction constant $sp \in \mathcal{I}$ during an activity of the type identified by the operation constant $read \in \mathcal{OP}$, with respect to the social group identified by the social group constant $work \in \mathcal{SG}$. In particular, it states that any subject represented by the variable $S \in \mathcal{V}_S$ is allowed to read an object represented by the variable $O \in \mathcal{V}_O$ if its origin is the *work* social group and the association of a certain context represented by the variable $C_2 \in \mathcal{V}_C$, which is the current context of the activity, is also the *work* social group. In a nutshell, this rule states that the work social group allows the subject-participant interaction during a read activity only if both the subject’s origin and the association of the activity are the work social group.

Asset rules. An asset rule contains *uallow* predicate in its head and any combination of the *org*, *cctx*, *foc*, *asc*, *set*, *=* and the security property predicates in its body. It states the necessary conditions for allowing a particular interaction during an activity with respect to a particular asset. It is specified for a particular operation, which is identified by a constant, and for any subject and object, which are represented by a variable. This means that it applies to a set of activities rather than to a single activity. The particular asset for which an asset rule is specified, as well as the particular interaction, are represented by constants in the predicate. Asset rules are specified by the user. Consider the following asset rule as an example:

$$\begin{aligned} uallow(s_1, sp, S, read, O) \leftarrow & (S, s_1), org(work, S), \\ & cctx(C, S, read, O), asc(public, C). \end{aligned}$$

This rule states the conditions for allowing the subject-participant interaction identified by the interaction constant $sp \in \mathcal{I}$ during an activity of the type identified by the operation constant $read \in \mathcal{OP}$, with respect to the particular asset identified by the asset constant $s_1 \in \mathcal{S}$. In particular, it states that any subject represented by the variable $S \in \mathcal{V}_S$ is allowed to read an object represented by the variable $O \in \mathcal{V}_O$ if it equals the particular asset s_1 , its origin is the *work* social group and the association of a certain context represented by the variable $C \in \mathcal{V}_C$, which is the current context of the activity, is the social group identified by the constant *public* $\in \mathcal{SG}$. In brief, this rule states that the user allows the subject-participant interaction of a particular subject s_1 during a read activity only if the association of the activity is the public social group.

Global rules. A global rule contains the *allow* predicate in its head and any combination of the other predicates in its body. It states the necessary conditions for allowing, denying or terminating an activity according to the combined group rules from all concerned social groups. It is usually specified for any subject, operation and object, which are each represented by a variable. Thus, it usually applies to a set of activities and rather than to a single one. Consider for example the following composition rule:

$$\begin{aligned} allow(S, read, O) \leftarrow & org(G_1, S), org(G_2, O), \\ & gallow(G_1, so, S, read, O), gallow(G_2, so, S, read, O). \end{aligned}$$

This states that any subject represented by $S \in \mathcal{V}_S$ can perform a read operation identified by the constant $read \in \mathcal{OP}$ on any object $O \in \mathcal{V}_O$ if the subject-object interaction identified by the constant $so \in \mathcal{I}$ is allowed for this activity by the social groups in the origin of the subject and the object. Since the origin social groups of the subject and the object are represented by the variables $G1 \in \mathcal{V}_{SG}$, this rule applies to any social group and consequently to any subject and object, rather than only to those originating from particular social groups. In brief, this rule states that a subject is allowed to read an object only if the subject-object interaction during the read activity is allowed by the origin social groups of both the subject and the object.

5.3.2 Semantics of the Language

The semantics of a logic language determines the meaning behind the legal expressions of its syntax. Since the syntax of the proposed language conforms to the syntax of Datalog, the semantics of Datalog consequently applies to the new language by default. In this subsection, the semantics of Datalog are briefly described and related to the syntax of the proposed language. A detailed formal definition of this can be found in [92].

The purpose of Datalog is to enable deductions of new facts from facts which are already known, according to defined rules. Let E denote the set of all facts which are known, referred to as the environment, and let P denote the set of all rules which are defined, referred to as the program⁸. Let the following requirements restrict which predicates can be contained in E and P : (i) each predicate contained in P must not appear in E ; and (ii) a predicate contained in E can appear only in the body of a rule contained in P , and not in its head. A new fact, referred to as a result, is a literal made up of predicates which appear in the head of the rules in P , but which do not exist in P or in E . Let R denote the set of all results which can be deduced from E and P . Often, only a subset of the resulting facts is of interest for deduction and this is therefore specified as a special concept, the goal, $g \subseteq R$, of the program P .

The concepts of Datalog can be easily translated into first-order logic. Each fact f is an atomic formula f^* and each rule p is a formula p^* of the form $\forall X_1, \dots, \forall X_m (L_0 \leftarrow L_1 \wedge L_2 \wedge \dots \wedge L_n)$, where $X_1, \dots, X_m$ are all the variables occurring in p . Consequently, the environment E corresponds to a conjunction E^* of all the formulas f^* such that $f \in E$, whereas the program P corresponds to a conjunction P^* of all the formulas p^* such that $p \in P$.

The semantics of a Datalog program P can be described as a mapping $S_P : 2^{EA} \rightarrow 2^{RA}$, where EA denotes the set of all possible facts which can be part of the environment and RA denotes the set of all possible facts which can be part of the resultant facts. The semantics S_P associates each environment $E \subseteq EA$ with a set of resultant facts $R \subseteq RA$, such that the corresponding atomic formula r^* of each result fact $r \in R$ is a logical consequence of E^* and P^* , $E^* \wedge P^* \models r^*$. If a goal g is given, then the semantics of the program P with respect to g becomes a mapping $S_{P,g} : 2^{EA} \rightarrow 2^{RA}$, such that $g \subseteq S_P(P)$.

The entailment of r^* involves the evaluation of the truth value of the formula $r^* \leftarrow E^* \wedge P^*$. This formula must be true for each interpretation, that is for each meaning assigned to the constants and the predicates. This means that if each formula f^* from E^* and p^* from P^* is true under an interpretation, then r^* must also be true. Since rules usually contain variables, the entailment process needs to be performed for each substitution of variables with appropriate constants.

The syntax of the proposed language is related to the semantics of Datalog as follows. A policy evaluation regarding a request for an activity (s, op, o) corresponds to a deduction of the fact that the requested activity is allowed to be performed, i.e., that the particular subject is allowed to perform the particular operation on the particular object, $allow(s, op, o)$. This fact corresponds to a goal in a Datalog program. Naturally, this deduction requires an environment and a program in order to be carried out. An environment in Datalog corresponds to a set of facts which contain the predicates defined in the proposed language. It represents the system state at the moment of the evaluation, as defined in the policy model proposed above, together with the current context of the requested activity. On the other hand, a program in Datalog corresponds to the group, asset and global rules specified using the new language. This includes the rules within all group, asset and global policies. The syntactical elements of the proposed language can be translated to first-order logic in the same way as the Datalog ones. The policy evaluation thus becomes the entailment of the atomic formula that the requested activity is allowed to be performed.

Example. Consider as an example the following syntactical elements specified with the proposed language.

- Let $\mathcal{S} = \{app_1\}$, $\mathcal{O} = \{app_1, d_1\}$, $\mathcal{OP} = \{create, open\}$, $\mathcal{SG} = \{work, empty\}$, $\mathcal{PV} = \{0, 1\}$, $\mathcal{I} = \{so, sp\}$, and $\mathcal{C} = \{c_1, c_2, c_3, \dots\}$ be the sets of constant symbols. These

⁸In Datalog, the set of known facts is called the Extensional Database (EDB) and the set of defined rules is called a program P or an Intensional Database (IDB).

indicate that there are two objects in the system state, app_1 and d_1 , and one subject, app_1 , which can perform only *create* and *open* operation activities. They also denote that two social groups are defined (*work* and *empty*), there are two possible interactions during an activity which need to be controlled (subject-object (*so*) and subject-participant (*sp*) interactions) and that the security properties can have two values, which simply denote whether they are provided (1) or not (0). The set of possible contexts contains all the possible combinations of these elements, among which, there is a particular context: $c1 = ((app_1, open, d_1), work, empty, empty, 1, 0, 0, 0, 0, 0)$.

- Let $\mathcal{V}_S = \{S\}$, $\mathcal{V}_O = \{O\}$, $\mathcal{V}_{OP} = \{OP\}$, $\mathcal{V}_C = \{C_1, C_2\}$, $\mathcal{V}_{SG} = \{G\}$ be the sets of variable symbols. These denote the variable symbols that will be used in the policy rules to denote any symbol from the related sets of constant symbols.
- Let the following group rules be part of the group policy of the *work* social group.

$$gallow(work, so, S, open, O) \leftarrow org(work, S), org(work, O). \quad (5.1)$$

$$gallow(work, sp, S, open, O) \leftarrow org(work, S), \\ cctx(C, O), asc(empty, C). \quad (5.2)$$

Rule (5.1) represents the conditions for an open activity to be allowed in respect of a subject-object interaction, as defined by the *work* social group; it states that a subject can open an object only if they originate from the same social group. Furthermore, rule (5.2) also represents the conditions for an open activity to be allowed as defined by the *work* social group, but with respect to the subject-participant interaction. It states that a subject originating from the *work* social group can open an object only if the association element of the current context of this activity is the *empty* social group.

- Let the following global rule be part of the global policy.

$$allow(S, open, O) \leftarrow org(G_1, S), org(G_2, O), \\ gallow(G_1, so, S, open, O), gallow(G_2, so, S, open, O), \\ gallow(G_1, sp, S, open, O), gallow(G_2, sp, S, open, O). \quad (5.3)$$

This rule states that a subject can open an object only if the conditions for allowing this activity are satisfied, with respect to both the subject-object and subject-participants interactions, as defined by the social groups from which the subject and the object originate.

- Finally, let the following facts be part of the environment.

$$org(work, app_1). \quad (5.4)$$

$$org(work, d_1). \quad (5.5)$$

$$cctx(c_1, app_1, open, d_1). \quad (5.6)$$

Facts (5.4) and (5.5) indicate the origin of app_1 and d_1 , respectively, and fact (5.6) indicates the current context of the requested activity ($app_1, open, d_1$).

After translating these rules into first-order logic, the policy evaluation becomes the entailment of the atomic formula $allow(app_1, open, d_1)$, that is, the evaluation of the truth value of the following formula⁹

$$\forall S \in \mathcal{S}, \forall O \in \mathcal{O}, \forall OP \in \mathcal{OP}, \forall G \in \mathcal{SG}, \forall C \in \mathcal{C} \\ (allow(app_1, open, d_1) \leftarrow (5.1)^* \wedge (5.2)^* \wedge (5.3)^* \wedge (5.4)^* \wedge (5.5)^* \wedge (5.6)^*) \quad (5.7)$$

5.4 Evaluation of the Policy Model and Language

The applicability of the proposed policy model and language is demonstrated here empirically. Firstly, the proposed policy model is used to analyse ubiquitous social systems. Secondly, the proposed policy language is used to specify various templates of security policies applicable to ubiquitous social systems. Thirdly, the policy model and language are applied to the use cases defined in Section 3.4. As part of this analysis, the possible operation of a context-aware security system (CASS) that utilizes the proposed model is described.

5.4.1 Application of the Model within Ubiquitous Social Systems

In order to demonstrate the applicability of the proposed policy model, security relevant interactions between assets are analysed for the activities supported by the model. Each of these interactions can easily be translated into interactions between social groups by mapping assets to their originating social groups. The analysis identifies relevant threats to assets during these interactions, together with potential consequences for their originating social groups. For this purpose, the taxonomy of threats described in Section 2.1.2 is used.

The protection of assets from the perspective of a social group can be seen as analogous to protection of citizens from the perspective of a country. A country defines how its citizens can interact with each other and with the citizens of other countries. Each country controls these interactions by controlling the movement of citizens, both native and foreign, across the country's borders. For example, in order to interact with citizen y from another country, Y , citizen x needs to cross the border of his/her own country, X , into Y , where the interaction will take place. Both countries can control this interaction by preventing or allowing the movement of x at their border crossing points. Country X can control the movement of x upon leaving the country, while country Y can control the movement of x upon entering the country (Fig. 5.6). In order for an interaction to take place, the movement of x from country X to country Y must be allowed by both X and Y at the same time. When this is translated to ubiquitous social systems, a country becomes a social group and the citizens of that country become the assets of interest to that social group. The citizen x who initiates an interaction can be considered as the subject which performs the activity during which the interaction happens. Therefore, in order for an interaction between two assets to be allowed, the activity during which this interaction takes place must be allowed by both of the social groups related to the assets involved. As a result, each interaction can be seen from two different aspects, one for each of the social groups involved, and different threats and consequences may apply to each aspect.

The possible interactions according to the proposed model are described below. Fig. 5.7 illustrates the main blocks used to represent these interactions graphically. It should be noted that each graphical representation of an interaction represents the assets after the activity has been performed.

⁹Please note that $((RuleNumber))^*$ stands for $((RuleNumber))$ translated into first-order logic.

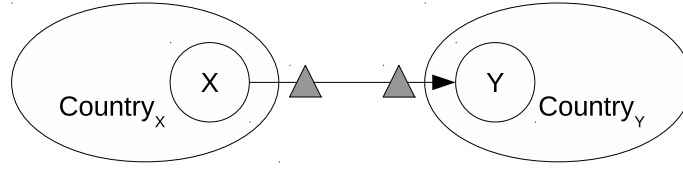


Figure 5.6: Border-crossing analogy.

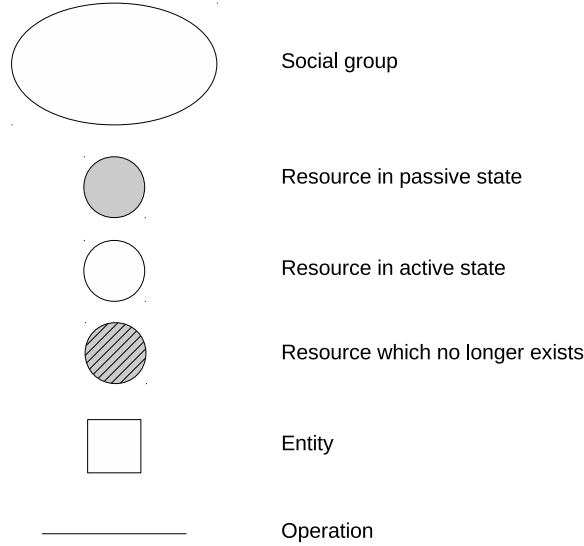


Figure 5.7: Components used in the graphical representation of interactions.

5.4.1.1 Subject-Object Interactions

The concept of a subject-object interaction stems directly from the concept of the particular type of activity during which it happens. Since there is a different interaction for each type of activity, each of these are analyzed separately. S and O are used to denote the subject and the object, and A and B are used to denote the origin social group of the subject and the object, respectively.

Create activity: The interaction during this type of activity consists of S creating a new object O . Translated to social groups, this interaction denotes the creation of a new asset in social group B , performed by an asset from social group A (Fig. 5.8). According to the proposed policy model, each newly created object inherits its origin social group from the subject which created it, meaning that S and O will always originate from the same social group, that is, $A = B$. As a result, this interaction does not have any security consequences.

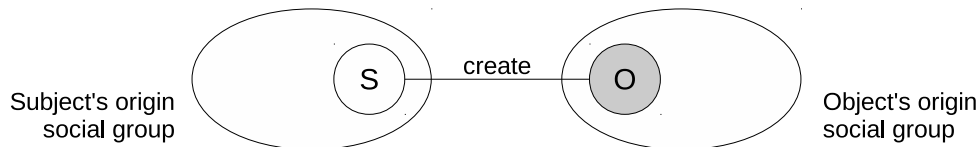


Figure 5.8: Subject-object interaction during the create activity.

Destroy activity: The interaction during this type of activity consists of S destroying O . Translated to social groups, this denotes destruction of an asset in social group B ,

performed by an asset from social group A (Fig. 5.9). This interaction is relevant only from the perspective of social group B , as it clearly poses a threat to the availability of the information contained in O , in the case of data and channel, and of the functionality of O , in the case of an application. For example, a malicious subject from social group A can intentionally destroy O , or the user can accidentally destroy this while A is her/his focus. The consequence in both of these cases is disruption—the information contained in O , or the functionality it provides, will no longer be available to the members of B .

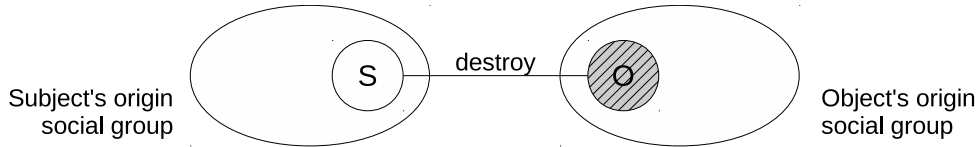


Figure 5.9: Subject-object interaction during the destroy activity.

Open activity: The interaction during this type of activity consists of S activating O so that it can access the information contained in O , which is either data or a channel. When their social groups are taken into account, this becomes the activation of an asset from social group B by an asset from social group A (Fig. 5.10). This interaction does not itself pose a threat to either A or B , because the actual usage of O has not yet taken place. However, it should be allowed to happen only if S is allowed to perform read or write activities upon O .

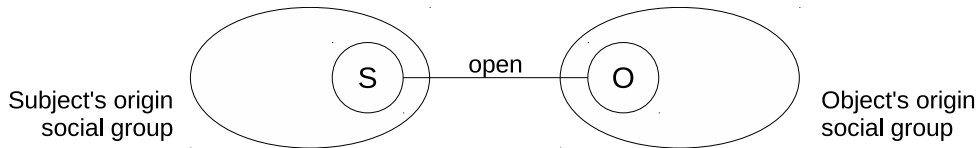


Figure 5.10: Subject-object interaction during the open activity.

Execute activity: The interaction during this type of activity consists of S starting O so that its functionality can be utilized. When their social groups are taken into account, this becomes the starting of an asset from social group B by an asset from social group A (Fig. 5.11). From the perspective of social group A this poses a threat to the confidentiality of the information contained in S and can result in unauthorized disclosure. For example, a malicious S , which contains certain information related to A , while providing functionality for it, can expose this information to the members of B by writing it as input information to O during start-up. On the other hand, from the perspective of social group B this interaction poses a threat to the confidentiality and integrity of the functionality O provides and can result in unauthorized disclosure or disruption. The former threat occurs because S opens O , probably in order to achieve some goals related to A ; however O is licensed to B and its functionality should be used only for achieving the goals of B . The latter threat occurs if S is malicious and starts O incorrectly, which can affect the correctness of the goals achieved by O .

Close activity: The interaction during this type of activity consists of the subject deactivating the object so that its usage is terminated. When their social groups are taken into account, this becomes a deactivation of an asset in social group B by an asset in social group A (Fig. 5.12). This interaction is relevant only from the perspective of social group B , as it poses a threat to the availability of O and can result in disruption. For example, a malicious subject in social group A can intentionally close O , and thus the information

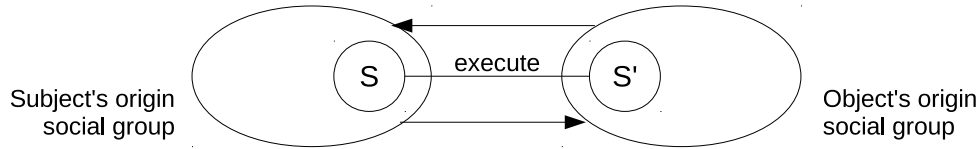


Figure 5.11: Subject-object interaction during the execute activity. (The terminology S' denotes that the object becomes a subject after execution.)

contained in O , or the functionality it provides, will no longer be available to the assets in B .

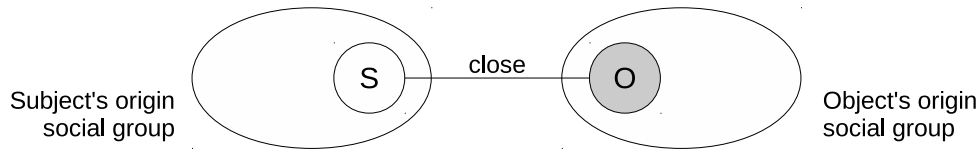


Figure 5.12: Subject-object interaction during the close activity.

Read activity: The interaction during this type of activity consists of S reading information contained in O , which results in an information flow from O to S . For example, this may be information stored previously in O , either during some activity, in the case of data, or during a message exchange, in the case of a channel. When this interaction is translated to social groups, it becomes an information flow from social group B to social group A (Fig. 5.13). From the perspective of social group A , this means the importing of information from B , which poses threat to the integrity of the functionality S provides and can result in deception. For example, if the information in O is false, the functionality of S may be incorrect, and this can reflect on the correctness of the goals achieved by the members of A . On the other hand, from the perspective of social group B , this means a leakage of information to A , which poses a threat to the confidentiality of the information O contains and can result in unauthorized disclosure. For example, S could later expose the information obtained from O , which is related to social group B , to the members of A .

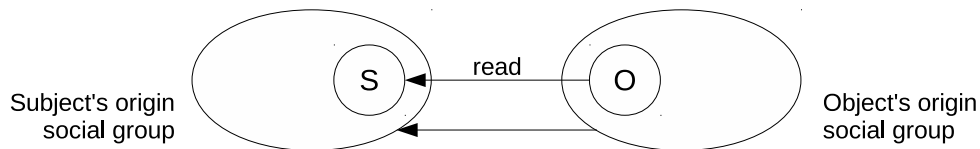


Figure 5.13: Subject-object interaction during the read activity.

Write activity: The interaction during this type of activity consists of S modifying the information contained in O , which results in an information flow from S to O . For example, this may be information stored in O , either for future use in some activity, in the case of data, or for message exchange, in the case of a channel. When this interaction is translated to social groups, it becomes an information flow from social group A to social group B (Fig. 5.14). From the perspective of social group A , this means the leakage of information to B , which poses a threat to the confidentiality of the information S contains and can result in unauthorized disclosure. For example, a malicious S , which contains certain information related to A while providing functionality for it, can expose this information to the members of B by writing to its object O . On the other hand, from the perspective of social group B this means importing information from A , which

poses a threat to the integrity of the information O contains and can result in deception or disruption. For example, a malicious S can falsify or corrupt the information contained in O in order to cause incorrect functionality in B 's subjects later on.

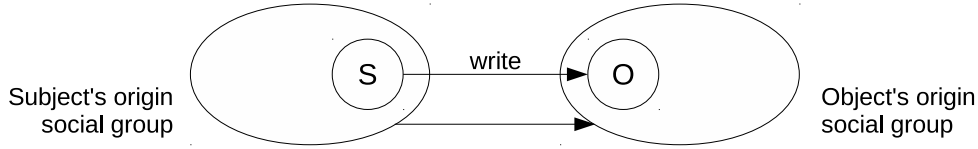


Figure 5.14: Subject-object interaction during the write activity.

5.4.1.2 Subject-User Interactions

This interaction describes the motivation of the user in using the subject to perform an activity of some particular type. If the subject and the user are denoted as S and U , and the subject's origin social group and the social group in the user's focus as A and B , respectively, this interaction can be translated as U using a subject from social group A to perform an activity oriented towards achieving the goals of social group B (Fig. 5.15).

From the perspective of social group A , this interaction poses a threat to the confidentiality of the functionality provided by S or the information contained in it, and can result in unauthorized disclosure and usage. For example, U may use S intentionally to achieve the goals of B , although S is licensed to A and its functionality should be used only to achieve the goals of A . During this, U may even accidentally expose information related to A to members of B . On the other hand, from the perspective of social group B , this poses threats to: (i) the confidentiality of the information contained in its objects and can result in their unauthorized disclosure; and (ii) the integrity and availability of both the information contained in its assets and the functionality they provide, and can result in their disruption.

For example, U can easily make errors while performing activities. First, U may believe that S originates from social group B and supply information to it which is related to B . However, S originates from A and can then provide this information to the members of A . Furthermore, U may accidentally delete or modify B 's objects or operate on B 's subjects incorrectly, due to incorrectly operating A 's subject S at the same time. Both of these scenarios can affect the correctness of the goals achieved for social group B . In addition, S may be malicious and may try to drain the hosts computational or battery power, or its memory or communication capabilities.

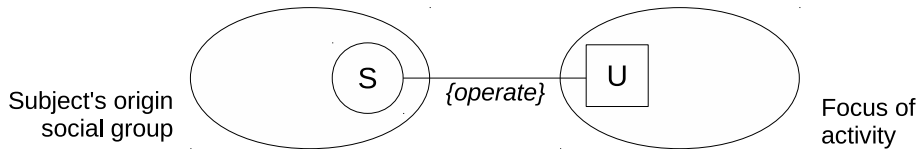


Figure 5.15: Subject-user interaction.

5.4.1.3 Subject-Participant Interactions

These interactions describe the direct involvement of the participants in the activities performed by the subject. They occur only during read and write activities through sensor

and network channels, as these are the only activities in which social entities can participate. All other activities, such as create, destroy, open, close and execute activities, and read and write activities through internal channels, take place within the host and do not involve participants. As a result, these activities do not pose any threat to the involved assets of interest. If the subject and the participants are denoted as S and P , and the subject's origin social group and the association of the activity as A and B , respectively, the subject-participant interactions can be analysed as follows.

Read activity: This interaction is similar to the subject-object interaction during the same type of activity. It consists of S receiving information from P , which results in an information flow from P to S . For example, this may be information sensed about P via the microphone channel, input by P through the touchscreen, or received from P through a network channel in the form of incoming messages. Translated to social groups, this interaction becomes an information flow from social group B to social group A (Fig. 5.16). From the perspective of social group A , this means importing information from B , which poses a threat to the integrity of the functionality provided by S and can result in deception. For example, if the information obtained from P is false, the functionality of S may be incorrect, and this may affect the correctness of the goals achieved for social group A . On the other hand, from the perspective of social group B , this means the leaking of information to A , which poses a threat to the confidentiality of the information from/about P and can result in unauthorised disclosure. For example, S can intercept a conversation between P and later provide this information to the members of A . In order to control this interaction: (i) social group A needs to specify in its policy the association of the participants to which its subjects are allowed to leak information and under which conditions; and (ii) social group B needs to specify the origin social group of the subjects which are allowed to import information from/about its members and under what conditions.

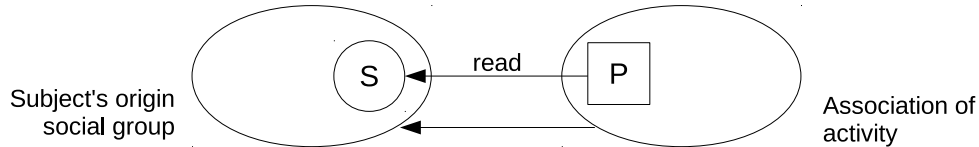


Figure 5.16: Subject-participant interaction during the read activity.

Write activity: This interaction is also similar to the subject-object interaction during the same type of activity. It consists of S sending information to P , which results in an information flow from S to P . For example, this may be information computed by S and shown to P on the screen, or sent to P via a network channel as outgoing messages. Translated to social groups, this interaction becomes information flow from social group A to social group B (Fig. 5.17). From the perspective of social group A , this means leakage of information to B , which poses a threat to the confidentiality of the information S contains and can result in unauthorised disclosure. For example, a malicious S , which contains certain information related to A while providing functionality for it, may expose this information to the members of B . On the other hand, from the perspective of social group B , this interaction means importing (consuming) information from A , which poses a threat to the integrity of the activities P performs and can result in deception and disruption. For example, a malicious S may falsify or corrupt the information sent to P in order to cause the incorrect achievement of B 's goals.

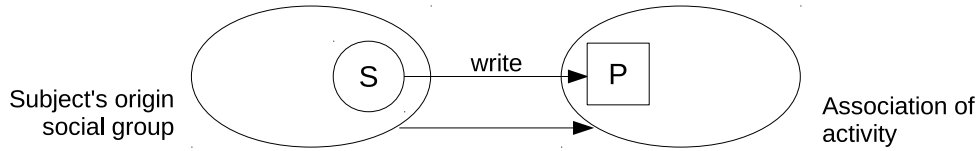


Figure 5.17: Subject-participant interaction during the write activity.

5.4.1.4 Subject-Observer Interactions

These interactions describe the influence of the surrounding environment consisting of the observers, on the subjects performing activities, and vice versa. Similarly to subject-participant interactions, these take place only during read or write activities through sensor and network channels, since other types of activities take place within the host and do not involve observers. For convenience, the subject and the observers are denoted as S and B , while the origin social group of the subject and the setting of the activity are denoted as A and C in describing these interactions (Fig. 5.18).

From the perspective of social group A , these interactions consist of B witnessing that certain activities are performed by S . Although B may play a key role in these activities by enabling them, for example as part of the communication link between the host and the participants, B may also easily become a participant in these activities. Therefore, these interactions pose a threat to the confidentiality of the information contained by S , as well as to the integrity of the functionality it provides. The consequences resulting from these threats are unauthorised disclosure and deception. For example, a malicious B can wiretap the environment in order to intercept confidential information exchanged between S and members of A . For this purpose, it can further falsify some of the information exchanged, which can also affect the correctness of functionality of S . In addition, a malicious B may try to spoof the user and masquerade as a legitimate user, in order to gain direct access to the functionality of S and the information it contains. On the other hand, from the perspective of social group C , these interactions consist of enabling the activities performed by S by providing computing and communication infrastructure, in form of devices which are members of B . This poses a threat to the integrity and availability of B , and can result in disruption and usurpation. For example, a member of social group A can use a malicious S , either deliberately or accidentally, which can interrupt a service provided by B or hinder its delivery to the other members of C . Moreover, S may misuse the authorisations of the user as a member of C and gain control of some of B 's services.

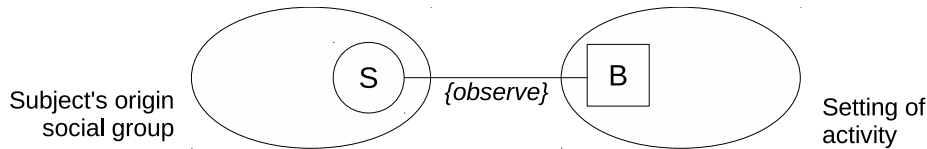


Figure 5.18: Subject-observer interaction.

5.4.2 Application of the Language within Ubiquitous Social Systems

In this section, the proposed policy language is used for the specification of security policies for ubiquitous social systems. The purpose of this is threefold. Firstly, it serves as an example of how the policy language can be applied in practice within the proposed security policy model. The possible contents of all three types of supported security policies, i.e.

group, asset and global policies, are specified. With respect to group and asset policies, a number of templates of security policies are specified which are intended to control a particular interaction between assets during an activity. All templates are based on the security analysis of the ubiquitous social system described in Section 5.4.1. With respect to global policy, an actual security policy is specified for combining the relevant group policies during each activity within ubiquitous social systems. Secondly, this section facilitates policy specification for the various policy makers. Each of the templates defined for group and user policies can be easily instantiated into readily applicable security policies. Since all of the policy rules in these templates are based on the security analysis presented in Section 5.4.1, each of the resulting policies are relevant to the protection of assets in ubiquitous social systems. Policy makers need to understand what kind of protection the defined templates offer, as well as the security risks involved in using these assets in each type of interaction. Following this, they can simply choose the most appropriate templates according to their needs. Thirdly, this section serves as a preparatory step for the evaluation of the policy model and language in the next section. Each of the specified templates and policies is applicable to the use cases defined in Section 3.4; defining them here makes the evaluation of the use cases in Section 5.4.3 more succinct and comprehensive.

A template is defined as a partially specified policy. It is a set of rules in which one or more constant symbols are left unspecified. These symbols are referred to here as parameters of the template. An important characteristic of these templates is that they are composed of rules which are recurring, that is, which are applicable to controlling activities in various situations of ubiquitous social systems and for various policy makers. A template becomes a complete security policy after appropriate constants have been substituted for all template parameters. As an example, consider a template composed of the following rules:

$$\begin{aligned} condSO(\langle group \rangle, S, open, O) &\leftarrow org(\langle group \rangle, S), org(\langle group \rangle, O). \\ condSO(\langle group \rangle, S, read, O) &\leftarrow org(\langle group \rangle, S), org(\langle group \rangle, O). \end{aligned}$$

This template has one parameter, $\langle group \rangle$, which stands for a social group constant symbol, $\langle group \rangle \in \mathcal{SG}$. The rules specify that a subject can open and read an object only if they both originate from the same social group; however, this social group is not yet specified in the template. These rules can be considered as recurring because they define a basic and intuitive security requirement, which is applicable to any social group, such as *work*, *home* or *public*. If a policy maker, e.g. the one responsible for the *work* social group, wants to instantiate this template as a complete policy, all that is needed is to substitute the $\langle group \rangle$ parameter with the constant symbol representing that particular social group, e.g. for *work*, $work \in \mathcal{SG}$. This results in a group policy represented by the following rules:

$$\begin{aligned} condSO(work, S, open, O) &\leftarrow org(work, S), org(work, O). \\ condSO(work, S, read, O) &\leftarrow org(work, S), org(work, O). \end{aligned}$$

5.4.2.1 Group Policy

Several templates of a group policy can be defined to control each possible interaction between assets during a single activity. These templates are based on the analysis presented in Section 5.4.1 and aim to mitigate the identified threats to the social group assets. Three types of group policy templates can be identified with respect to their restrictiveness:

- **Closed interaction templates:** These are the most restrictive templates. They allow assets related to a given social group to interact only with assets related to the same social group. With respect to the analogy drawn between social groups and countries, this enables protection of assets corresponding to a closed border.
- **Shared interaction templates:** These templates relax the restrictiveness of the closed interaction templates. They allow assets related to a given social group to interact with assets related to another particular social group. With respect to the analogy between social groups and countries, this gives protection of assets corresponding to a regulated border. While some direct interactions are allowed by these templates, additional asset policies need to be specified for other interactions before they are allowed. Asset policies in this case resemble the visas required to cross a country's borders.
- **Open interaction templates:** These are the least restrictive templates. They allow assets related to a given social group to interact with any asset, regardless of its relationship with a social group. This gives protection of assets corresponding to an open border.

5.4.2.1.1 Subject-Object Interaction Templates These templates define the conditions for allowing subject-object interactions in regard to a given social group $\langle this \rangle$. They can be defined from two perspectives: (a) from the perspective of $\langle this \rangle$ social group as the origin of the subject; and (b) from the perspective of $\langle this \rangle$ social group as the origin of the object.

5.4.2.1.1.1 Closed Subject and Object This template is defined from perspectives (a) and (b). It states that a subject is allowed to carry out an operation on an object if they both originate from the same (given) social group. It consists of the following rules:

$$gallow(\langle this \rangle, so, S, create, O) \leftarrow org(\langle this \rangle, S). \quad (5.8a)$$

$$gallow(\langle this \rangle, so, S, destroy, O) \leftarrow org(\langle this \rangle, S), org(\langle this \rangle, O). \quad (5.8b)$$

$$gallow(\langle this \rangle, so, S, open, O) \leftarrow org(\langle this \rangle, S), org(\langle this \rangle, O). \quad (5.8c)$$

$$gallow(\langle this \rangle, so, S, exec, O) \leftarrow org(\langle this \rangle, S), org(\langle this \rangle, O). \quad (5.8d)$$

$$gallow(\langle this \rangle, so, S, close, O) \leftarrow org(\langle this \rangle, S), org(\langle this \rangle, O). \quad (5.8e)$$

$$gallow(\langle this \rangle, so, S, read, O) \leftarrow org(\langle this \rangle, S), org(\langle this \rangle, O). \quad (5.8f)$$

$$gallow(\langle this \rangle, so, S, write, O) \leftarrow org(\langle this \rangle, S), org(\langle this \rangle, O). \quad (5.8g)$$

This template allows interactions only between subjects and objects from the same social group. As a result, information and functionality related to $\langle this \rangle$ social group remain within the group, which prevents the possible security consequences discussed in Section 5.4.1.1 from occurring.

5.4.2.1.1.2 Shared Object with Other Subject This template is defined only from perspective (b). It states that an operation can be carried out on an object which

originates from the given social group if the subject originates from a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, so, S, open, O) \leftarrow org(\langle other \rangle, S), org(\langle this \rangle, O), \\ cctx(C, S, open, O), not(asc(empty, C)). \end{aligned} \quad (5.9a)$$

$$\begin{aligned} gallow(\langle this \rangle, so, S, close, O) \leftarrow org(\langle other \rangle, S), org(\langle this \rangle, O), \\ cctx(C, S, close, O), not(asc(empty, C)). \end{aligned} \quad (5.9b)$$

$$\begin{aligned} gallow(\langle this \rangle, so, S, read, O) \leftarrow org(\langle other \rangle, S), org(\langle this \rangle, O), \\ cctx(C, S, read, O), not(asc(empty, C)). \end{aligned} \quad (5.9c)$$

$$\begin{aligned} gallow(\langle this \rangle, so, S, write, O) \leftarrow org(\langle other \rangle, S), org(\langle this \rangle, O), \\ cctx(C, S, write, O), not(asc(empty, C)). \end{aligned} \quad (5.9d)$$

This template applies only to sensor and network channels¹⁰ and promotes information flow between $\langle this \rangle$ and the $\langle other \rangle$ social group, in both directions. This is essential for subjects of $\langle this \rangle$ group to communicate through sensor or network channels with the help of system applications. As explained above, system subjects such as driver applications are responsible for such communication occurring through the various device sensors and network elements. This template enables these system subjects to read and write to sensor or network channels originating from $\langle this \rangle$ social group, and thus to operate as intended.

Although this poses threats to the confidentiality of information contained in the shared $\langle this \rangle$ objects and to integrity of $\langle this \rangle$ subjects that operate with them, the risk of consequences is low. The reason for this is twofold. Firstly, channels do not act as information storage, and leakage can occur only if some subject actively reads a channel while $\langle this \rangle$ subject writes to it. However, the proposed model allows social groups to control which subjects can read their objects (handled by these templates), and which subjects are active while they are in the focus of the user (handled by the subject-user interaction templates described below). As a result, this mitigates the possibility of information leakage. Secondly, system subjects such as driver applications are trusted to operate as expected. For example, this means that they transmit the information exchanged via channels to the specified device element, and do not save it somewhere on the device in order to leak it in the future. Nevertheless, approaches such as SE Linux [102] or SE Android [103] can provide finer-grained security policies for specifying which actions each system subject is allowed to take.

5.4.2.1.1.3 Shared Subject with Other Object This template is defined only from perspective (a). It states that a subject which originates from a given social group can carry out an operation on an object if the object originates from a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, so, S, open, O) \leftarrow org(\langle this \rangle, S), org(\langle other \rangle, O), \\ cctx(C, S, open, O), not(asc(empty, C)). \end{aligned} \quad (5.10a)$$

$$\begin{aligned} gallow(\langle this \rangle, so, S, close, O) \leftarrow org(\langle this \rangle, S), org(\langle other \rangle, O), \\ cctx(C, S, close, O), not(asc(empty, C)). \end{aligned} \quad (5.10b)$$

¹⁰It should be recalled that these channels differ from other objects in that activities involving them have associations and settings which are different from the empty social group. This is represented in the last condition in the rules: $not(asc(empty, C))$.

$$\begin{aligned} \text{gallow}(\langle this \rangle, so, S, read, O) &\leftarrow \text{org}(\langle this \rangle, S), \text{org}(\langle other \rangle, O), \\ &\quad \text{cctx}(C, S, read, O), \text{not}(\text{asc}(\text{empty}, C)). \end{aligned} \quad (5.10c)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, so, S, write, O) &\leftarrow \text{org}(\langle this \rangle, S), \text{org}(\langle other \rangle, O), \\ &\quad \text{cctx}(C, S, write, O), \text{not}(\text{asc}(\text{empty}, C)). \end{aligned} \quad (5.10d)$$

This template is similar to the template *Shared Object with Other Subject*, and also applies to sensor and network channels;¹¹ however, it represents the opposite perspective. It is intended for the system social group, so that system subjects such as driver applications can operate and enable communication via sensor and network channels for a subject originating from the $\langle other \rangle$ social group. The confidentiality of information is less of an issue for the system social group than the integrity of its subjects. However, system subjects are trusted, and the risk of security consequences is low. As mentioned above, approaches such as those in [102, 103], can provide additional security if needed.

5.4.2.1.2 Subject-User Interaction Templates These templates define the conditions for allowing subject-user interactions regarding a given social group $\langle this \rangle$. They can be defined from two perspectives: (a) from the perspective of $\langle this \rangle$ social group as the origin of the subject; and (b) from the perspective of $\langle this \rangle$ social group as the focus of the user, i.e. the focus of the activity which is controlled.

5.4.2.1.2.1 Closed Subject and Focus This template is defined from both aspects (a) and (b). It states that a subject is allowed to perform an activity in the given focus if it originates from the same social group as in the focus. It consists of the following rules.

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, create, O) &\leftarrow \text{org}(\langle this \rangle, S), \\ &\quad \text{cctx}(C, S, create, O), \text{foc}(\langle this \rangle, C). \end{aligned} \quad (5.11a)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, destroy, O) &\leftarrow \text{org}(\langle this \rangle, S), \\ &\quad \text{cctx}(C, S, destroy, O), \text{foc}(\langle this \rangle, C). \end{aligned} \quad (5.11b)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, open, O) &\leftarrow \text{org}(\langle this \rangle, S), \\ &\quad \text{cctx}(C, S, open, O), \text{foc}(\langle this \rangle, C). \end{aligned} \quad (5.11c)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, exec, O) &\leftarrow \text{org}(\langle this \rangle, S), \\ &\quad \text{cctx}(C, S, exec, O), \text{foc}(\langle this \rangle, C). \end{aligned} \quad (5.11d)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, close, O) &\leftarrow \text{org}(\langle this \rangle, S), \\ &\quad \text{cctx}(C, S, close, O), \text{foc}(\langle this \rangle, C). \end{aligned} \quad (5.11e)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, read, O) &\leftarrow \text{org}(\langle this \rangle, S), \\ &\quad \text{cctx}(C, S, read, O), \text{foc}(\langle this \rangle, C). \end{aligned} \quad (5.11f)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, write, O) &\leftarrow \text{org}(\langle this \rangle, S), \\ &\quad \text{cctx}(C, S, write, O), \text{foc}(\langle this \rangle, C). \end{aligned} \quad (5.11g)$$

This template restricts the user to using only subjects originating from the social group in the current focus while trying to achieve its goals. At the same time, it ensures that

¹¹This is specified by the last condition in the rules: $\text{not}(\text{asc}(\text{empty}, C))$.

subjects originating from a social group are used only for the purposes of that social group. This results in goals related to $\langle this \rangle$ social group being achieved with trustworthy apps and the functionality of these apps remaining only within the group. As a result, this prevents the possible security consequences discussed in Section 5.4.1.2 from occurring.

5.4.2.1.2.2 Shared Focus with Other Subject This template is defined only from aspect (b). It states that a subject is allowed to perform an activity when the focus is the given social group if it originates from a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, su, S, create, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, create, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.12a)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, destroy, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, destroy, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.12b)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, open, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, open, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.12c)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, exec, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, exec, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.12d)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, close, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, close, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.12e)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, read, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, read, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.12f)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, write, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, write, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.12g)$$

This template promotes $\langle other \rangle$ subjects to be active while the user strives to achieve the goals of $\langle this \rangle$ social group. Primarily, $\langle other \rangle$ applies to system subjects. Any social group needs to allow system applications to operate while it is in focus, so that its subjects can operate and utilize the device's internal components. Since system apps are assumed to be trusted, the risk of security consequences in this case is low. However, this does not apply to subjects originating from social groups other than the system, and an additional security assessment is needed prior to using this template with them.

5.4.2.1.2.3 Shared Subject with Other Focus This template is defined only from perspective (a). It states that a subject which originates from a given social group is allowed to perform an activity while the focus is a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, su, S, create, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, create, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.13a)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, destroy, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, destroy, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.13b)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, open, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, open, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.13c)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, exec, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, exec, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.13d)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, close, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, close, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.13e)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, read, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, read, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.13f)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, su, S, write, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, write, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.13g)$$

This template promotes $\langle this \rangle$ subjects to be active while the user strives to achieve the goals of $\langle other \rangle$ social group. Primarily, $\langle this \rangle$ applies to system applications. The system social group needs to allow system applications to operate in any focus, so that subjects originating from the social group in focus can operate as intended. Since the user is the only member of the system social group and is considered to be trusted, the risk of security consequences for the system social group is low. However, this does not apply to subjects originating from other social groups, and an additional security assessment is needed prior using this template with them.

5.4.2.1.3 Subject-Participant Interaction Templates These templates define the conditions for allowing subject-participant interactions regarding a given social group $\langle this \rangle$. They can be defined from two perspectives: (a) from the perspective of $\langle this \rangle$ social group as the origin of the subject; and (b) from the perspective of $\langle this \rangle$ social group as an association of the activity which characterizes the participants.

5.4.2.1.3.1 Closed Subject and Participant This template is defined from both aspects (a) and (b). It states that a subject which originates from a given social group is allowed to perform an activity if the association of that activity is the same (given) social group. It consists of the following rules.

$$\begin{aligned} \text{gallow}(\langle this \rangle, sp, S, open, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, open, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.14a)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sp, S, close, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, close, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.14b)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sp, S, read, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, read, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.14c)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sp, S, write, O) &\leftarrow org(\langle this \rangle, S), \\ &cctx(C, S, write, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.14d)$$

This template allows interactions between subjects and participants from the same social group. This results in information related to $\langle this \rangle$ social group remaining within the group, which prevents the possible security consequences discussed in Section 5.4.1.3 from occurring.

5.4.2.1.3.2 Shared Subject with Other Participant This template is defined only from perspective (a). It states that a subject which originates from a given social group is allowed to perform an activity if the association of that activity is a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sp, S, open, O) \leftarrow org(\langle this \rangle, S), \\ cctx(C, S, open, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.15a)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, close, O) \leftarrow org(\langle this \rangle, S), \\ cctx(C, S, close, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.15b)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, read, O) \leftarrow org(\langle this \rangle, S), \\ cctx(C, S, read, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.15c)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, write, O) \leftarrow org(\langle this \rangle, S), \\ cctx(C, S, write, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.15d)$$

This template allows subjects of a given social group to perform activities in which members from another social group participate. The template promotes information flow between $\langle this \rangle$ and $\langle other \rangle$ social groups. It is primarily intended to be used by the system social group and is necessary to enable subjects from another social group to operate with network and sensor channels. As an example, consider the activity of a network or sensor driver which reads or writes information to a channel with participants originating from $\langle this \rangle$ social group. Although these cases may possibly have implications for the integrity of the activities performed by members of $\langle this \rangle$ social group, the risk of these consequences is low, due to reasons similar to those explained in the *Shared Subject with Other Object* template.

5.4.2.1.3.3 Shared Participant with Other Subject This template is defined only from perspective (b). It states that a subject is allowed to perform a read or write activity while the association is the given social group, if it originates from a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sp, S, open, O) \leftarrow org(\langle other \rangle, S), \\ cctx(C, S, open, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.16a)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, close, O) \leftarrow org(\langle other \rangle, S), \\ cctx(C, S, close, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.16b)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, read, O) \leftarrow org(\langle other \rangle, S), \\ cctx(C, S, read, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.16c)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, write, O) \leftarrow org(\langle other \rangle, S), \\ cctx(C, S, write, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.16d)$$

This template allows members of a social group to participate in activities performed by subjects from another social group. The template promotes information flow between $\langle this \rangle$ and $\langle other \rangle$ social groups. It is primarily intended to be used with subjects originating from the system social group, and is necessary to enable operations through network and sensor channels. As an example, consider the activity of a network or sensor driver which reads

or writes information to a channel with participants originating from $\langle this \rangle$ social group. Although these cases may possibly have implications for the integrity of the activities performed by members of $\langle this \rangle$ social group, the risk of these consequences is low, due to reasons similar to those explained in the *Shared Object with other Subject* template.

5.4.2.1.3.4 Shared Subject with Other Participant (Incoming Flow) This template is defined only from perspective (a). It states that a subject which originates from a given social group is allowed to perform a read activity if the association of that activity is a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sp, S, open, O) &\leftarrow org(\langle this \rangle, S), \\ cctx(C, S, open, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.17a)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, close, O) &\leftarrow org(\langle this \rangle, S), \\ cctx(C, S, close, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.17b)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, read, O) &\leftarrow org(\langle this \rangle, S), \\ cctx(C, S, read, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.17c)$$

This template allows subjects originating from a social group to perform activities in which members of another social group participate. Since it applies only to read operations, it is essentially intended for activities involving channels via input sensors, such as a microphone, camera, touch etc.¹² The template promotes information flow from $\langle other \rangle$ to $\langle this \rangle$ social group, which may have implications for the subject's functionality. When an application's functionality does not depend on the information received from the input sensors, the risk of security consequences for $\langle this \rangle$ social group is low. For example, this is true for a *Voip* application, which emits the voice data received from the microphone only to the participants in the conversation. However, when an application's functionality depends on this received information, an additional security assessment is needed prior to using this template.

5.4.2.1.3.5 Shared Participant with Other Subject (Incoming Flow) This template is defined only from perspective (b). It states that a subject is allowed to perform a write activity while the association is the given social group if it originates from a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sp, S, open, O) &\leftarrow org(\langle other \rangle, S), \\ cctx(C, S, open, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.18a)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, close, O) &\leftarrow org(\langle other \rangle, S), \\ cctx(C, S, close, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.18b)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, write, O) &\leftarrow org(\langle other \rangle, S), \\ cctx(C, S, write, O), asc(\langle this \rangle, C). \end{aligned} \quad (5.18c)$$

This template allows members originating from a social group to participate in activities performed by subjects from another social group. Since it applies only to write operations,

¹²Any communication through network channels would also require a write operation to be possible for the members of the $\langle other \rangle$ social group.

it is essentially intended for activities involving channels via output sensors, such as the screen, speakers etc.¹³ The template promotes information flow from the $\langle other \rangle$ to $\langle this \rangle$ social group. This occurs when the user consumes resources from the host together with other participants, which are not members of the social group from which the resources originate. As an example, consider the activity of a user showing home photos to a work colleague. Although these cases may have implications for the integrity of the activities performed by members of $\langle this \rangle$ social group, the risk of these consequences is low, since these activities are usually performed with the knowledge of both the user and the participants in $\langle this \rangle$ social group.

5.4.2.1.3.6 Open Subject and Participant This template is defined from both perspectives (a) and (b). From perspective (a), it states that a subject which originates from a given social group is allowed to perform an activity in any association of that activity. On the other hand, from perspective (b), it states that any subject is allowed to perform an activity while the association of that activity is the given social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sp, S, open, O) &\leftarrow . \\ gallow(\langle this \rangle, sp, S, close, O) &\leftarrow . \end{aligned} \quad (5.19a)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, read, O) &\leftarrow . \\ gallow(\langle this \rangle, sp, S, write, O) &\leftarrow . \end{aligned} \quad (5.19b)$$

This template facilitates subject-participant interactions, as it does not place any restrictions on these. However, all of the security consequences discussed in 5.4.1 are applicable to the resources related to $\langle this \rangle$ social group as a result. This template is useful for a social group if there is a low possibility of attacks upon its resources.

5.4.2.1.4 Subject-Observer Interaction Templates These templates define the conditions for allowing subject-observer interactions regarding a given social group $\langle this \rangle$. They can be defined from two perspectives: (a) from the perspective of $\langle this \rangle$ social group as the origin of the subject; and (b) from the perspective of $\langle this \rangle$ social group as the setting of the activity, which characterizes the observers.

5.4.2.1.4.1 Closed Subject and Observer This template is defined from both perspectives (a) and (b). It states that a subject which originates from a given social group is allowed to perform an activity if the setting of that activity is the same social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sb, S, open, O) &\leftarrow org(\langle this \rangle, S), \\ &\quad cctx(C, S, open, O), set(\langle this \rangle, C). \end{aligned} \quad (5.20a)$$

$$\begin{aligned} gallow(\langle this \rangle, sb, S, close, O) &\leftarrow org(\langle this \rangle, S), \\ &\quad cctx(C, S, close, O), set(\langle this \rangle, C). \end{aligned} \quad (5.20b)$$

¹³Any communication through network channels would also require a read operation to be possible for the members of the $\langle other \rangle$ social group.

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, read, O) &\leftarrow org(\langle this \rangle, S), \\ &\quad cctx(C, S, read, O), set(\langle this \rangle, C). \end{aligned} \quad (5.20c)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, write, O) &\leftarrow org(\langle this \rangle, S), \\ &\quad cctx(C, S, write, O), set(\langle this \rangle, C). \end{aligned} \quad (5.20d)$$

This template allows interactions between subjects and observers from the same social group. This results in information related to $\langle this \rangle$ social group remaining within the group, which prevents the possible security consequences discussed in Section 5.4.1.3 from occurring.

5.4.2.1.4.2 Shared Subject with Other Observer This template is defined only from perspective (a). It states that a subject which originates from a given social group is allowed to perform an activity if the setting of that activity is a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, open, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, open, O), \\ &\quad set(\langle other \rangle, C), intO(1, O), confO(1, O). \end{aligned} \quad (5.21a)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, close, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, close, O), \\ &\quad set(\langle other \rangle, C), intO(1, O), confO(1, O). \end{aligned} \quad (5.21b)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, read, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, read, O), \\ &\quad set(\langle other \rangle, C), intO(1, O), confO(1, O). \end{aligned} \quad (5.21c)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, write, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, write, O), \\ &\quad set(\langle other \rangle, C), intO(1, O), confO(1, O). \end{aligned} \quad (5.21d)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, open, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, open, O), \\ &\quad set(\langle other \rangle, C), intO(-1, O), confO(-1, O). \end{aligned} \quad (5.21e)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, close, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, close, O), \\ &\quad set(\langle other \rangle, C), intO(-1, O), confO(-1, O). \end{aligned} \quad (5.21f)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, read, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, read, O), \\ &\quad set(\langle other \rangle, C), intO(-1, O), confO(-1, O). \end{aligned} \quad (5.21g)$$

$$\begin{aligned} \text{gallow}(\langle this \rangle, sb, S, write, O) &\leftarrow org(\langle this \rangle, S), cctx(C, S, write, O), \\ &\quad set(\langle other \rangle, C), intO(-1, O), confO(-1, O). \end{aligned} \quad (5.21h)$$

This template allows subjects originating from a given social group to perform activities which are observed by members from another social group. As described earlier (Section 4.4.1), this applies only to sensor and network channels, as it is only for these objects that the setting can be different from the *empty* social group. For network channels, these activities are allowed only if the confidentiality and integrity of the object are achieved. This is specified in the first four rules (a–d) using the $intO(1, O)$ and $confO(1, O)$ conditions. For sensor channels, these activities are always allowed, as specified in the last four rules (e–h) using the $intO(-1, O)$ and $confO(-1, O)$ conditions.¹⁴

¹⁴It is assumed that the confidentiality and integrity of sensor channels are both “-1”, meaning that they are undefined.

5.4.2.1.4.3 Shared Observer with Other Subject This template is defined only from perspective (b). It states that a subject is allowed to perform a read or write activity while the setting is the given social group, if it originates from a particular $\langle other \rangle$ social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sb, S, open, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, open, O), set(\langle this \rangle, C). \end{aligned} \quad (5.22a)$$

$$\begin{aligned} gallow(\langle this \rangle, sb, S, close, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, close, O), set(\langle this \rangle, C). \end{aligned} \quad (5.22b)$$

$$\begin{aligned} gallow(\langle this \rangle, sb, S, read, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, read, O), set(\langle this \rangle, C). \end{aligned} \quad (5.22c)$$

$$\begin{aligned} gallow(\langle this \rangle, sb, S, write, O) &\leftarrow org(\langle other \rangle, S), \\ &cctx(C, S, write, O), set(\langle this \rangle, C). \end{aligned} \quad (5.22d)$$

This template allows members originating from a given social group to observe activities performed by subjects from another social group. This occurs when an entity consumes resources from its host, while being surrounded by observers which are not members of the social group from which the resources originate. As an example, consider the activity of an employee looking at home photos while at work. Although these cases may have some implications for security (as described in Section 5.4.1.4), the risk of these consequences is low, since these activities are usually performed with the knowledge of both the entities performing them and the social group containing the observers.

5.4.2.1.4.4 Open Subject and Observer This template is defined from both perspectives (a) and (b). From perspective (a), it states that a subject which originates from a given social group is allowed to perform an activity in any setting of that activity. On the other hand, from perspective (b), it states that any subject is allowed to perform an activity while the setting of that activity is a given social group. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sb, S, open, O) &\leftarrow . \\ gallow(\langle this \rangle, sb, S, close, O) &\leftarrow . \end{aligned} \quad (5.23a)$$

$$\begin{aligned} gallow(\langle this \rangle, sb, S, read, O) &\leftarrow . \\ gallow(\langle this \rangle, sb, S, write, O) &\leftarrow . \end{aligned} \quad (5.23b)$$

This template facilitates subject-observer interactions, as it does not place any restriction on these. However, all of the security consequences discussed in Section 5.4.1 are applicable to the resources related to $\langle this \rangle$ social group as a result. The template is useful for a social group if there is a low possibility of attacks upon its resources.

5.4.2.1.5 Interaction Templates with Relaxation These templates define the conditions for allowing interactions with relaxation, regarding a given social group $\langle this \rangle$.

5.4.2.1.5.1 Shared Subject with Other Focus, with Relaxation This template is defined from perspective of $\langle this \rangle$ social group as the origin of the subject. It states that a subject which originates from a given social group is allowed to perform an activity while the focus is on a particular $\langle other \rangle$ social group, if this is allowed according to an asset policy specified for that particular subject. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, su, S, create, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, create, O), foc(\langle other \rangle, C), \\ & uallow(S, su, S, create, O). \end{aligned} \quad (5.24a)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, destroy, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, destroy, O), foc(\langle other \rangle, C), \\ & uallow(S, su, S, destroy, O). \end{aligned} \quad (5.24b)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, open, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, open, O), foc(\langle other \rangle, C), \\ & uallow(S, su, S, open, O). \end{aligned} \quad (5.24c)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, exec, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, exec, O), foc(\langle other \rangle, C), \\ & uallow(S, su, S, exec, O). \end{aligned} \quad (5.24d)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, close, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, write, O), foc(\langle other \rangle, C), \\ & uallow(S, su, S, close, O). \end{aligned} \quad (5.24e)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, read, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, read, O), foc(\langle other \rangle, C), \\ & uallow(S, su, S, read, O). \end{aligned} \quad (5.24f)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, write, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, write, O), foc(\langle other \rangle, C), \\ & uallow(S, su, S, write, O). \end{aligned} \quad (5.24g)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, exec, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, exec, O), foc(\langle other \rangle, C), \\ & uallow(O, su, S, exec, O). \end{aligned} \quad (5.24h)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, close, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, write, O), foc(\langle other \rangle, C), \\ & uallow(O, su, S, close, O). \end{aligned} \quad (5.24i)$$

This template promotes $\langle this \rangle$ subjects to be active, while the user strives to achieve the goals of the $\langle other \rangle$ social group. It is similar to the *Shared Subject with Other Focus* template, but with two differences. Firstly, it includes an additional condition (*uallow*), which specifies that approval from the user is needed in the form of an asset policy.¹⁵ Secondly, it includes two further rules, which allow the $\langle this \rangle$ subject to remain active

¹⁵A template is defined for this type of asset policy in the next section.

when the focus changes to the $\langle other \rangle$ social group.¹⁶ With this template, the security decisions are “transferred” to the user; if s/he issues appropriate asset policies, then these activities are allowed.

5.4.2.1.5.2 Shared Focus with Other Subject, with Relaxation This template is defined from the perspective of $\langle this \rangle$ social group as the focus of the user, i.e. the focus of the activity which is controlled. It states that a subject which originates from a particular $\langle other \rangle$ social group is allowed to perform an activity while the focus is the given social group, if this is allowed according to an asset policy specified for the particular focus. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, su, S, create, O) &\leftarrow org(\langle other \rangle, S), \\ &\quad cctx(C, S, create, O), foc(\langle this \rangle, C), \\ &\quad uallow(\langle this \rangle, su, S, create, O). \end{aligned} \quad (5.25a)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, destroy, O) &\leftarrow org(\langle other \rangle, S), \\ &\quad cctx(C, S, destroy, O), foc(\langle this \rangle, C), \\ &\quad uallow(\langle this \rangle, su, S, destroy, O). \end{aligned} \quad (5.25b)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, open, O) &\leftarrow org(\langle other \rangle, S), \\ &\quad cctx(C, S, open, O), foc(\langle this \rangle, C), \\ &\quad uallow(\langle this \rangle, su, S, open, O). \end{aligned} \quad (5.25c)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, exec, O) &\leftarrow org(\langle other \rangle, S), \\ &\quad cctx(C, S, exec, O), foc(\langle this \rangle, C), \\ &\quad uallow(\langle this \rangle, su, S, exec, O). \end{aligned} \quad (5.25d)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, close, O) &\leftarrow org(\langle other \rangle, S), \\ &\quad cctx(C, S, close, O), foc(\langle this \rangle, C), \\ &\quad uallow(\langle this \rangle, su, S, close, O). \end{aligned} \quad (5.25e)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, read, O) &\leftarrow org(\langle other \rangle, S), \\ &\quad cctx(C, S, read, O), foc(\langle this \rangle, C), \\ &\quad uallow(\langle this \rangle, su, S, read, O). \end{aligned} \quad (5.25f)$$

$$\begin{aligned} gallow(\langle this \rangle, su, S, write, O) &\leftarrow org(\langle other \rangle, S), \\ &\quad cctx(C, S, write, O), foc(\langle this \rangle, C), \\ &\quad uallow(\langle this \rangle, su, S, write, O). \end{aligned} \quad (5.25g)$$

This template promotes $\langle other \rangle$ subjects to be active while the user strives to achieve the goals of $\langle this \rangle$ social group. It is similar to the *Shared Focus with Other Subject* template, but with one difference. Here, the policy rules include an additional condition (*uallow*) which specifies that approval from the user is needed in the form of an asset policy.¹⁷ With this template, the security decisions are transferred to the user; if s/he issues appropriate asset policies, then these activities are allowed.

¹⁶Prior to being activated, the $\langle this \rangle$ subject was a passive object, and the activity of its activation is kept as a continuous activity.

¹⁷A template for this type of asset policy is defined in the next section.

5.4.2.1.5.3 Shared Subject with Other Participant (Outgoing Flow), with Relaxation This template is defined from perspective of $\langle this \rangle$ social group as the origin of the subject. It states that a subject which originates from a given social group, is allowed to perform a write activity while the association is a particular $\langle other \rangle$ social group, if this is allowed according to an asset policy specified for the particular subject. It consists of the following rules.

$$\begin{aligned} gallow(\langle this \rangle, sp, S, open, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, open, O), asc(\langle other \rangle, C), \\ & uallow(S, sp, S, open, O). \end{aligned} \quad (5.26a)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, close, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, write, O), asc(\langle other \rangle, C), \\ & uallow(S, sp, S, close, O). \end{aligned} \quad (5.26b)$$

$$\begin{aligned} gallow(\langle this \rangle, sp, S, write, O) \leftarrow & org(\langle this \rangle, S), \\ & cctx(C, S, write, O), asc(\langle other \rangle, C), \\ & uallow(S, sp, S, write, O), \\ & not(uallow(S, sp, S, read, O)). \end{aligned} \quad (5.26c)$$

This template allows subjects originating from a given social group to perform activities in which members of another social group participate. Since it applies only to write operations, it is essentially intended for activities involving channels via input sensors, such as the microphone, camera, touch, etc.¹⁸ This template promotes certain information from $\langle this \rangle$ social group to be shared with members of the $\langle other \rangle$ social group. Although such information flow poses threats to the confidentiality of this information, the risk of this consequence is acceptable, since these activities are regulated by the user. If the user considers certain information to be appropriate for sharing, s/he will issue an asset policy for this information, and these activities then will be allowed.¹⁹ This is assured by the last condition in the rules (using the *uallow* predicate).

5.4.2.2 Asset Policy

Several templates are defined here for asset policies controlling three possible interactions between assets during a single activity: subject-user, object-user and subject-participant interaction. In a similar way to group policy templates, these templates aim to mitigate the threats to the social group assets identified in Section 5.4.1. As expected, all of these aim to enable shared interaction between assets.

5.4.2.2.1 Subject-User Interaction Templates These templates define the conditions for allowing subject-user interactions involving a given asset related to a given social group $\langle this \rangle$.

5.4.2.2.1.1 Shared Subject with Other Focus This template is defined for a particular subject as a given asset and from the perspective of $\langle this \rangle$ social group as the origin of this subject. It states that a given subject $\langle asset \rangle$ which originates from $\langle this \rangle$

¹⁸Any communication via network channels would also require a read operation to be possible for the members of the $\langle other \rangle$ social group.

¹⁹A template for this type of asset policy is defined in the next section.

social group is allowed to perform an activity while the focus is on a particular $\langle other \rangle$ social group. This template corresponds to the *Shared Subject with Other Focus, with Relaxation* template.

$$\begin{aligned} uallow(\langle asset \rangle, su, S, create, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, create, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27a)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, destroy, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, destroy, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27b)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, open, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, open, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27c)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, exec, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, exec, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27d)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, close, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, close, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27e)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, read, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, read, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27f)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, write, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, write, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27g)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, exec, O) \leftarrow &= (\langle asset \rangle, O), org(\langle this \rangle, O), \\ &cctx(C, S, exec, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27h)$$

$$\begin{aligned} uallow(\langle asset \rangle, su, S, close, O) \leftarrow &= (\langle asset \rangle, O), org(\langle this \rangle, O), \\ &cctx(C, S, close, O), foc(\langle other \rangle, C). \end{aligned} \quad (5.27i)$$

5.4.2.2.1.2 Shared Focus with Other Subject This template is defined for the user as a given asset and from the perspective of $\langle this \rangle$ social group as the focus. It states that a given subject $\langle asset_x \rangle$ which originates from a particular $\langle other \rangle$ social group is allowed to perform an activity while the focus of this activity, and thus the focus of the user, is $\langle this \rangle$ social group. For clarity, the user in the first argument of the *uallow* predicate is represented by the social group in focus ($\langle this \rangle$). This template corresponds to the *Shared Focus with other Subject, with relaxation* template.

$$\begin{aligned} uallow(\langle this \rangle, su, S, create, O) \leftarrow &= (\langle asset_x \rangle, S), org(\langle other \rangle, S), \\ &cctx(C, S, create, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28a)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, destroy, O) \leftarrow &= (\langle asset_x \rangle, S), org(\langle other \rangle, S), \\ &cctx(C, S, destroy, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28b)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, open, O) \leftarrow &= (\langle asset_x \rangle, S), org(\langle other \rangle, S), \\ &cctx(C, S, open, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28c)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, exec, O) \leftarrow &= (\langle asset_x \rangle, S), org(\langle other \rangle, S), \\ &cctx(C, S, exec, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28d)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, close, O) \leftarrow &= (\langle asset_x \rangle, S), org(\langle other \rangle, S), \\ &cctx(C, S, close, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28e)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, read, O) \leftarrow &= (\langle asset_x \rangle, S), org(\langle other \rangle, S), \\ &cctx(C, S, read, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28f)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, write, O) \leftarrow &= (\langle asset_x \rangle, S), org(\langle other \rangle, S), \\ &cctx(C, S, write, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28g)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, exec, O) \leftarrow &= (\langle asset_x \rangle, O), org(\langle other \rangle, O), \\ &cctx(C, S, exec, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28h)$$

$$\begin{aligned} uallow(\langle this \rangle, su, S, close, O) \leftarrow &= (\langle asset_x \rangle, O), org(\langle other \rangle, O), \\ &cctx(C, S, close, O), foc(\langle this \rangle, C). \end{aligned} \quad (5.28i)$$

5.4.2.2.2 Subject-Participant Interaction Templates These templates define the conditions for allowing subject-participant interactions involving a given asset related with a given social group $\langle this \rangle$.

5.4.2.2.2.1 Shared Subject with Other Participant (Outgoing Flow) This template is defined for a particular subject as a given asset and from the perspective of $\langle this \rangle$ social group as the origin of this subject. It states that the given subject $\langle asset \rangle$ which originates from $\langle this \rangle$ social group is allowed to perform an activity while the association is a particular $\langle other \rangle$ social group. This template corresponds to the *Shared Subject with other Participant (Outgoing Flow), with relaxation* template.

$$\begin{aligned} uallow(\langle asset \rangle, sp, S, open, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, open, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.29a)$$

$$\begin{aligned} uallow(\langle asset \rangle, sp, S, close, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, close, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.29b)$$

$$\begin{aligned} uallow(\langle asset \rangle, sp, S, write, O) \leftarrow &= (\langle asset \rangle, S), org(\langle this \rangle, S), \\ &cctx(C, S, write, O), asc(\langle other \rangle, C). \end{aligned} \quad (5.29c)$$

5.4.2.3 Global Policy

A global policy is defined here which is applicable to any ubiquitous social system. For each interaction during an activity, it combines the group policies specified by each social group concerned, of which an asset is involved in an interaction. In particular, it combines the following group policies: (i) the group policy for all interactions, as specified by the social group of the subject's origin; (ii) the group policy for subject-object interaction, as specified by the social group of the object's origin; (iii) the group policy for subject-user interaction, as specified by the social group in the focus of the activity; (iv) the group policy for subject-participant interaction, as specified by the social group in the association of the activity; and (v) the group policy for subject-observer interaction, as specified by the social group in the setting of the activity. In the case where the association and/or setting are the empty social group, then group policies for subject-participant and/or subject-observer interactions respectively are not taken into account. The global policy consists of

the following rules.

$$\begin{aligned}
allow(S, OP, O) \leftarrow & org(G1, S), org(G2, O), \\
& cctx(C, S, OP, O), \\
& foc(G3, C), asc(G4, C), set(G5, C), \\
& not(asc(empty, C)), not(set(empty, C)), \\
& gallow(G1, so, S, OP, O), gallow(G2, so, S, OP, O), \\
& gallow(G1, su, S, OP, O), gallow(G3, su, S, OP, O), \\
& gallow(G1, sp, S, OP, O), gallow(G4, sp, S, OP, O), \\
& gallow(G1, sb, S, OP, O), gallow(G5, sb, S, OP, O).
\end{aligned} \tag{5.30a}$$

$$\begin{aligned}
allow(S, OP, O) \leftarrow & org(G1, S), org(G2, O), \\
& cctx(C, S, OP, O), \\
& foc(G3, C), set(G5, C), \\
& asc(empty, C), \\
& gallow(G1, so, S, OP, O), gallow(G2, so, S, OP, O), \\
& gallow(G1, su, S, OP, O), gallow(G3, su, S, OP, O), \\
& gallow(G1, sb, S, OP, O), gallow(G5, sb, S, OP, O).
\end{aligned} \tag{5.31a}$$

$$\begin{aligned}
allow(S, OP, O) \leftarrow & org(G1, S), org(G2, O), \\
& cctx(C, S, OP, O), \\
& foc(G3, C), asc(G4, C), \\
& set(empty, C), \\
& gallow(G1, so, S, OP, O), gallow(G2, so, S, OP, O), \\
& gallow(G1, su, S, OP, O), gallow(G3, su, S, OP, O), \\
& gallow(G1, sp, S, OP, O), gallow(G4, sp, S, OP, O).
\end{aligned} \tag{5.32a}$$

$$\begin{aligned}
allow(S, OP, O) \leftarrow & org(G1, S), org(G2, O), \\
& cctx(C, S, OP, O), \\
& foc(G3, C), \\
& asc(empty, C), set(empty, C), \\
& gallow(G1, so, S, OP, O), gallow(G2, so, S, OP, O), \\
& gallow(G1, su, S, OP, O), gallow(G3, su, S, OP, O).
\end{aligned} \tag{5.33a}$$

As can be seen, the global policy in fact implements the basic implicit policy rule.

5.4.3 Application of the Model and Language within Use Cases

In this section, the proposed policy model and language are used to analyze the use cases from Section 3.4. In order to facilitate the readability of the thesis, only Use Case 1 is

analyzed here; the remainder of the use cases are analyzed in Appendix B. In this analysis, the applicable security policies are specified and the way in which security decisions are made is described in detail. In order to give a representative example, security decision which repeat or are similar to the ones already described are not described in further detail here; Appendix C lists all the security decisions for each of the activity requests. It should be noted that this analysis is complementary to the one given in Section 4.4.1, and is based on the same assumptions.

The applicable security policies are specified in the use cases for each social group. However, for better readability, the templates from which these policies can be specified are represented, rather than complete security policies.

For the work and home social groups different security policy are specified in each use case. This is reasonable, as the use cases are designed so that each of these social groups have different requirements. It can be expected that these policies will be specified by particular members of the social group. For example, the work policy will be specified by the IT department within the company, whereas the home policy will be specified by one of its members after consultation with the others.

For the public social group different security policy are specified for each use case. Although this might seem unusual, the reason behind it is to present and evaluate the various templates defined earlier. The aim is to generate a single public security policy that can be installed in each mobile device, which will probably be specified by the designers of the context-aware security system which utilizes the proposed policy model and language (CASS).

For system social group a single policy is specified for each use case. This policy will be installed in each mobile device together with the CASS. Table 5.4 lists the templates from which the system policy is composed.

Table 5.4: System social group policy.

Interaction	Template
Subject-Object	Closed subject and object Shared subject with work channel Shared subject with home channel Shared subject with public channel
Subject-User	Closed subject and focus Shared subject with work focus Shared subject with home focus Shared subject with public focus
Subject-Participant	Closed subject and participant Shared subject with work participant Shared subject with home participant Shared subject with public participant
Subject-Observer	Open subject and observer

5.4.3.1 Use Case 1

Tables 5.5, 5.6 and 5.7 list the interaction templates from which the work, home and public social group policies are composed, respectively.

Scene 1. Alice is in her office and starts the *Office_W* app in order to prepare a report. This is achieved by the *Launcher_W* app through performing several activities. Firstly, it opens a channel to the screen, $a_1 = (Launcher_W, open, scr_{W1})$, and sends the

Table 5.5: Work social group policy.

Interaction	Template
Subject-Object	Closed subject and object Shared channel with system subject
Subject-User	Closed subject and focus Shared focus with system subject
Subject-Participant	Closed subject and participant Shared participant with system subject
Subject-Observer	Closed subject and observer Shared observer with system subject

Table 5.6: Home social group policy.

Interaction	Template
Subject-Object	Closed subject and object Shared channel with system subject
Subject-User	Closed subject and focus Shared focus with system subject
Subject-Participant	Closed subject and participant Shared participant with system subject Shared subject with public participant (incoming flow)
Subject-Observer	Open subject and observer

list of available apps to it, $a_2 = (Launcher_W, write, scr_{W1})$. After this, it opens a channel to the touch sensor, $a_3 = (Launcher_W, open, tch_{W1})$, and receives information through this on the application that Alice wants to start, as she taps on the icon for this app, $a_4 = (Launcher_W, read, tch_{W1})$. Since Alice wants to start the $Office_W$ app, it executes this application, $a_5 = (Launcher_W, exec, Office_W)$. The $Launcher_W$ app then submits a request for each of these activities, and the CASS makes a security decision for each request. As expected, except for a_1 , all requests are submitted after the previous activity has been allowed and performed.

The decision-making regarding activity a_1 takes place as follows. Firstly, the CASS obtains the security context of this activity. This contains the focus, association and setting, which are all the work social group. Then, the CASS identifies the origin of the subject and object from the system state. Both origins are also the work social group, since both subject and object are created by a work application: the $Launcher_W$ app is created by the $Installer_W$ app, which is responsible for creating work applications, whereas the scr_{W1} channel was created earlier by the $Launcher_W$ app. Next, the CASS evaluates the explicit policy rules by verifying whether $allow(Launcher_W, open, scr_{W1})$ follows from security policies specified, with respect to the information obtained. According to the global policy, this translates to a verification of whether it follows from each applicable group policy that the particular interaction for which it is specified is allowed. For this decision, only group policies specified by the work social group are applicable, since all assets involved in a_1 are related to this social group. Since work group policies are based on the closed template for each of the interactions, all interactions during a_1 are allowed according to these rules: (i) subject-object (SO) interaction is allowed because the subject and the object originate from the same social group; (ii) subject-user (SU) interaction is allowed because the subject's origin is same as the focus of the activity and the user; (iii) subject-participant

Table 5.7: Public social group policy.

Interaction	Template
Subject-Object	Closed subject and object Shared channel with system subject
Subject-User	Closed subject and focus Shared focus with system subject
Subject-Participant	Open subject and participant
Subject-Observer	Open subject and observer

(SP) interaction is allowed because the subject's origin is same as the association of the activity; (iv) subject-observer (SB) interaction is allowed because the subject's origin is same as the setting of the activity. As a result, the $allow(Launcher_W, open, scr_{W1})$ predicate is evaluated as true. Finally, the CASS uses the implicit policy rules to interpret the outcome from the evaluation. In accordance with the basic policy rule, the CASS takes a security decision that a_1 is allowed. Following this, the $Launcher_W$ app opens a channel scr_{W1} to the screen. After a_1 is performed, it is added to the set of continuous activities, A_C , and its security context is added to the set of current contexts, CC .

Activity a_2 is performed by the same subject and on the same object, but consists of a different operation. In addition, its focus, association and setting are the same social group as for a_1 . For these reasons, the group policies specified by the work social group are also the only ones applicable to this decision. The CASS allows this activity to be performed in the same way as a_1 . After this, the $Launcher_W$ app writes the list of available apps on the screen to Alice, via the channel scr_{W1} .

Activities a_3 and a_4 are similar to a_1 and a_2 and differ only in the object of the activity, tch_{W1} . Since this object is also a sensor channel, the security context of a_3 and a_4 is same as for a_1 and a_2 ²⁰. Since tch_{W1} was also created earlier by the $Launcher_W$ app, in the same way as scr_{W1} , this means that its origin is also the work social group. As a result, group policies specified by work social group are also the only applicable ones for this decision. Thus, the CASS makes the same security definition for a_2 as for a_1 , that is, the $Launcher_W$ app is allowed to open and read the tch_{W1} channel. In the same way as a_1 , after a_3 is performed, it is added to the set of continuous activities, A_C , and its security context is added to in the set of current contexts, CC .

The CASS makes decisions regarding activity a_3 in a similar manner. Since this activity takes place within the host, its association and setting are empty, and its focus remains the work social group. The object in this activity is the $Office_W$ app, which also originates from the work social group, since it is created by the same application as the $Launcher_W$. Thus, the group policies specified by the work social group are also the only ones applicable to each interaction. As a result, the CASS allows this activity. After a_3 is performed, it is added to the set of continuous activities, A_C , and its security context is added to the set of current contexts, CC . At the same time, the $Office_W$ app is added to the set of subjects, S , since it is now capable of performing activities in the future.

Activities a_1 , a_2 , a_3 and a_4 are accompanied by four additional activities. As explained above, certain system applications enable other, user installed, applications to use the host's hardware in a convenient manner. These additional activities describe the operation of these system applications. Activities a_1 and a_2 are accompanied by activities of a certain app $ScreenDriver$ opening and reading the screen channel, $a_{1a} = (ScreenDriver, open, scr_{W1})$ and $a_{2a} = (ScreenDriver, read, scr_{W1})$, whereas activities

²⁰More precisely, security contexts are same in each element, except for the activity element.

a_3 and a_4 are accompanied by activities of a certain app *TouchDriver* opening and writing to the touch channel, $a_{3a} = (TouchDriver, open, tch_{W1})$ and $a_{4a} = (TouchDriver, write, tch_{W1})$. All these activities have the same security context as activities a_1 to a_4 , but are performed by system applications. Thus, the security policy specified by the system social group is also taken into account during security decisions regarding these activities, in addition to those specified by the work social group. All interactions during these activities are allowed according to these policies, since the work and system social groups use compatible policy templates to enable such shared interaction. In particular, (i) subject-object (SO) interaction is allowed because the work group uses the *Shared channel with system subject* and the system group uses the *Shared subject with work channel* interaction templates; (ii) subject-user (SU) interaction is allowed because the work group uses the *Shared focus with system subject* and the system group uses the *Shared subject with system focus* interaction templates; (iii) subject-participant (SP) interaction is allowed because the work group uses the *Shared participant with system subject* and the system group uses the *Shared subject with system participant* interaction templates; (iv) subject-observer (SB) interaction is allowed because the work group uses the *Shared observer with system subject* and the system group uses the *Open subject and observer* interaction templates. As a result, CASS allows all activities a_{1a} to a_{4a} .

Once started, the *Office_W* app submits requests for creating channels via the screen and the touch sensors, in order to communicate with Alice. The channels are not yet part of the set of objects, since they are not yet created. Thus, these activities are allowed by default in the model. However, after their performance, the origin of the newly created objects is set to be the work social group, which is inherited from the *Office_W* app.

The subsequent activities can be analyzed in the same way as the ones above. These are the activities performed by the *Office_W* app during the preparation of the report, which include reading from the touch channel, writing to the screen, and opening, reading and writing to the data object, doc_{W1} . The work group policies are also the only applicable ones for these decisions, and these activities are all allowed according to these policies.

The decision-making for the activity of opening to the network channel, $a_5 = (Office_W, open, ch_{W7})$, is carried out as follows. The CASS obtains the following information necessary for the decision. The focus of a_5 is the work social group, and its association and setting are the public social group. The origin of the newly created network channel ch_{W7} is also the work social group, which is inherited from the *Office_W* app. Thus, in addition to the group policies specified by the work social group, group policies specified by the public social group are also taken into account in this decision, in particular, for subject-participant and subject-observer interaction. These public policies are based on the open templates and allow any interaction. The evaluation is carried out as follows: (i) subject-object (SO) interaction is allowed because the subject and the object originate from the same social group; (ii) subject-user (SU) interaction is allowed because the subject's origin is same as the focus of the activity and the user; (iii) although subject-participant (SP) interaction is allowed from the perspective of the public social group, since this allows any interaction, this interaction is not allowed from perspective of the work social group, as the association is different than the subject's origin. (iv) although subject-observer (SB) interaction is allowed from the perspective of the public social group, since it allows any interaction, this interaction is not allowed from the perspective of the work social group, since the setting is different than the subject's origin. As a result, the $allow(Office_W, open, ch_{W7})$ predicate is evaluated as false according to the global policy. The CASS interprets this outcome according to the basic implicit rule into security decision that the activity a_5 is denied. Following this, the *Office_W* app is not allowed to open the ch_{W7} channel to the remote server. Therefore, the *Office_W* app is not allowed to

open a channel to a remote server and to communicate through it.

In the last activity in this scene, the *Office_W* app tries to open another document *doc₂*, $a_6 = (Office_W, doc_{H1})$, which originates from the home social group. The focus of this activity remains the same as before, while the association and setting are empty. For this activity, the CASS takes also the group policy for the subject-object interaction specified by the home social group, in addition to the group policies specified by the work social group. This home policy is based on the closed template, in the same way as the one from the work social group. According to these two policies, subject-object interaction is not allowed during this activity, from both perspectives, since the subject and object originate from different social group. Thus, the CASS evaluates the predicate $allow(Office_W, open, doc_{H1})$ as false according to the global policy, and interprets this as a security decision that the a_6 is denied. This prevents the *Office_W* from opening the data.

Scene 2. In this scene, Alice is taking a coffee break with her colleague Adam in the company's kitchenette. During the entire scene Alice performs activities using only applications which originate from the work social group as the subjects; these operate only with objects that originate from the same social group. The focus for all of these activities is her work social group, their association and setting are either the work social group or they are empty. As a result, the group policies specified from the work social group are the only ones applicable to the decisions on all these activities. According to these policies, which are based on the closed templates, all interactions during these activities are allowed to happen. Thus, the CASS allows all of them.

Scene 3. Alice is in her office and wants to call home. In order to be able to start the appropriate application for this, she uses the *Launcher* to open the *Launcher_H* app, which is responsible for starting home applications. The CASS recognizes this request for an activity as a change in Alice's focus, from the work to the home social group, and reacts accordingly as follows.

The CASS makes a new security decision for each continuous activity with respect to its security context, which is kept in the set of current contexts. The set of continuous activities consists of the opening activities of all active application objects, i.e., subjects, as well as, the channel and data objects they use. One can imagine that, at this moment, this set contains the opening activities of the various OS driver applications and the *Launcher* app, the *Office_W* app, and the screen and touch channels they use for communication with the user. All these objects are related to the system social group, except for the *Office_W* app and the channels it has opened, which are related to the work social group. Since Alice's focus is the home social group, the group policy for subject-user interaction specified by this social group is taken into account for each decision. For the other interactions, group policies specified by the system or work social group are taken into account, depending on the origin of the subject and object²¹. Subject-object, subject-participant and subject-observer interactions are all allowed for these activities, in the same way as when they were requested²², since these interactions are not affected by the focus change. Subject-user interaction is not allowed for activities related to work applications, since both work and home social groups use the closed template in their group policies for this interaction. This interaction is allowed only for activities related to system applications, since the home social group also uses the *Shared focus with system subject* template and the system group uses the *Shared subject with home focus* template. Thus, only activities related to system applications are allowed according to the explicit policy rules. After interpreting this

²¹Since the association and setting for these activities is empty, no group policy is taken into account from the perspective of the participant and observer during these decisions.

²²Since all these activities are continuous, they have been requested and allowed at some point in the past.

outcome, the CASS generates security decisions that all work objects need to be closed, according to the contextual and termination implicit rules.

Alice is able to operate the *Launcher* app, since this is a system application, and the *Launcher* is then able to open the *Launcher_H* app, according to the defined exceptions for it. However, *Launcher_H* is not able to use the sensor channels (e.g., via screen and touch sensors) in a work setting, because this is not allowed according to the group policy for subject-observer interaction, as specified by the work social group. This policy is based on the closed template and allows only work subjects to operate in this work setting. As a result, Alice is not able to use the *Launcher_H*, or any other home or public application while at work. She has no other choice, but to open the *Launcher_W* app, which brings her focus back to the work social group.

Alice decides to leave the company premises in order to call home using her mobile device. All activities using the *NFCManager_W* app on her way out of the building are allowed according to the group policies specified by the work social group, as these are the only ones that are applicable to these decisions. After Alice leaves the building, her focus changes from work to public, which makes the CASS react accordingly and close all open work objects. This is similar to the above, since the public social group uses the same templates for subject-user interaction in her group policy.

As soon as Alice is outside, she starts the *Launcher_H* app. As expected, she is now able to operate home apps via screen and touch channels, since the setting for these activities is no longer the work social group, but is now the public group. Both public and home social groups allow subject-observer interaction since they use the open template for this interaction in their group policy. All other interactions are also allowed according to the home social group policies, which are the only ones applicable to them.

Alice starts the *Voip_H* app and tries to call her daughter. As explained earlier (Section 4.4), this conversation takes places as a result of four activities: read and write activities via a network channel to her daughter's device, a read activity via a microphone channel, and a write activity via a speaker channel. The CASS makes decisions for these activities as follows. It allows subject-object and subject-user interactions during all of these, since home, which is the only social group concerned, uses the closed templates for these interactions in its group policies. Furthermore, the activities via the network channel have the home social group as the association and the public social group as the setting. Since home social group uses the closed template for subject-participant interaction and the open template for the subject-observer interaction in its policies, the CASS allows both of these interactions. Thus, it allows both of these activities to be performed. According to these templates, the subject-participant and subject-observer interactions during the activity via the speaker are also allowed, as this has the home social group for both association and setting. Hence, the CASS also allows this activity to be performed. Finally, the activity via the microphone has the public social group as both association and setting. While subject-observer interaction during this activity is allowed according to the same template as for the other activities, other templates are used for evaluating the subject-participants interaction. The home social group also uses the *Shared subject with public participant (incoming flow)* which allows its subjects to read via channels in the case where the association is the public social group. On the other hand, the public social group uses the open template for this interaction, which allows any subject to operate via channels while its members are participants. Since the CASS allows both of these interactions during this activity, this activity is also allowed to be performed. As a result, Alice is able to make the call to her daughter.

Scene 4. Alice is on her way home and her focus is her public social group. She operates with the *Assistant_P* app, which also originates from the public social group. The

association and setting of these activities, which take place via a screen and touch channels, are both the public social group. Hence, during these activities only interactions between assets related to the public social group take place; these are allowed according to its group policies. According to these policies the CASS also allows communication via the network channel to the remote server, as the association and setting of these activities are also the public social group. However, it does not allow the activity of the *Assistant_P* app reading a work document, since the work social group does not allow subject-object interaction according to its group policy, which is based on the closed interaction template. In addition, it does not allow the *Assistant_P* app to operate in a work setting, according to the closed template used in the work group policy for the subject-observer interaction. Thus, Alice cannot use this app while at work.

When Alice is in front of her home she uses the *Launcher* to start the *Launcher_H* app, which changes her focus from public to home. Following this change, the CASS makes a new security decision for each continuous activity with respect to the new security context. Since the home social group uses the closed template for subject-user interaction, the CASS closes all public applications and leaves only the system applications.

The CASS also allows all other activities which Alice performs with applications originating from the home social group. However, it does not allow Alice to operate the *Office_W* app, which is intended for work purposes only. This is because the work social group uses the closed template for the subject-observer interaction, which allows Alice to operate work applications only in a work setting. This is similar to Alice being unable to operate *Launcher_H* at work, but from the opposite perspective.

Chapter 6

Prototype

In this chapter, a prototype of a context-aware security system is described which is applicable to ubiquitous social systems. This prototype enables the specification and enforcement of security policies and is intended for mobile devices operating on Android. The prototype is based on the proposed conceptual model of security context defined in Chapter 4 and the security policy model and language defined in Chapter 5. Firstly, several works related to the prototype are described; following this, a description of the architecture of the prototype and its implementation is given. Finally, an evaluation of the prototype is presented.

6.1 Related Work

In order to provide sufficient background information for the prototype, a description is given in this section of the characteristics of the Android operating system and its security.

6.1.1 Android

Android is a mobile operating system developed by the Open Handset Alliance consortium, which is led by Google [104]. It is designed for mobile devices, primarily smartphones and tablets. However, it can be also found on a number of other devices, for example: notebooks, game consoles, digital cameras, televisions, cars, wrist watches, etc. This makes it one of the most installed operating systems. Because its source code is released under open source licenses¹, Android is popular among technology companies and enthusiasts that require functional, low-cost and customizable operating system. As a result, it has gained a large community of developers during the past years, which continuously extend it with new features. Many of the community developed extensions are later merged with the official release of Android.

Android's software stack is composed of three layers: kernel, middleware and application layer (Fig. 6.1). Kernel layer is located at the bottom of the software stack and is responsible for communication with the device's hardware. It provides core functionalities for memory management, process scheduling, device drivers, file system and networking. It is based on the Linux kernel, over which it adds some additional drivers to provide the functionalities needed for a mobile device OS. One of the most notable additions is the Binder, which is responsible for handling communication between applications and their components.

Furthermore, middleware layer provides interfaces for kernel's functionalities and it thus located above the kernel layer. It consists of three elements: a set of native libraries,

¹Nevertheless, most vendors ship their devices with additional proprietary software added to Android.

a runtime environment and an application framework. Native libraries provide certain basic facilities in the OS, such as those for graphical processing, secure communication, database management, etc. They are written in C/C++. Runtime environment enables the execution of applications. It contains core Java libraries and a virtual machine, which is tailored specifically for resource constrained mobile devices. Application framework provides essential services for operation and development of applications. As examples of such services consider those concerned with management of windows, location, audio, telephony, wireless communication, packages, etc.

Finally, application layer is located on the top of the software stack and consists of the various user-installed applications. Naturally, users install applications in order to utilize the device resources through them. Android applications, shortly referred to as apps, are usually found on content delivery platforms, which act as stores or marketplaces where apps can be bought online. App stores are maintained by different providers, for example Google, Amazon or Samsung, which define their own policies about who can upload apps and how. Android apps are usually written in Java and are compiled into a special DEX bytecode, which executes on the virtual machine.



Figure 6.1: Software stack of Android OS.

Compared to other operating systems, Android provides substantially different application abstractions, which cannot be found in traditional desktop and server OSs. In particular, it provides four types of components for implementing an application's functionality: (i) activity, which comprises the application's user interface, (ii) service, which provides background processing and acts as daemon, (iii) broadcast receiver, which handles asynchronous messages between applications' components, and (iv) content provider, which acts like application's data server, that can be also queried by other applications.

Nevertheless, Android represents running applications as processes, similar as other operating systems.

Communications between user applications and between an application and device's internal components, such as sensors, storage and network components, occur at kernel and middleware layer. In both cases, device's internal components are not accessed directly, but through certain system applications, which encapsulate them and provide interfaces for operating with them. At kernel layer, such applications are the device drivers, whereas at middleware layer, such applications are called device services. The latter are introduced in Android as part of its application framework.

Communications at kernel layer go through the file system [105]. Running applications, i.e., processes, access files by reading or writing information contained in them, or they create new ones or delete the existing ones. The Linux kernel represents all internal components as files. Storage is represented through regular files, simply referred to as files, whereas other components are represented as special files, so called device files. In this perspective, a device driver acts as application that communicates with an internal component (hardware) and a device file, representing this component in the file system. An important type of special files are sockets, which represent end-points for communication, either between two applications running on the device or between an application and a remote party, through the device's network capabilities. All forms of communication at kernel layer are usually referred to as inter-process (IPC) communication and are provided by the Linux kernel as standard forms of communication.

On the other hand, communications at middleware layer go through application components. Activities, services, and broadcast receivers communicate between each other by passing special message objects called intents. An intent denotes an intention of an application component to communicate with another component, either from the same or different application. It includes two parts: a recipient, which declares the intended component for communication, and an information part, which is optional and specifies a remote procedure provided by the recipient, together with its associated arguments. For example, intents can be used to start an activity, stop a services, or broadcast information to a broadcast receivers. Communication through intents is a novel mechanism introduced in Android. On the other hand, communication with content providers occurs directly, without intents. Android labels each content provider with a unique resource identifier (URI). Other components communicate with content providers by simply addressing their URIs, together with the desired operation they want to perform over their content. For example, a service can insert, update, or delete information in a content provider. Analogous to IPC, all forms of communication in this layer are usually referred to as inter-component communication (ICC).

Android prefers ICC as a primary form of communication with the internal components. Applications access many of the sensor and network components by communicating with their device services. For example, an application accesses with the location sensor or the telephony component by exchanging intents with the location or the telephony manager, as their specific device services, respectively. However, some internal components are still accessed through the forms for IPC. For example, applications access the microphone and the camera and the Wi-Fi component, by directly using their device drivers.

6.1.2 Android Security

Android is based on the following security policy model. It uses the familiar abstractions of subject, object and rights, as in many other policy models. It refers to rights as 'per-

missions’, and this has become a widely used term in Android terminology.² The model considers applications as subjects, whereas it considers the device’s internal components, such as storage, sensors and network components, as objects. Since application components can also be accessed by other applications, they are also considered as objects. The model defines permissions for direct access to objects and for the administration of objects. Each permission refers to a particular type of object and to a particular type of access to these objects. For example, the *READ_CONTACTS* permission allows a subject to read the contact’s data, and the *WRITE_EXTERNAL_STORAGE* permission allows a subject to write to all data saved in the external storage. While Android offers a rich set of predefined permissions, it also enables application developers to define their own permissions. However, these permissions apply only to the control of access to components from their own applications, and do not apply to system-wide resources. In addition, Android uses labels in its policy models, which are assigned only to storage objects such as data files.

Android supports the following policy rules in its model. Explicit policy rules are represented as relations between subjects and permissions. Each subject is assigned a set of permissions during installation, which is static and cannot later be changed. This means that once given to a subject, permissions cannot be revoked, nor extended with new ones later. The permissions requested by a subject during installation are also obligatory, i.e. if they are not granted, the installation of the subject will fail. Furthermore, Android’s model identifies the following implicit policy rules. A subject can access an object only if it possesses the appropriate permission for the particular types of object and access. As may be expected, the permissions required for system-wide resources are identified by the model itself, whereas for application-specific resources they are identified by their developers. While this rule applies for all objects, the following rule applies only for storage objects (e.g. data files). Each object created by a subject is labeled as its own object. The subject’s own objects are considered private, and cannot be accessed by other subjects. If a subject wants to share an object with another subject, it needs to label it as public.

Android enforces these policy rules as follows. It uses several checks in the middleware layer to enforce the subject’s permissions. These checks are mainly placed in the Activity Manager Service, which is responsible for managing the ICC. When a subject tries to access a component from another application via the ICC, the Activity Manager Service checks whether or not this subject has the appropriate permission for the particular type of object and access. If the subject has this permission, it allows the access; otherwise, access is denied. All permissions granted to applications during installation are stored by the Package Monitor Service. Furthermore, Android uses the standard file system access control mechanisms from the Linux kernel [106] to achieve the isolation of objects owned by the different subjects. In particular, it assigns to each subject a unique user identifier (UID) during installation, and a private directory according to this UID. Subjects can use this directory for saving objects (files) which they do not want to share with other subjects, that is, objects they consider private. If a subject wants to share an object with other subjects, it needs to save it in a directory which is publicly accessible. Android also uses this UID to achieve isolation of subjects during runtime. It does this by executing each subject in its own instance of the virtual machine, which cannot be accessed by other subjects. These isolation mechanisms are usually referred to as application sandboxing.

Many security enhancements to Android have been proposed in the last years. From the perspective of functionality, they enhance Android in two primary directions. Firstly, they aim to enhance Android’s permission framework, which is generally considered to be coarse-grained. Apex [107] and CRePE [108] do this by adding flexibility to the permission-

²In this discussion, the term permission will also be used rather than right.

granting process. Apex enables users to deny some of the permissions requested by an application, rather than the default all-or-nothing approach. It also supports the granting of permission under certain conditions, which may involve straightforward contextual information (e.g. time of day, number of SMS etc.). CRePE also adds contextual conditions to permission granting. It handles richer and more diverse contextual information than Apex; however, these are still low-level permissions. In both approaches, the contextual conditions are checked at runtime and permissions are granted or denied accordingly. In contrast, Dr. Android and Mr. Hide [109] enhance the permission framework by increasing the granularity of the permissions. It partitions Android's predefined permissions into a set of more finely grained ones, which can be granted to applications separately. For example, it partitions the permission for Internet access into several domains depending on the destination address. Thus, an application must have Internet permission for the public domain in order to access public web pages. Aurasium [110] and AppGuard [111] also enable the specification of more fine-grained permissions, and allow the revoking of permissions. However, they are primarily concerned with how to enforce this with the minimum amount of modification to the Android operating system, and do not consider contextual information. In fact, these two approaches, together with [109], can be installed as a stand-alone application in which security policies are specified and evaluated. However, this requires each application to be modified and repacked before installation, in order for a special code for the enforcement of security policies to be added.

Furthermore, several approaches try to enhance Android's isolation mechanism. Android's application sandbox isolates only a single application, together with its private directory. These approaches aim to isolate many resources together, essentially achieving a "soft virtualization". For example, Cells [69] provides multiple environments, called virtual phones, on a single hardware phone. Each virtual phone is a standard Android environment, with all its functionalities and device components. Cells considers only one virtual phone to be in the foreground, that is, to be displayed at any time, and considers the other virtual phones to be running in the background, without being displayed. Each virtual phone is configured to have different access rights for different device components. For each device component, a virtual phone may have no access, shared access or exclusive access. Other approaches isolate only applications and data, combined in the form of compartments. TrustDroid [68] refers to these compartments simply as domains. It centers around three domains, depending on the level of trust in their resources: trusted, untrusted and system. While trusted and untrusted domains must be isolated from each other, the system domain must be accessible by the other domains in order to preserve the correct functionality of their applications. However, since it labels all resources from a certain domain, it is possible that many other domains may also be supported. TrustDroid further employs simple contextual information to filter network traffic, generated by applications from a particular domain. For example, untrusted applications should not operate on a corporate network. MOSES [67] refers to these compartments as security profiles. A security profile is composed of applications and data which are considered to be similar, such as for corporate and home purposes. It considers only one security profile as active at a given moment; which profile is active depends on the security context. In order to enable the automatic activation and deactivation of profiles as the security context changes, MOSES relates various contextual information to each security profile. It supports both low-level (e.g. time and location) and high-level contextual (e.g. reputation and trust level) information.

Generally, security enhancements to Android are implemented using a technique called hooking. In computer programming, hooking is used to alter or extend the behavior of an operating system or an application. It is achieved through the interception of function

calls, messages or events passed between software components, and their replacement with different ones, known as hooks [112]. As may be expected, hooks can be placed in each of Android's software stack layers. In the kernel layer, hooks can be placed in the file system and replace the system calls for operation with both regular (i.e. storage) and special files (e.g. sockets and device files). In the middleware layer, hooks can be placed in the Activity Manager Service, and these replace the APIs in which permission checks are implemented. Both kernel and middleware hooks are applied by physically modifying or replacing the system calls and APIs in Android's source code. In order for these hooks to be used in existing mobile devices, a modified version of Android must first be installed. Of the approaches described above, those in [67–69, 107, 108] utilize hooks at the kernel and/or middleware layer. On the other hand, in the application layer, hooks can be placed within each application. They replace the same system calls and APIs as the lower-layer hooks, but are introduced as new APIs which wrap the functionality of the system APIs. This requires that each call to the system version of an API within the application is replaced with a call to the new one. Thus, application layer hooks are applied by modifying the application's source code. Each application needs to be repacked together with the modified code, and this modified version is then installed on the mobile device. As a result, application hooks can be used on an existing mobile out of the box. Of the approaches described above, those in [109–111] utilize hooks at the application layer.

In addition, some approaches, such as SE Android [103] and Android Security Module [112], aim to modify Android using an access control framework containing all possible kernel and middleware hooks. Such a framework would enable other Android enhancements and new policy models defined specifically for Android to be implemented easily. However, prior to such solution becoming applicable, further analysis and discussion of the complete set of hooks is necessary. In addition, this type of solution is required to be accepted by Google as part of Android, in order to become widely available. A similar solution to this exists for Linux, in the Linux Security Module [113].

6.2 Description of the Prototype

In this section, a prototype of a context-aware security system applicable to ubiquitous social systems is presented. This prototype is based on the conceptual model of security context defined in Chapter 4 and the security policy model and language defined in Chapter 5. In particular, the architecture of the prototype and a possible implementation are discussed.

6.2.1 Architecture of the Prototype

The prototype for ubiquitous social systems is composed of five elements, as illustrated in Fig. 6.2.

- The *Policy Decision Point (PDP)* is responsible for making decisions regarding all activities performed in the ubiquitous social system. As such, it is the central element in the architecture. In addition, the PDP is also responsible for administration of the system state, according to the security policy model. This includes (i) updating the states of resources according to the object lifecycle (Fig. 5.3); (ii) assigning an origin to resources upon their creation; (iii) keeping track of continuous activities; and (iv) monitoring their security contexts and detecting changes.
- The *Policy Enforcement Point (PEP)* is responsible for enforcing the decisions made by the PDP and thus ensuring that subjects perform only allowed activities on ob-

jects. Generally, the PEP is considered as a dummy element, which either allows the subject to perform the requested activity or not. However, in this prototype, the PEP is also capable of performing certain activities on subjects and objects, in order to enforce reactions as security decisions. This additional functionality of PEP is coordinated by the PDP.

- The *Context Manager (CM)* is responsible for supplying the security context of all activities. It provides the following functionalities: context sensing, representation and reasoning.
- The *Policy Administration Point (PAP)* is responsible for the administration of the context-aware security system, by providing an interface for specification of security policies. Policy makers can use this interface to specify their desired policy rules, which are further combined in group policies and stored in the policy database. At the same time, the user can use this interface while specifying certain relaxations from group policies, which are stored as asset policies in the database. In both cases, the PAP enables browsing through the policy templates saved in the policy database, and through the system state, which makes instantiation of templates easier.

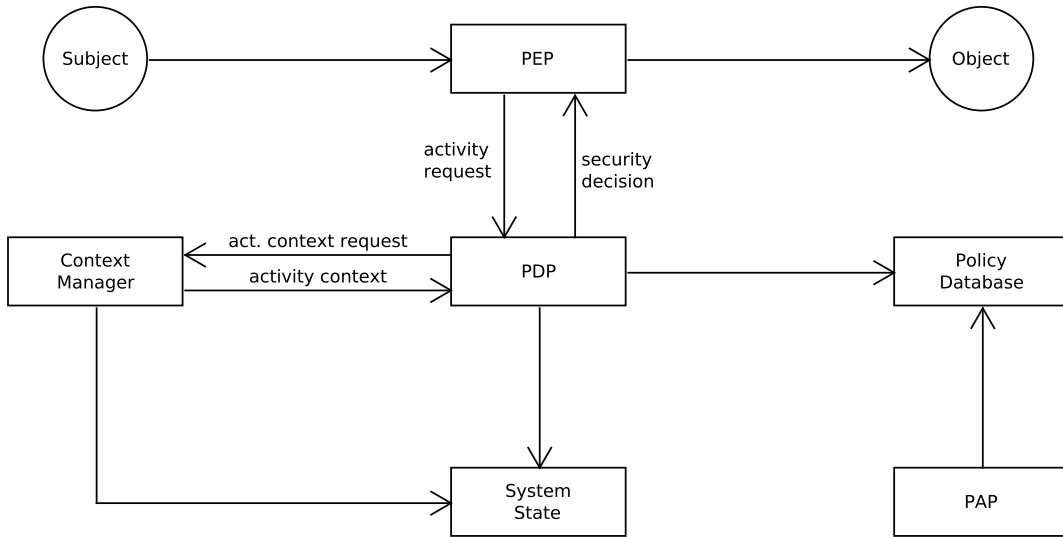


Figure 6.2: Prototype architecture.

These elements are rarely found together in the existing literature, even though PDP, PEP and PAP are regularly included in existing policy-based systems [59, 60], and the CM encompasses the functionalities regularly found in existing context-aware systems [57].

The proposed prototype operates as follows. The PEP monitors all objects and intercepts each attempt by a subject to carry out an operation on an object. For each attempt, it generates a request for an activity, $req(s, op, o)$ and sends this request to the PDP. An activity request contains information about the subject which initiates the activity, $s \in S$, the operation of the activity, $op \in OP$ and the object on which the activity will be performed. The PDP checks the validity of the received request; it obtains information for this purpose from the system state database. If a request is valid according to the resource lifecycle, it continues with making a security decision; otherwise, it sends a response to the PEP that the activity is denied, $deny(s, op, o)$.

Prior to making a security decision, the PDP prepares itself by gathering the necessary information from the neighbor elements. It first acquires the security context of the requested activity, $ctx(s, op, o)$, from the CM. It does this by sending a request for context, $ctxReq(s, op, o)$, which contains the activity for which the context is needed. The CM assesses the security context of the activity and sends it back to the PDP. For this purpose, it consults the system state database if any information is required. Next, the PDP obtains information about the subject and object of the requested activity from the system state. Finally, it retrieves the applicable security policies from the policy database. Except for global policies, security policies are not applicable to all decisions. A group policy is applicable if it is specified by a social group whose asset is involved in the requested activity. For example, if no asset related to the home social group is involved in an activity, then the home group policy is not be taken into account. Similarly, an asset policy is applicable if it is specified for a particular asset involved in the requested activity.

Once the preparation steps have been taken, the PDP makes a security decision. Following an activity request, the possible decisions are either that the activity is allowed, $allow(s, op, o)$, or denied, $deny(s, op, o)$. The PDP sends the decisions to the PEP immediately after making them. The PEP enforces these decisions as follows. In the case of $allow(s, op, o)$, the PEP allows the subject s to carry out the desired operation op over the object o . On the other hand, in the case of $deny(s, op, o)$, PEP simply drops the intercepted attempt from the subject and no activity is performed. However, in both cases the PEP notifies the PDP about the summary of the enforced decision, so that PDP can update the system state if necessary.

For a continuous activity, the PDP must assure that a security decision $allow(s, op, o)$, made following a request, holds throughout its performance. One reason why such a decision may no longer hold is a change in the security context of the activity, $ctx(s, op, o)$, becoming inappropriate according to the policies. In view of this, the PDP monitors the security context of the continuous activities which are currently being performed, detects changes in them and initiates another security decision following each change. When a context change occurs, the PDP acts as if the continuous activity has been requested again, that is, it prepares to make a security decision in the same way as above. However, in this case, the possible decisions are either that the activity is allowed, $allow(s, op, o)$, or that it needs to be terminated, $terminate(s, op, o)$. The PDP sends only the latter decision to the PEP; the former does not require enforcement, since the activity (s, op, o) is already under way. The $terminate(s, op, o)$ instruction informs the PEP that a reaction to a particular object o needs to be performed; however, it does not specify what this reaction should be. According to the proposed model, only one reaction is applicable, that is, closing the object. The PEP enforces this activity by performing the close operation on o .

In the rest of this section, the PDP and PEP are described in more detail, as these two elements are specific to the proposed approach and prototype architecture. Additional details are not provided for the other elements, as these tend to be generic and have similar characteristics to many other context-aware or policy-based systems. With respect to the CM, the reader is referred to [114], which contains a survey of the ways in which context management is handled in existing context-aware security systems, and to [36, 61–63], which offer surveys of context management approaches in context-aware systems in general.

6.2.1.1 The Policy Decision Point (PDP)

The PDP performs four operations: validation, evaluation, interpretation, and monitoring.

- *Validation:* During this operation, the PDP checks the validity of an activity request, according to the resource lifecycle defined in Section 5.2 and the current state of the object, contained in the system state. It examines whether the requested operation can be carried out on the object while in the current state. Open and destroy operations can be performed only on passive objects, while read, write, execute and close operations can only be carried out for active objects. Following this operation, the PDP determines whether or not the activity request is valid.
- *Evaluation:* During this operation, the PDP evaluates whether a requested activity is allowed with respect to a given set of security policy rules. In addition to an activity request, the PDP also receives the current security context of the requested activity as an input for this operation. At the core of this operation is the Evaluator component (Fig. 6.3), which is a reasoning engine that implements the evaluation of the policy rules. Since the proposed language for policy specification is a dialect of Datalog, the Evaluator is based on Datalog. Following this evaluation, the PDP determines whether or not a requested activity is allowed to be performed.
- *Interpretation:* During this operation, PDP interprets the output from the evaluation of the requested activity into a security decision. In the case where the activity is allowed to be performed, PDP generated $allow(s, op, o)$ as a decision. On the other hand, if the activity is not allowed, then the PDP consults the system state to check whether the activity is already running. If not, it generates $deny(s, op, o)$ as a decision; otherwise, it generates $terminate(s, op, o)$. In the latter case, this decision is followed by the activities which implement the effects of the $terminate(s, op, o)$ decision.
- *Monitoring:* During this operation, a special PDP component called Monitor (Fig. 6.3) monitors the security contexts of all activities which are currently being performed and detects changes in these. For a particular activity a , this is carried out as follows. The Monitor repeatedly obtains the current security context of the activity, $cctx'(a)$, from the CM, at specific time intervals. It then compares the values of the elements in $cctx'(a)$ with the ones saved for this particular activity in the current context component of the system state, CC , i.e. $cctx(a)$. In the case where any of these values differ, it generates an event representing a change in the current security context of activity a , e.g. $\Delta cctx(a)$, and sends this event to the PDP as an input. At the same time, it updates $cctx(a)$ with $cctx'(a)$ in the system state, as a result of which the elements in CC always have current values. In the case where $cctx'(a) = cctx(a)$, the Monitor does nothing.

The PDP makes a security decision following two events. The first is an activity request, and this event is generated by the PEP. The PDP first performs a validation operation on the received activity request. In the case where this is valid, the PDP continues by requesting the current security context of the requested activity from the Monitor component. The Monitor re-sends the security context request to the Context Manager, which computes the security context of the activity and responds. The Monitor then transfers the response to the PDP, which combines it with the activity request and sends it to the Evaluator component. The Evaluator evaluates the applicable security policies, given the system state, the received request and the security context, and generates a response for the PDP as to whether or not the activity is allowed to be performed. Next, the PDP performs an interpretation operation on the evaluation outcome, and generates a security decision. In the case of an invalid activity request, the PDP directly generates a security

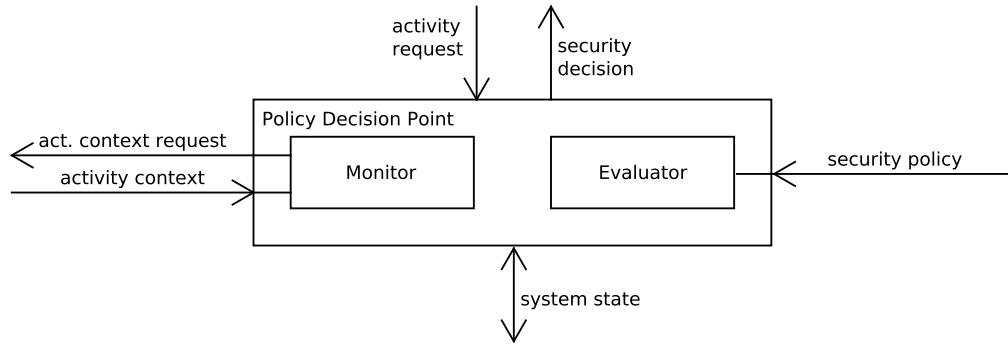


Figure 6.3: Components of the Policy Decision Point.

decision that the requested activity is denied. Fig. 6.4 illustrates the flow of the decision-making process in the PDP during an activity request. Each security decision made during this process needs to be enforced.

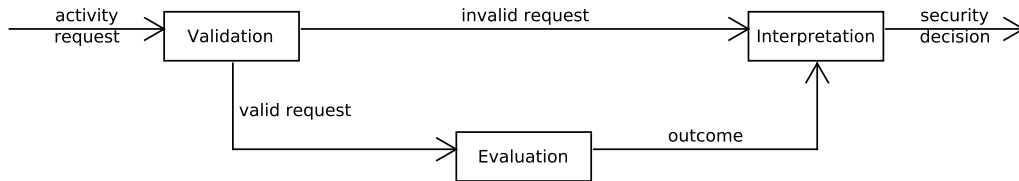


Figure 6.4: Flow of the decision making process during activity request.

The second event is a change in the security context of a continuous activity, and this event is generated by the Monitor. The PDP acts upon this event as if the activity for which the security context has changed has been requested again. A new security decision for this activity is initiated, although this time without the validation step. This is because this step has already been performed during the first decision for this activity, when the activity was initially requested. In addition, the PDP omits the retrieval of the security context, as it already has this information; this was sent by the Monitor together with the context change event. The PDP initiates this security decision process by directly requesting the Evaluator to evaluate the applicable security policies for this activity. The Evaluator acts in the same way as before, regardless of whether the activity has been requested again. Next, the PDP interprets the evaluation outcome it has received, and returns a security decision. During its interpretation, the PDP is aware of whether a security decision has been triggered following an activity request or a security context change. Fig. 6.5 illustrates the flow of the decision-making process in PDP during a context change. Only the *terminate*(*s, op, o*) decision needs to be enforced during this operation; *allow*(*s, op, o*) is already enforced, as the activity is already taking place.

6.2.1.2 The Policy Enforcement Point (PEP)

The PEP performs two operations: interception and enforcement.

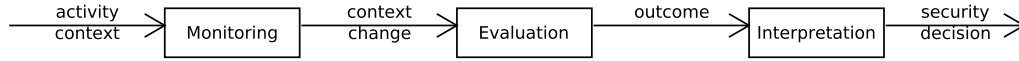


Figure 6.5: Flow of the decision making process following a context change.

- *Interception:* During this operation, the PEP monitors all objects and intercepts any request for an activity on any of them. A request for an activity is an attempt by a subject to initiate an operation on an object. This operation does not allow an activity to be performed until a security decision is made on this.
- *Enforcement:* During this operation, the PEP enforces the security decisions. In the case where an activity is allowed to be performed, the PEP allows the subject to carry out the requested operation on the object; otherwise, in the case where an activity is denied, PEP simply drops the attempt initiated by the subject. Finally, following a $terminate(s, op, o)$ decision, the PEP terminates an activity by closing the object o . The prototype acts as a subject in these activities. This behavior is considered here to be a basic form of adaptation.

The PEP operates as follows. It monitors each object and intercepts any attempt by a subject to initiate an operation on an object. Following each such attempt, it generates a request for an activity with the received information, sends the generated request to the PDP and awaits a security decision. When the decision arrives, it is sent for enforcement. The PEP reports the successful enforcement of a decision to the PDP, which further updates the system state database.

6.2.2 Implementation of the Prototype

Based on the architecture described above, a proof-of-concept of the prototype was implemented. This proof-of-concept was intended as a simulation of the prototype's operation, and was developed solely for the purpose of evaluating its feasibility and the performance of its pdecision-making process. A more 'realistic' implementation would require the appropriate incorporation of the Policy Enforcement Point within Android, in order for security decisions to be enforced. Firstly, the technical details of the proof-of-concept are described below. Following this, some valuable insights are offered into the ways in which this prototype can be implemented within Android. Finally, the policy evaluation in Datalog is discussed in more detail.

6.2.2.1 Technical Details

The prototype was implemented using the Python programming language. Python was chosen due to its simplicity, flexibility and code readability, which make software development fast and straightforward. At the same time, Python is a widely used and popular general-purpose language, especially within scientific communities, with a large standard library and many open-source libraries available for its extension.

The programming concepts used for the implementation of the proof-of-concept are given in Fig. 6.6. Class objects were defined for each element of the prototype, except for the PAP; this was not implemented since the usage and instantiation of the various policy templates defined in Section 5.4.2 as particular policies is straightforward with the help of a text editor (e.g. Emacs). Class objects were also defined for the two PDP

components, Evaluator and Monitor. As expected, the methods within the classes represent each functionality provided by the elements and components, according to the description given in Section 6.2.1. In addition, a class called *Resource* was defined to represent a generic resource in the model. This class captures the type, origin and state of the resources used in the proposed model and provides methods for their modification.

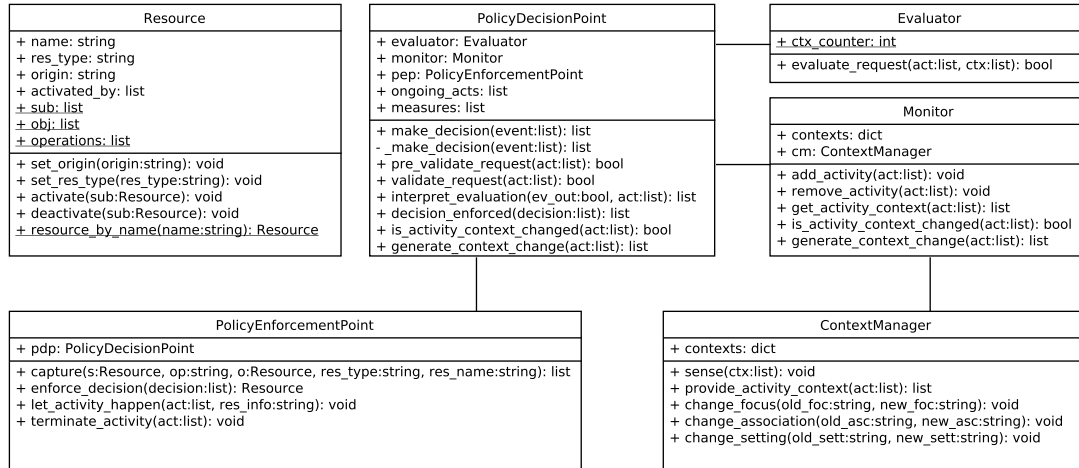


Figure 6.6: Class diagram for the proof-of-concept.

The proof-of-concept prototype completely implements the functionalities of the PDP, as described in the section above. The Evaluator component uses the *pyDatalog* Python library³ as a Datalog engine. *pyDatalog* is a powerful Datalog interpreter with support for negation. It adopts a slightly different syntax than pure Datalog, which requires additional translation of the proposed policy rules.⁴ The Monitor keeps track of the current context of continuous activities and generates context change events.

While the functionalities of the PDP are fully implemented, this is not the case with the functionalities of the PEP and the CM. The proof-of-concept prototype partially implements the functionality of the PEP, without attaching it to the OS. This means that the PEP is not incorporated into the OS, for example through hooking, as explained in Section 6.1.2. Due to this, it cannot capture 'real' activity requests and enforce security decisions for objects (e.g. data, channels and apps). However, the implementation of the PEP provides the necessary functionality for generating activity requests and enforcing security decisions for instances of the Resource class, as objects. By appropriately linking the Resource class with the OS internals (data, channels and apps) and by placing the PEP's *capture* method within OS hooks, the PEP operation is complete. With regard to the CM, the proof-of-concept prototype only simulates its functionalities. The development of context sensing, representation and reasoning are omitted here; instead, the security context is computed 'manually'. This approach was found to be easier when evaluating the feasibility and performance of the decision-making process according to the proposed model, as it does not have a direct influence on this. Nevertheless, the duration of these processes does affect the overall performance of a context-aware security system based on this prototype (i.e. based on the proposed model).

The prototype was deployed on an Android-powered mobile phone. A simple graphical

³<https://sites.google.com/site/pydatalog/>

⁴For example, '&' is used instead of ',' for concatenation of terms, and 'object' is already used as a keyword, meaning it cannot be used as a predicate.

application was developed which wrapped around the proof-of-concept implementation and started simulations of certain scenes from the use cases defined in Section 3.4. The *Kivy* library⁵ and the *buildozer* tool⁶ were used for this purpose. *Kivy* is a Python library for the cross-platform development of mobile applications that makes use of multi-touch user interfaces. It provides a wide range of software components for graphical interaction with the user (so-called widgets such as a button, label, pop-up window etc.), which can be easily combined with other code written in Python. *buildozer* is a tool for packaging mobile applications and their installation on mobile devices. It automates the entire process by handling all the prerequisites needed for porting a Python application to Android. For example, in this case, *buildozer* downloaded and installed the *python-for-android* library,⁷ the Android Software Development Kit (SDK)⁸ and the Android-specific libraries for the particular mobile phone used. It then created a package for Android (known as APK) from the graphical application and installed it on the mobile phone.

6.2.2.2 Relationship with Android

The prototype can be implemented in Android as follows. The whole prototype can be implemented as an application, in which each element is implemented as a specific component. The PDP, PEP and CM can be implemented as services, since these operate in the background and do not interact directly with the user. However, the PAP needs to be implemented as an activity, in order to provide an interface for interaction with the user. The system state and policy database elements can be implemented as content providers, since they are only used for storing and retrieving information. The enforcement layer described in Section 5.2.3 can also be applied to Android, since Android shares a kernel with Linux. However, this is not sufficient, as it does not capture some of Android's specific characteristics. In order to fix this, it is necessary to extend the representation of data and channel objects. Content providers must be represented as data objects in the proposed model, and intents as channel objects.

While the implementation of the PDP, PAP, system state and policy database elements are straightforward and only necessary at the application layer of Android's software stack, the implementation of the PEP and CM is more complicated and requires hooking.

In a similar way to the other Android security enhancements described in Section 6.1.2, the PEP can be implemented in two ways. One approach is to use hooks placed into the application layer; this is practical, as it does not require a modification of Android's source code. However, it frequently breaks applications, and requires an extensive knowledge of their source code in order for all system calls to be appropriately covered [112]. Otherwise, it can be easily circumvented. Another approach is for the PEP to use hooks placed in middleware and/or the kernel layer. This approach requires modification of the OS, and is therefore more difficult to apply on existing mobile devices. Approaches such as SE Android [103] and ASM [112] can facilitate this, since they try to identify relevant hooks in these layers and to provide an interface for them.

The implementation of the CM requires hooking because of virtual sensors. Virtual sensors provide information obtained from certain communications occurring either through the network or between application components. In order to be able to monitor and detect this information, the CM needs to monitor all communications within the mobile device. This can be achieved by exploiting the same hooks used by the PEP for enforcement. For example, a hook used for controlling network communication, which replaces or wraps

⁵<https://kivy.org/>

⁶<https://buildozer.readthedocs.io>

⁷<https://python-for-android.readthedocs.io>

⁸<https://developer.android.com>

around system calls related to socket operations (e.g. send to or receive from socket) can easily provide information about the participants in the communication. This information can be further refined and combined by the CM as the association of an activity. In terms of contextual information from hardware sensors, the CM can monitor and detect this in the same way as any other Android application, through sensor drivers (managers) placed in the middleware layer.

6.2.2.3 Policy Evaluation in Datalog

In terms of Datalog, a policy evaluation is a query as to whether the requested activity is allowed to be performed within a given environment and program (as explained in Section 5.3). A policy evaluation takes place within the Evaluator component from the PDP element of the prototype. As explained above, the Evaluator encompasses a Datalog engine in which the security policies are actually evaluated. Prior to each evaluation, the Evaluator needs to obtain an environment and a program and to provide these to the Datalog engine.

An environment is composed of ground facts which represent the system state and the security context of the requested activity. Since the Datalog engine can easily understand relational databases, system state elements are taken on-the-fly. If it is required to represent these elements as ground facts, a ground fact needs to be defined for each element from each finite set from the domain of discourse of the policy language. Specifically, this includes the following sets: the set of subjects S , the set of objects O , the set of operations OP , the set of social groups SG , the set of property values PR , the set of security contexts CTX and the set of interactions during an activity I . This requires a specific predicate to be used for each finite set. For example, the following ground facts:

$$\begin{aligned} &group(system). \\ &object(launcher). \end{aligned} \tag{6.1}$$

indicate that a social group *system* and an object *launcher* are defined in the system state. In addition, a specific predicate is also needed to define an object's origin. For example, the following ground fact:

$$org(system, launcher). \tag{6.2}$$

denotes that the origin of the *launcher* is the *system* social group.

The security contexts of activities need to be encoded as ground facts in Datalog. As stated above, each element from the set of security contexts CTX is represented as a ground fact. However, these facts actually represent placeholders for security contexts, from which the particular focus, association, setting and security properties are not discernible. In view of this, an additional predicate is introduced called *context_value*, which combines a security context placeholder with the particular values of its elements. For example, the following two predicates:

$$\begin{aligned} &context(c4). \\ &context_value(c4, launcher, open, scr1, work, work, work, -1, -1). \end{aligned} \tag{6.3}$$

indicate that the security context *c4* is defined in the system state, which has an activity element (*launcher, open, scr1*); this has the *work* social group as its focus, association and setting, and the confidentiality and integrity of the channel *scr* is undefined.

A program is composed of rules contained in the applicable security policies, retrieved from the policy database. The question of which security policies are applicable to a

particular evaluation is open for debate. One solution is for the Evaluator to obtain all security policies as applicable. This is a simple solution, since there is only one program for all policy evaluations; however, it can potentially give rise to performance issues, since the number of rules to be consulted may be large. Another solution is for the Evaluator to obtain only those rules which are specified by the social groups whose assets are involved in the requested activity. This is carried out as follows. As applicable group policies are taken, those which are specified by the social groups in the subject's and object's origin, as well as in the focus, association and setting of the requested activity. In the case where asset policies are specified for the particular subject and object, as well as for the social groups in the focus and association, these are also taken into account. All global policies are taken into account by default, since they apply to each security decision. This solution requires the Evaluator to perform additional steps in determining the applicable rules; however, this can improve the performance of the policy evaluation as fewer rules are consulted.

Nevertheless, all applicable policy rules contained in the program need to be translated to pure Datalog, in order for the Datalog engine to be able to understand them. Translation of policy rules is straightforward and proceeds as follows. Firstly, the logic symbols " $\leftarrow$ " and " $\neg$ " are simply translated as " $:$ $\neg$ " and " not ", respectively. Then, the special predicate " $=$ " is represented in infix (e.g., $S = sub1$) instead of prefix notation (e.g., $=(S, sub1)$). Finally, additional predicates are added to each rule, in order to specify to which finite set from the domain of discourse the variables used in the policy rules belong. As expected, they are added with the logical connective of conjunction " $,$ ", in the body of the rules. In addition, auxiliary rules are defined which have the *ctx*, *foc*, *asc*, *set*, *confO*, and *intO* predicates in their heads. These rules extract from the security context the elements that these predicates represent.

Once the environment and the program are provided to the Datalog engine, the policy evaluation takes place by providing the query $allow(s, op, o)$ as a goal in the program.

6.3 Evaluation of the Prototype

The performance of the prototype was evaluated using experiments, during which the use cases defined in Section 3.4 were simulated. In order to be able to do this, the proof-of-concept prototype was extended using additional code. This extended proof-of-concept operated as follows: it received a list of events and security contexts as input, and produced security decisions for each of these as output.⁹ As expected, the security decisions were made with regard to the relevant security policies for the particular use cases. Appendix C describes the operation of this prototype in more detail.

The prototype is only a proof of concept and was implemented mainly to demonstrate the feasibility of the approach presented here. Although it can be run on mobile devices, it is not suitable for this purpose, because it was not developed specifically for Android and thus suffers from poor performance. In order to carry out a relevant evaluation of the proposed model, experiments were performed using an older laptop with the following characteristics: an Intel Core 2 Duo T5470 processor at 1.60GHz, 4GB of RAM, and Ubuntu Linux v14.04 OS. The reason for this was that a computer with this configuration strongly resembles the latest flagship mobile devices, while enabling a smooth execution for the proof-of-concept prototype.

Experiment 1: Each scene from the use cases described in Section 3.4 was simulated. Four social groups were used for each use case (system, work, home and public) and several typical objects were used during the simulations. The operational time of the prototype

⁹As example of these types of inputs and outputs, consider the fields in the tables in Appendix C, which were generated using the proof-of-concept prototype.

was measured while making security decisions. More specifically, the time needed for the prototype's PDP element to make a single security decision was recorded. For each use case scene, the decision-making time was calculated as a mean value of the decision times of all security decisions made during the scene. These values are given in Table 6.1. For comparison, the operation time of some of the operations supported by the proposed policy model and language were measured. These times were calculated as mean value of ten measures and are given in Table 6.2. In both cases, the PyPy implementation¹⁰ of the Python language was used, which runs Python programs faster.

Table 6.1: Duration of decision making in the prototype.

Use Case	Scene	Times (ms)
Use Case 1	Scene 1	29.530±17.407
	Scene 2	33.781±22.562
	Scene 3	29.616±20.984
	Scene 4	30.221±23.114
Use Case 2	Scene 1	30.774±20.527
	Scene 2	33.619±21.770
	Scene 3	30.343±17.578
	Scene 4	32.250±19.558
Use Case 3	Scene 1	34.553±21.093
	Scene 2	35.536±18.936
	Scene 3	34.536±26.123
	Scene 4	34.180±20.965

This experiment showed that the process of decision making in this prototype adds a moderate overhead to the system operation, which is still measured in milliseconds. This overhead is considerable compared with activities involving data resources; however, it is small when compared with activities performed via a network channel (e.g., open and read via a channel). When considering a more complex activity which combines several of the activities supported by the proposed model, the resulting duration of the decision making is the sum of the decision making for each of the combined activities. As an example, consider the complex activity of an app a_1 showing data d_1 on a projector, via a channel c_1 . This combines four activities: 1) app a_1 opens data d_1 ; 2) app a_1 reads data d_1 ; 3) app a_1 opens channel c_1 ; 4) app a_1 writes on channel c_1 . The duration of the decision making for this complex activity to be allowed is cca. $120ms$ ($\approx 4 \times 30ms$), which is still comparable with activities over network channels.

Experiment 2: A single use case scene was simulated (Scene 1 from Use Case 2, selected arbitrarily) with a different number of objects and social groups. The number of objects and social groups influences the performance of the prototype, due to the increased number of facts and rules in the Datalog program; this prolongs the operation time of the Datalog engine and consequently of the prototype. The number of objects increases the number of facts in the Datalog program, since each object needs to be added to the system state. The number of social groups also increases the number of rules in the Datalog program, since the policies of the social groups need to be added to the policy database.¹¹ Table 6.3 shows the operation time of the prototype for a single use case scene with varying numbers of objects and social groups. The PyPy implementation of the Python language was also

¹⁰<http://pypy.org/>

¹¹This also increases the number of facts in the Datalog program, as the particular objects of the social groups need to be added to the system state; however, this is negligible.

Table 6.2: Duration of certain operations on resources. A network channel was used for these measurements. Note that in this case, open and read operations depend heavily on the network traffic and the location of the device on the other side of the channel. In these measurements, this device was a Google server (www.google.com).

Resource	Operation	Time (ms)
Data	open	0.0121 ± 0.0031
	close	0.0178 ± 0.0026
	read	0.0134 ± 0.0040
	write	0.0059 ± 0.0035
Channel	open	11.8966 ± 4.3269
	close	0.0922 ± 0.0214
	read	95.3551 ± 5.5586
	write	0.0627 ± 0.0104

used in this experiment.

Table 6.3: Duration of decision making in the prototype, for varying number of facts (objects) and rules (soc. groups). Times are measured for evaluations of Scene 1 from Use Case 2. The default number of facts and rules is specific to the particular use case scene. (The default number of rules results from four social groups in the prototype; 1,000 rules result from 15 social groups and 10,000 rules result from 150 social groups in the prototype.)

Rules / Facts	264 Facts (Default)	1,000 Facts	10,000 Facts
266 Rules (Default)	30.774 ± 20.527 ms	31.102 ± 20.058 ms	32.059 ± 20.233 ms
1,000 Rules	53.531 ± 32.914 ms	55.757 ± 36.594 ms	57.542 ± 34.449 ms
10,000 Rules	464.722 ± 407.444 ms	488.128 ± 429.542 ms	493.152 ± 433.437 ms

This experiment demonstrates that the number of facts, i.e. objects, in the system slightly increases the duration of decision making, thus affecting the overhead by a small amount. This is very important for the practical usage of the prototype, meaning that it is scalable for applications in real operating system environments, such as Android, which involve a very large number of objects. Furthermore, the experiment shows that the number of rules, i.e. social groups, may have an influence on the overhead. However, it can be expected that users will participate only in a few social groups, and that this will not compromise the system performance.

Chapter 7

Conclusions

This chapter summarizes the work presented in this thesis. Firstly, the way in which the hypotheses postulated in this thesis are verified is discussed. Following this, the original scientific contributions of this work are presented.

7.1 Discussion

The main hypothesis in this thesis is:

Context-aware security based on security policies which utilize higher-level security context abstractions is needed for the protection of ubiquitous social systems.

This hypothesis is verified by verifying the following hypotheses.

Ubiquitous social systems require context-aware security provisioning. Security solutions which are insensitive to context (e.g. those described in Section 3.3.2) are static during their operational time. They usually operate in ‘one-size fits all’ fashion, meaning that the same approaches are applied in various situations of usage and within various social domains. Due to this, they cannot address the specific security challenges (described in Section 3.3.1) arising from the characteristics of ubiquitous social systems (described in Section 3.2). For example, mobile virtualization approaches, such as those in [67–69], enable the appropriate isolation of assets of interest by providing a virtual machine for each social group. However, the user needs to switch between virtual machines manually when the focus changes, which does not correspond to the principles of ubiquitous social systems. In addition, these approaches are restricted to only a single virtual machine operating at one time, which is insufficient for situations when applications originating from two different social groups are required to be running simultaneously (“shared focus” situations).

Security solutions must embrace security context in order to be applicable in ubiquitous social systems. Context-aware security solutions provide fine-grained control over activities and thus facilitate the protection of assets. The security context offers a set of valuable information for describing activities, and this can easily be used in the specification of expressive security policies. These security policies can capture the additional requirements arising from the increased number of interactions in ubiquitous social systems, for each of the social groups involved. Furthermore, context-aware security solutions provide security seamlessly, with minimal user involvement. They are able to detect changes in the security context and react to these changes accordingly. As a user’s focus changes, they adjust their

decisions according to the requirements of the social group which is currently in the user's focus. This is of invaluable assistance for the user, and can minimize the possibilities for accidental mistakes and security incidents. Finally, context-aware security solutions can provide different levels of security within different social groups, depending on their requirements. In this way, they can balance the trade-off between security, performance and efficiency and utilize the limited hardware resources of mobile devices more effectively. Existing consumer-oriented applications provide only a single set of security mechanisms, which are either too much or not enough for certain social groups and their assets. In addition, the proposed context-aware security system (CASS) and its applicability in the typical ubiquitous social system use cases (see Chapter 5.4.3) also contribute to this end.

A conceptual model of security context needs to take into account related social aspects. A ubiquitous social system is a mixture of many domains (e.g. professional, personal or public life) which are part of the life of a single person—the user. Often, many of these domains have conflicting goals or involve complex collaboration patterns. The user uses a single mobile device to perform various activities within these domains and typically performs these activities simultaneously. As a result, ubiquitous social systems blur the boundaries between the various domains, and the activities performed within these domains cannot be easily separated. The concept of social groups is used here to describe these various administrative domains. Three abstractions are used in the proposed conceptual model of security context (focus, association and setting), which are based on the notion of social groups. Through the evaluation of typical use cases in Section 4.4.1, it is shown that these social abstractions contain information which is valuable in describing the various administrative domains and the activities performed within them. Therefore, a conceptual model of security context needs to take into account social context, in order to be applicable to ubiquitous social systems.

A security policy model based on a conceptual model of security context is expressive enough for the protection of ubiquitous social systems. This hypothesis is verified in the exhaustive evaluation presented in Section 5.4. Firstly, the proposed policy model was applied to analyze various security relevant interactions between assets during activities in ubiquitous social systems (Section 5.4.1). This analysis included all the possible operations supported in the proposed model, i.e. read, write, execute, create, and destroy (which are usually supported by other policy models) and open and close (which are specific to the new proposed model). The latter are of great importance in the quest to provide context-awareness and reactive (adaptive) behavior. Following this, the proposed policy model was applied to several specific ubiquitous social systems use cases. In particular, the proposed policy model was used to specify security policies applicable to the use cases and to analyze how these policies provide appropriate protection for the assets in these use cases. The applicable security policies were specified using the proposed policy language, which was defined specifically for the proposed policy model.

The ubiquitous social system use cases were diverse; each use case centers around a member of three social groups as a user. These social groups are referred to as work, home and public, according to their relationship with the user, and were designed to cover various security requirements. While the public social group has similar requirements in all three use cases, as expected, the work and home social groups differ in their requirements. For example, the work social group from Use Case 1 has more conservative requirements, whereas the work social group from Use Case 2 has more liberal requirements which follow modern technological trends. The social groups in the use cases were purposely designed to have significant variations in their requirements, in order to shape the “space of use cases”. However, the proposed policy model remains applicable to many other social groups, whose requirements are either similar, combined or are within this “space”. These use cases were

not arbitrary, but were based on typical scenarios encountered in common practice; they also take into account the security guidelines from standardized security documents.

It is difficult to prove whether the expressive ability of a model is sufficient. During the definition of the proposed conceptual and policy models, an attempt was made to be as straightforward as possible, while remaining as practical as possible. For example, the approach in [115], which was adopted to describe the resources, proved to be successful in the area of security metrics, although it is based on a few simple concepts. The current set of concepts appears to be practical for describing diverse ubiquitous social system use cases. Although the examples are simple, they capture the unique characteristics of ubiquitous social systems.

Security policies based on higher-level security context abstractions facilitate policy specification, management and reuse, while providing adequate performance for context-aware security systems. In general, adaptation to higher level abstractions is more sophisticated for context-aware systems, as has already been pointed out by other researchers [36–38]. This condenses the pool of possible contextual information into a set of a few abstractions, which better represent the human perception of reality. Some higher-level security context abstractions are defined here, namely activity, focus, association, setting, and several security properties.¹ Such a small number of abstractions enabled the development of a policy model for ubiquitous social systems, and consequently the definition of a policy language based on this.

Policy models and languages are essential for the analysis and specification of security policies, as security research has shown many times before (e.g. the policy models and languages described in Section 5.1). Policy makers can use the proposed policy model to analyze various security relevant interactions between assets during activities in ubiquitous social systems, as described in Section 5.4.1. They can then use the proposed policy language to specify security policies that best fit their desired security requirements. Examples of these policies are given in Sections 5.4.2 and 5.4.3 and Appendix B. Moreover, the proposed policy model and language do not place reasoning rules directly into policies, as with some policy models and languages (e.g. [27, 48]), which means that they do not require policy makers to have an exhaustive knowledge of context reasoning. In view of all this, the proposed policy model and language together facilitate policy specification in ubiquitous social systems.

Furthermore, using the proposed policy model for the analysis of security relevant interactions between assets during various activities (presented in Section 5.4.1), inspired the definition of various templates of security policies using the proposed policy language (presented in Section 5.4.2). These templates were useful for specifying group security policies for four different social groups (work, home, public and system), as well as for several asset policies applicable within various ubiquitous social systems use cases (presented in Section 5.4.3). Since these use cases are typical, the defined security policy templates can also be considered useful for other ubiquitous social systems use cases. Therefore, the proposed policy model and language can also be said to facilitate policy management and reuse.

The proposed policy model and language have a modest impact on the performance of context-aware systems. The experiments described in Section 6.3 show that the process of decision making in the prototype adds a moderate overhead to the system operation, which is still measured in milliseconds. This overhead is considerable compared with activities on data resources; however, it is small when compared with activities performed via a network

¹ Although only the confidentiality and integrity of objects are used throughout this thesis, other security properties may also be applicable. However, their number is also expected to be small. For example, such security properties might be the authenticity of the user, the authenticity of participants, the confidentiality and integrity of the subject etc.

channel (e.g. open and read via a channel). More importantly, the number of objects in the system affects the overhead only marginally, as it slightly increases the duration of decision making. The number of social groups may also have an influence on the overhead; however, users are expected to participate only in a few social groups, and this will not compromise the system performance. The prototype is only a proof-of-concept, implemented mainly to show the feasibility of the proposed approach; a prototype developed in Java and natively for Android is likely to show an improved performance over the proof-of-concept prototype. The ever-increasing processing capabilities of contemporary mobile devices are also likely to contribute to this end.

The proposed policy model and language can improve the performance of context-aware security systems. The security policies are relieved from context reasoning rules; this decouples context reasoning from policy evaluation, and can ensure that the performance of the policy decision point (PDP) is unaffected by it. In addition, the defined higher-level security context abstractions are more stable than sensor data, which can be uncertain and may change frequently. This leads to fewer changes in the relevant security context, and consequently to fewer evaluations of the security policies in the PDP. It is likely that the performance of the PDP can greatly benefit from this type of evaluation.

7.2 Conclusion

The work presented in this thesis gives rise to the following original contributions to existing scientific knowledge.

A conceptual model of security context. A conceptual model of security context is defined here. The proposed conceptual model identifies the important concepts that constitute security context and the relationships that exist between them. These concepts are based on a more complete definition of security context, which clearly separates security context from context in general and represents its dynamic nature. The proposed conceptual model describes security context more clearly than the alternatives. It represents security context only through a few high-level abstractions. As pointed out by numerous researchers before, this makes context-aware security systems more sophisticated, since higher-level context abstractions represent the human perception of reality more effectively and are more stable. However, context abstractions relevant to security were lacking in the literature and the proposed conceptual model aims to fill this gap. Moreover, the proposed conceptual model improves the current understanding of security context by introducing social concepts as abstractions for the first time. The social concepts used here are not arbitrary, but are defined in a consistent manner. Altogether, the proposed conceptual model is expressive and practical enough to describe diverse ubiquitous social system use cases and to make them secure.

A security policy model for ubiquitous social systems. A security policy model is defined here for ubiquitous social systems. It identifies the necessary abstractions and rules for the specification and analysis of security policies applicable to ubiquitous social systems. It is based on the proposed conceptual model of security context and represents its practical usage.

The proposed security policy model centers around the concept of security context and gives security context a primary role in decision making. This means that each security decision in this model is made solely on the basis of the security context. However, unlike other models, security context is not treated as an arbitrary concept which needs to be defined afterwards during policy specification. Instead, all possible contextual information is defined as specific abstractions in the proposed policy model. Since the policy abstrac-

tions are few and are at a higher level of abstraction, the proposed policy model enables a clear and precise analysis of the way in which concrete contextual information influences access control, as its underlying security service.

Furthermore, the proposed policy model provides more intuitive abstractions for ubiquitous social systems. In order to address the social aspects of these systems, social concepts are introduced here as policy abstractions. These social concepts are not arbitrary, like those in other policy models, but are defined in a consistent manner based on the proposed conceptual model of security context. They enable the development of expressive security policies which are able to capture the novel requirements arising from the characteristics of ubiquitous social systems. Next, two abstractions were introduced specifically to provide context-aware behavior. One of these (*continuous activity*) provides a means of detecting context changes, while the other (*terminate activity*) provides a means for reacting to context change, in the case where the current security context is no longer appropriate. Finally, a state diagram was designed to represent the lifecycle of an object. This diagram concentrates on the possible states in which an object can exist and a small set of activities which can be performed on an object. As such, it facilitates the detection of context changes and the specification of reactions to these changes.

In addition, the proposed policy model combines the security policies specified by various policy makers. This enables each social group to specify security policies containing its requirements for the protection of its assets of interest. At the same time, this enables the user to specify certain relaxations from these requirements, which refer to a particular asset. During decision making, all policies specified by the social groups whose assets are involved in the activity are taken into account. If a relaxation is specified by the user for one of these assets, then this is also taken into account. However, such relaxations must be first allowed by the social group to which the particular asset is related. In order to facilitate the combination of security policies, several possible interactions are identified between the assets involved in an activity. Essentially, this combination of security policies means that the proposed policy model is a mixture of originator- and discretionary-based control.

A security policy language and templates for ubiquitous social systems. A security policy language is defined for ubiquitous social systems based on the proposed policy model. This policy language enables policy makers to easily specify security policies for ubiquitous social systems. Existing languages are not applicable for this purpose, since they do not provide the appropriate policy abstractions and rules. Even though some of them could be extended, it was found to be more intuitive and straightforward to define a new policy language. This policy language supports only the specification of authorization policy rules, while policy rules for adaptation are handled within the proposed policy model, according to the object lifecycle. These security policies comprise only a few high-level context abstractions, and are relieved from context reasoning rules and policy enforcement actions. This makes them simple and easy to specify, manage and reuse. Moreover, the proposed security policies are consistent and complete, since the new policy language is based on Datalog and this follows from its characteristics.

In addition, a range of security policy templates are defined for ubiquitous security systems. Each template comprises a set of policy rules which are recurring and can be used by various policy makers to control activities. The templates are relevant for the protection of assets, since they are based on thorough security analyses of ubiquitous social systems. They facilitate policy specification and reuse, since each of them can be easily instantiated into readily applicable security policies, as soon as policy makers specify their missing parts. However, policy makers need to understand what type of protection these templates offer and what are the security risks for their assets in each particular situation.

A comprehensive view of the security of ubiquitous social systems. Ubiquitous social systems are described through several use cases. These use cases are representative, as they are based on typical scenarios encountered in common practice, many of which are also described in other works. These scenarios are specifically designed to emphasize the specific characteristics of ubiquitous social systems. In order to capture the important aspects of security, they take into account security guidelines from standardized security documents. As such, these use cases enable a better understanding of ubiquitous social systems. Moreover, these use cases are diverse and cover various security requirements from several social domains. The existing approaches mainly deal with specific use cases, which usually address only a part of the ubiquitous social system. In view of this, the use cases are useful for a more complete and realistic evaluation of context-aware security systems, applicable to ubiquitous social systems.

Ubiquitous social systems are systematically analyzed here from the perspective of security. This involved two analyses. Firstly, the security relevant interactions between the assets involved in the activities supported by the policy model were analyzed. This analysis identified the possible threats to the assets during these interactions, together with potential consequences for their originating social groups. Both analyses aid in a fuller understanding of the security challenges of ubiquitous social systems and how these can be addressed with appropriate security policies. Secondly, specific use cases were analyzed by describing the operation of an applicable context-aware security system. This system utilizes the proposed conceptual model of security context, and the proposed security policy model and language. In particular, the way in which the system reasons about the contextual information which is part of the security context, and how it uses this to control activities, is described. In addition, all the security policies needed for the system to make security decisions are specified. These analyses are invaluable for policy makers to better understand and use the proposed policy templates during policy specification.

A novel approach for context-aware security provisioning. Three tasks present a challenge for designing and developing a context-aware system: context specification, management and usage. Of these, context specification and usage are specific to a particular system and are domain-dependent. Existing context-aware approaches in the security domain often omit the specification of context, and concentrate more on context usage aspects. In particular, they are mainly concerned with how to incorporate security context into security policies, and rarely include reactions (adaptation) as security decisions. In view of this, it is unclear whether and how these approaches adapt to context changes, and whether they do in fact provide context-awareness.

The proposed conceptual model facilitates the neglected task of context specification by clearly identifying the important concepts constituting security context and the relationships that exist between them. This helps system designers and developers to better understand the events which their systems are required to be aware of and adapt to, and to select this contextual information from the wide range of possibilities. Furthermore, not only do the proposed policy model and language enable the analysis and specification of security policies comprising security context, they also provide a means of modeling the reactive (adaptive) behavior relevant to security. This facilitates the task of context usage, since the proposed policy model aids policy makers in easily representing their desired security requirements for the protection of their assets of interest. Overall, by combining knowledge from context-aware systems, adaptable systems and security, this thesis facilitates the design and development of context-aware security systems.

7.3 Overall Vision

The work in this thesis advocates a context-aware security system (CASS) as a solution to the security challenges of ubiquitous social systems. The fundamental elements for the development of a CASS are defined and a proof-of-concept prototype is implemented using these; however, several tasks related to the administration of social groups are further needed, in order for this solution to be completely applicable in practice. In this section, the characteristics of the proposed CASS are summarized and the necessary complementary tasks are described.

The proposed CASS acts as a system application, enabling the specification of security policies and their enforcement on mobile devices running on Android. It monitors all resources on the device and intercepts each request for an activity initiated by the user. After examining the security context of the requested activity, the CASS decides whether to allow or deny it, according to the specified security policies. For some activities, the CASS continues to monitor their security context, and terminates them if they become inappropriate. The necessary policy model and language for the specification of security policies applicable in the CASS are defined. Moreover, a range of policy templates are defined to facilitate policy specification. In addition, a security analysis of ubiquitous social systems is provided, which is particularly useful in assessing the risk to the social group assets, a process preceding policy specification.

Four administration tasks are envisioned as being complementary to the proposed CASS, as follows.

- *Maintaining the structure of a social group.* Each member needs to be added to the list of social group members and assigned a role within the group, according to its competences and responsibilities. Since the role of a member can change during its participation in the social group, it needs to be updated accordingly. Members who leave the group need to be removed from the member list.
- *Obtaining information about social group members.* Various information about social group members are needed for accurate reasoning about the security context. The most important is the one necessary for their authentication. In its simplest form, a person can be authenticated by identifying their mobile device through certain network components (e.g. NFC or Bluetooth). A more secure approach would involve biometrics or a particular authentication protocol (e.g. Kerberos or Open ID), which would require further configuration. This type of authentication protocol would need to be designed specifically for this purpose. Other information about social group members, such as age, gender, relations or interests, can be obtained from social networks, or can be provided manually by the user.
- *Specifying context reasoning rules.* While some reasoning rules, for example those for reasoning about activities or security properties, might be applicable to all social groups, others, such as those for reasoning about focus, association and setting, seem to be specific for each social group. It is expected that social groups will have to specify their own rules for reasoning about focus, association and setting, according to their needs. In the same way as for security policies, certain templates of rules can be defined for reasoning about high-level contextual information, which can be easily instantiated by the social groups.
- *Deploying specific elements of social groups.* This includes the structure of the social group, information about its members, its context reasoning rules and its security policies. All these elements need to be deployed for each of the social group members.

Since many of these information change, for example, as members join or leave the social group, maintaining up-to-date information on each of the members' mobile devices is challenging. The existence of a certain service that will automate this task by pushing the updated information to the mobile devices when possible is envisioned.

It is also envisioned that each social group will have certain member(s) which will be responsible for its administration. For example, a social group administrator for a company (work social group) can be a member of the IT department, whereas for a family (home social group) it can be any member, e.g. one that is tech-savvy. For public and system social groups, CASS designers are expected to act as administrators, providing the necessary information either by default, or through some publicly available service. Many of the administration tasks may be able to be automated, making the administrator's job easier.

The correct operation of the proposed CASS depends on the integrity of the security policies, the reasoning rules and the contextual information. Each of these items needs to be appropriately secured during their deployment and verified prior to their use, in the same way as data in general. In particular, three integrity properties of data need to be assured: (i) data integrity, which is the property that the values contained in the data have not been changed in an unauthorized manner; (ii) source integrity, which is the property that the data is trustworthy, based on its source; and (iii) correctness integrity, which is the property that the underlying information represented by the data is accurate and consistent. For this purpose, standard security mechanisms for data security can be used, e.g. [50].

7.4 Future Work

In this section, several directions are suggested in which the work carried out in this thesis can be extended.

Real implementation. The prototype is implemented only as a proof of concept, mainly to show the feasibility of the proposed approach. The development of a more realistic implementation for mobile devices operating on Android is planned; a brief description of how this can be achieved is given in Section 6.2.2.2. For this implementation, the Java programming language will be used, as it has complete access to Android APIs; this implementation is expected to offer better performance than the proof-of-concept, since it will use native Android libraries.

Combination with other models. The proposed policy model was not designed to cover all possible use cases of access control. Instead, the goal was to address the specific challenges of ubiquitous social systems. In this regard, the proposed model concentrates on protecting assets from interactions between social groups, leaving all interactions within the same social group as allowed. For example, according to a policy based on the proposed closed interaction templates, home applications are not allowed to open work data, while any work application is allowed to do this. Sometimes, such coarse-grained control is not enough, and a subject should not be able to access all the objects originating from the same social group. Fortunately, there are existing policy models that already enable the specification of such security policies. Consider, for example, the Domain Type Enforcement (DTE) model [72, 73]; this model enables us to specify security policies that control which type of objects subjects from a certain domain are allowed to access. If social groups are related to the concept of domains, the proposed policy model can easily be combined with DTE. In this way, these two models will act in complementary manner, with the

proposed model being “above” DTE as it deals with more abstract concepts. Nevertheless, any combination of models will further require examination of how combined policies can be evaluated and enforced, as well as the appropriate extension of the prototype’s architecture.

Additional information flows. Even though the proposed security policies are shown to be sufficient for diverse use cases, they mainly allow interactions between assets related to the same social group (“closed interactions”). Interactions between assets from different social groups are allowed only in two cases: (i) if they involve a system subject, in which case they are essential for an operating system which functions in several social groups; or (ii) if they are explicitly guided by the user, in which case they are essential for satisfying the user’s modern requirements. These “shared interactions” are not explored further, since this work primarily concentrates on identifying and analyzing the possible interactions and information flows that occur through the assets. This task remains as future work. These interactions range from simple flows between two social groups to more complex ones, which arise from complex collaboration patterns between social groups, particularly in the business domain. As an example of the latter, consider a use case where a user works for three companies, two of which are close collaborators and the third is completely unrelated to them. Existing policy models such as [71, 73], provide some solutions which might be useful for the extension of the proposed model. Naturally, these need to be adapted first, in order to handle information flows through sensor and network channels.

Security properties. Security properties provide valuable information about the operation of (context-aware) security systems. By including this information in the proposed security policies, policy makers are enabled to define certain levels of security as their requirements. However, security properties are only incorporated as abstractions in the proposed policy model and language, leaving their definition and evaluation as future work. A brief description is given of how two security properties can be reasoned about and evaluated; however, this is only “scratching the surface”, as the intention was simply to demonstrate their applicability to ubiquitous social systems. The definition and evaluation of security properties are challenging tasks, which are probably worth a doctoral thesis in themselves. An excellent example of how this can be done is the Serenity project [33], which provides a good starting point for this future work. It seems likely that their approach can be further extended for application to ubiquitous social systems. Furthermore, the definition of security properties will open even new possibilities for future work. The prototype is currently able to provide only a simple adaptation, such as closing an opened resource in the case where the security context is no longer appropriate. However, once security properties are defined, context-aware security systems can become capable of more complex adaptations, for example, to automatically achieve a certain security property value. Work in this direction will probably require an appropriate extension of the proposed policy model and language in order to capture the definitions of security properties, and an extension of the proposed prototype architecture in order to capture how security property values can be achieved.

Appendix A

Evaluation of the Conceptual Model of Security Context

In this appendix, the proposed conceptual model of security context is used for analysis of Use Case 2 and Use Case 3 from Section 3.4; this analysis is same as the analysis of Use case 1 presented in Chapter 4. During the analysis presented here, one possible operation of the context-aware security system (CASS) is described which utilizes the proposed model. In particular, the way in which the CASS reasons about the security context of activities and uses it to control them is described. An attempt is made here to give a representative example; therefore, activities and security contexts that repeat or are similar to those already described are not discussed in further detail. Appendix C lists all the activities performed during the use cases and their security contexts. Four types of activities are included in the analysis: create, read, write and execute, whose meaning was given in Chapter 4. (Note that, Appendix B lists all the activities performed during the use cases and their security contexts, which also includes few more activities defined in Chapter 5.) In addition, Table A.1 describes the functionality of the applications used within the analyzed use cases.

Table A.1: Description of functionality of applications used in Use Case 2 and Use Case 3.

Application	Functionality
Launcher	Starts an application upon user request.
Installer	Installs an application from an app store.
Network Manager	Connects to a device from a network infrastructure, through the host's wireless capabilities (e.g., WiFi, 3G, etc).
NFC Manager	Connects to a device through the host's NFC capability.
Office	Enables word processing of various office documents.
File Manager	Enables managing of files and folders stored in host's memory.
VoIP	Enables voice communication between users over the Internet.
Camera	Takes photos and enables their management.
Cab	Organizes transportation within a city.
Email	Enables communication through emails.
Traffic	Generates driving routes between two locations, with respect to traffic conditions.
Notifier	Signals the user about some app's activity, in case this app is not allowed to interact with the user at the moment.
Video	Plays video files.

A.1 Use Case 2

Carol's work and home social group are defined as follows. Her work social group represents her employer, i.e., the ACME2 company. This social group consists of all persons employed by the company, as well as the various devices owned by it. This includes: Carol, Clark, access points, routers, printers, company servers, an access control device at the entrance of the company premises, etc. Carol has the role of researcher in this social group. We denote $work = ((Carol, researcher), (Clark, researcher), (AP_{W21}, access\ point), (AP_{W22}, access\ point), (R_{W2}, router), (P_{W2}, printer), (S_{W2}, server), \dots)$. On the other hand, her home social group represents her particular apartment. This social group consists of Carol, together with her home appliances, such as her router, stereo system, several smart meters, door lock system, etc. Carol has the role of resident in this social group. We denote $home = ((Carol, resident), (R_{H2}, router), (ST_{H2}, stereo\ system), (SM_{H2}, smart\ meter), (DL_{H2}, door\ lock), \dots)$.

Scene 1. Carol is at home and her focus is her home social group. The CASS determines this as soon as she enters the front door, as she used the NFC capability of her mobile device to unlock the door.

When Carol wants to open the results document she made earlier that day at work, she tries to use the *Launcher_W* app to start the particular *Office_W* app for work purposes. The CASS recognizes this as change in Carol's focus, from home to the work social group and sets this social group as focus of Carol's future activities. Carol uses the *Office_W* app as explained in Use Case 1: she operates with the app through a screen and touch channels, while the app operates with the document by reading it. For the activities which go through the sensor channels, association is the work social group, as Carol is the only participant in them, whereas setting is the home social group, as this is the smallest common social group between Carol and the devices Carol owns, such as her router, stereo system, or several smart meters, all of which emit wireless signals. On the other hand, for the rest of the activities, association and setting are the empty social group, as they happen inside the host. The CASS allows all these activities, according to the company and the home policy.

Next, Carol wants to take a photo of her and her dog. Without paying too much attention, she starts the *Camera_W* app, the one which is installed for work purposes. This application opens a channel to the camera sensor, in the same manner as the ones it opens to the screen and the touch sensor. When the app takes the photo, CASS recognizes this as an activity, a_1 : *Camera_W read cch_{W6}*, where *cch_{W6}* denotes the camera channel. The CASS determines the association of a_1 as the home social group, as this group characterizes the participants of a_1 , consisted of Carol and her appliances in the living room, the best. The CASS also determines the setting of a_1 as the home social group, because there are only home members near the host during this activity. According to the home security policy, a_1 is not allowed to be performed, so CASS denies it.

Carol uses the *Launcher_H* app in order to start the *Camera_H* app, which is installed for home purposes. The CASS recognizes this as a change of her focus from work to the home social group. Carol is allowed to take the intended photo of her and her pet with the *Camera_H* app. This is because the focus, association and setting of the activity analogous to a_1 are the home social group. The photo is saved in the host memory during an activity of the *Camera_H* app writing in a newly created data resource. The security context of these activities are determined as for any activity which happens inside the host.

Scene 2. Carol is in her office and her focus is the work social group. The CASS can determine this on the basis of her previous activities. As we described earlier, CASS can capture a certain activity of Carol authenticating on a company electronic access control system and use it to determine her focus. Carol wants to install an application for

public purposes on her mobile device. Thus, she starts the *Installer_P* app, responsible for installing applications for the public social group. When she tries to use the *Launcher_P* app in order to do this, CASS determines that her focus changes to the public social group.

Prior starting the *Installer_P*, Carol first needs to connect to a company access point, through which she will be able access to the Internet. Thus, she starts the *NetworkManager_P* app for public purposes. In this case, the association of the read/write activities involving a sensor channel is determined as the public social group, because she is the only participant in these activities and this social group is in her focus. The setting of these activities is determined as the work social group. The CASS scans her environment for wireless signals and identifies several entities as observers, for example, the access point and the printer in Carol's office, as well as Carol's colleagues in the near offices, which are recognized by their mobile device. All observers have two social groups in common with Carol— work and public, from which the work social group is more specific and defines them more precisely. Thus, it sets this social group to be the setting of these activities.

Upon start, the *NetworkManager_P* establishes a network channel ch_{P3} to the company access point, a_2 : *NetworkManager_P opens ch_{P3}* . This device, denoted as AP_{W22} , is the only participant in the activity. For security reasons, the ACME2 company provides two networks to its employees: one intended for work purposes and the other for public purposes. In this way it can keep public apps out of the corporate network, which essentially results in isolation of the corporate network. Device AP_{W22} is part of the “public network” provided by ACME2 company. It is member of both work and public social group, as any other device owned by ACME2, but its focus is set to be the public social group. This is opposite from devices being part of the work social group, which have the work social group in their focus. As a result, CASS determines the association of a_2 as the public social group. It further determines the setting of a_2 as the work social group, as it is the smallest social group in common for the participants and observers. The set of observers consists of the same social entities near the host as for previous activities occurring through sensor channels. According to the company policy, this activity is allowed. Devices like to AP_{W22} act similar as the host in this case, i.e., they are members of the work social group, but their focus is the public social group.

Carol starts the *Installer_P*, which consequently opens a channel to a remote server hosting the public app store. She selects the *Cab_P* as desired application for installation and the *Installer_P* downloads its source code and creates a new application resource in the host. During this, association and setting for activities involving screen and touch channels are public and work social groups, respectively, whereas association and setting for activities involving network channel are both the public social group. Also, the security context of the latter activities contains that the confidentiality and integrity of the network channel are achieved, as the *Installer_P* establishes a secure channel with the app store. The origin of the newly created *Cab_P* application is set to be the public social group, according to the origin of the *Installer_P* app. Then, Carol starts the *Cab_P* app, which also opens a channel to a remote server, where requests from all the consumers of the app are processed, similar to the *Installer_P* app. Association and setting for these activities are determined in similar manner. Finally, Carol submits a trip request to the *Cab_P* app and receives a pick-up time. All these activities are allowed according to the company policy.

Scene 3. Carol is in the cab with her colleague Clark and wants to show him some photos of her pet. She intends to use the *Camera_H* app, which, except for taking photos, also servers as a photo viewer. In particular, she intends to use the *Camera_H* app installed for the purposes of the home same group, since the photos originate from the same social group. As expected, Carol tries to start this app through the *Launcher_H* app. Based on

this activity, CASS changes her focus to the home social group.

As usual, Carol operates with the *Camera_H* app through sensor and touch channels. The association for these activities is determined as the home social group, as Carol is their only participant, and their setting is determined as the public social group as Clark and the cab driver are identified as their observers. When Carol tries to show the host's screen to Clark, CASS recognizes Clark also as a participant in the continuous activity of the *Camera_H* app opening the screen channel. CASS can achieve this through face recognition, or through a certain handshake protocol between the host and Clark's device. Upon this, CASS determines the association of this activity as the work social group and detects context change. According to the home policy, this activity is not allowed and needs to be terminated. Carol defines an asset policy for the *Camera_H* app, in order to be able to show it on the screen to Clark. After this, CASS allows the *Camera_H* app to open a channel to the screen, and Carol can show the intended photos to Clark.

Scene 4. Carol is at the airport, reviewing her presentation slides with the *Office_W* app. Her focus is her work social group, which is determined when she used the *Launch_W* app to start the *Office_W* app. For activities during the review which go through the sensor channels (screen and touch), association is the work social group, as Carol is the only participant. At the same time, the setting for these activities is the public social group, as there are many passengers in the hall which observe this activity, all of which are sensed by their mobile devices. On the other hand, for the activities that happen inside the host, association and setting are the empty social group.

When Carol wants to send the presentation slides to Clark, CASS recognizes this as an activity of the *Office_W* app trying to write to a channel, established with Clark's device. Since Clark is the only participant in this activity and since his focus is the work social group¹, the CASS determines the association of this activity as work. As can be expected, the setting of this activity is determined as public because of the many passengers being around. As some of them may capture this communication if unencrypted, CASS does not allow this activity unless a secure channel is established between the host and the Clark's device. This will achieve the properties of confidentiality and integrity of the communication channel.

A.2 Use Case 3

Bob's work and home social group are defined as follows. His work social group represents his employer, i.e., the ACME3 company. This social group consists of all persons employed by the company, as well as the various devices owned by it, access points, routers, company servers, etc. Bob has the role of seller in this social group. We denote $work = ((Bob, seller), (AP_{W3}, access\ point), (R_{W3}, router), (S_{W3}, server), \dots)$. On the other hand, his home social group represents his particular house. This social group consists of Bob, together with his home appliances, such as his router, TV set, several smart meters, door lock system, etc. Bob has the role of resident in this social group. We denote $home = ((Bob, resident), (R_{H3}, router), (TV_{H3}, TV\ set), (SM_{H3}, smart\ meter), (DL_{H3}, door\ lock), \dots)$.

Scene 1. Bob is at home, but his focus is his work social group. The CASS recognizes this as soon as Bob tries to use the *Launcher_W* app to start other work applications, such as the *NetworkManager_W* and the *Email_W* apps. This focus applies for all activities performed in this scene.

Bob operates with the *Launcher_W*, *NetworkManager_W* and *Email_W* apps as usual, through screen and touch channels. All these activities have the work social group as their

¹A certain exchange protocol between the host and Clark's device can help CASS to determine Clark's focus.

association, whereas they have the home social group as their setting. This is because Bob is their only participant, and Bob's home appliances are their observers. As expected, the activities of *Launcher_W* executing *NetworkManager_W* and *Email_W* apps have both association and setting empty, as they happens inside the host.

First, Bob starts the *NetworkManager_W* and connects to the home router, in order to get access to the Internet. The activities of opening, reading and writing through the network channel have the home social group as their association, as this social group is in the focus of the Bob's router, which is their only participant². The setting of these activities is the home social group, as Bob's home appliances remain their observers, as for the previous activities. The CASS allows all these activities according to the ACME3 company policy.

Then, Bob starts the *Email_W* app, which automatically connects to the company mail server and downloads Bob's work mails. The mail server acts as the only participant in the activities during this communication. We consider that its focus is the work social group and CASS knows this in advance. Thus, CASS determines the association of these activities as the work social group. Furthermore, this communication goes through a network channel, which is composed of Bob's home router, several company routers owned by ACME3, and many routers in between, which are part of a certain network infrastructure of the Internet provider(s). All these entities act as observers of the activities occurring through the channel, together with Bob's home appliances, as the host communicates with the Bob's router wirelessly. Thus, CASS determines the public social group as setting for these activities. Finally, the network channel is established with the help of the SSL protocol. This makes CASS to determine that the properties of confidentiality and integrity of the channel are achieved. As a result, the CASS allows all these activities according to the ACME3 company policy.

Scene 2. Bob is in his car and his focus is the public social group. The CASS can determine this as soon as Bob leaves his house and locks the front door with his mobile device.

Bob uses the *Traffic_P* app, which is installed on his device for public purposes. The app has opened four sensor channels, with: (i) the screen and touch sensors for visual communication with the user, (ii) the speaker, for audio communication with the user, and (iii) the GPS sensor, for obtaining the user's current location. In addition, the *Traffic_P* app has opened one network channel, for communication with a certain remote server, which provides more computation power for its functional tasks. CASS determines association and setting for activities occurring through the sensor channels as the public social group. This is because, Bob is their only participant and all entities surrounding Bob as he moves through the city are their observers. The public social group is in Bob's focus and it is the only common social group of the observers of these activities. Similarly, CASS determines association and setting of activities occurring through the network channel as the public social group, because, the remote servers is their only participant, and the many routers composing the network channel are their observers. The CASS allows all these activities.

While Bob uses the *Traffic_P* app, the *Voip_W* app, installed for work purposes, is working in the background. This app is allowed to receive an incoming call through a network channel, but it cannot communicate with Bob through a screen and touch channels, which means Bob cannot answer the call. CASS determines the association for activities occurring through network channels as the work social group, as their only participant is the colleague calling Bob. In addition, it determines that the confidentiality and integrity of the network channel is achieved as the *Voip_W* app establishes a secure channel with

²One can imagine that CASS knows this in advance, or receives this information through a certain protocol.

the participant's mobile device. On the other hand, CASS determines the association for activities occurring through sensor channel as the public social group, as their only participant is Bob and this group is in his focus. The setting for all these activities is the public social group.

The *Notifier* app, which is a system app, captures these events that the *Voip_W* app is denied access to one of the sensor channels, and notifies Bob, by writing short notification on the screen. Both association and setting of this activity is the public social group. When Bob tries to open the *Voip_W* app, for example through the *Launcher_W*³, CASS changes his focus from public to work social group. As a result, the *Traffic_P* app and all channels it has opened are closed.

After the focus change, the *Voip_W* app is able to operate with the screen and touch sensors. This is because association for these activities becomes the work social group. Bob calls his colleague back, by using the hands-free option. This makes the *Voip_W* app to open a channel to the microphone and the rear speaker. Although this speaker is louder than the front one, voices coming from it still cannot be heard from surrounding people, because Bob is driving with his car. Thus, CASS determines the association for these activities also as the work social group. In addition, CASS determines the setting for all these activities performed by the *Voip_W* app as the public social group, same as before.

When Bob exits the car, the association for the writing activity to the rear speaker becomes the public social group, as people surrounding Bob can now hear the emitted voices. As a result, CASS closes this channel. Bob must turn off the hands-free option in order to continue with the conversation. This makes the *Voip_W* app to open a channel to the front speaker, which is quieter. CASS allows this activity as its association is the work social group.

Scene 3. Bob is having a company meeting at a public venue and his focus is the work social group. CASS can recognize this focus similar as in the previous scenes, for example, as a result of a communication with a certain access control system, placed at the entrance of the conference room where the meeting is going to be held. This access control system can be introduced as a temporary member of the work social group. As Bob authenticates on it, CASS can recognize that his focus changes to the work social group.

Bob starts the *Office_W* app with the *Launcher_W*. It communicates with both applications through screen and touch channels. Association for these activities is the work social group, as Bob is their only participant, and setting is the public social group, as CASS senses many people outside the conference room, whose only common social group is the public social group. CASS allows these activities according to the ACME3 company policy.

Next, Bob opens the presentation slides and connects to the projector through a wireless network channel. Activities that happen through this channel have the work social group as their association, as only members of this group are currently present in the conference room. At the same time, setting for these activities is the public social group, as CASS senses that other entities are also present outside of the conference room. CASS allows these activities, as long as the confidentiality and integrity of the channel is achieved, that is, the communication between the host and the projector is encrypted.

When the security guard enters the conference room, association for these activities, happening through the network channel, change to the public social group. CASS can recognize this, for example, with the help of the access control system, which acts as some kind of context provider in this case. According to the ACME3 security policy, CASS needs to close this channel and thus terminate the presentation. In this way, the security guard will not be able to see the confidential company data on the screen.

³In fact, this only brings *Voip_W* app to the screen, as it is already executed and running.

Scene 4. Bob is at the front of his house, and he uses his mobile device to unlock the door. CASS recognizes this as a change in his focus, from public to home social group. As a result, it closes all public applications and the objects they have opened. System applications remain opened, together with two work applications: *Email_W* and *Voip_W*. Similar as in Scene 2, Bob has specified asset policies for these apps, so that they can operate in background while home social group is in his focus.

Bob opens the *Video_H* app and starts watching a movie on his TV set. *Video_H* is an application intended for home purposes and the TV set is a member of the home social group. These are all standard activities, which involve assets related with a single social group. Thus, CASS allows all of them.

Email_W app operates during all these activities. However, it cannot use screen and touch channels, due to it is a work application and associations of these activities is the home social group. Nevertheless, it can receive emails through a network channel, in which case association is the work social group, as the mail server is a member of this social group. It can also write the received emails in a database file. When such email arrives, the *Notifier* writes a short notification on the screen. If Bob wants to read a received email, he needs to first open the *Email_W* app with the *Launcher_W*⁴, which changes his focus to the work social group. This makes CASS to close the *Video_H* app, together with all objects it has opened.

⁴In fact, this only brings *Email_W* app to the screen, as it is already executed and running.

Appendix B

Evaluation of the Policy Model and Language

In this appendix, the proposed policy model and language are used for analysis of Use Case 2 and Use Case 3 from Section 3.4; this analysis is same as the analysis of Use Case 1 presented in Chapter 5. During the analysis, the applicable security policies are specified and the way in which security decisions are made is described in detail. In order to give a representative example, security decision which repeat or are similar to the ones already described are not described in further detail here; Appendix C lists all the security decisions for each of the activity requests. This analysis is complementary with the one in Section 4.4.1 and Appendix A, and is based on the same assumptions. (Note that, Appendix B lists all the security decisions for each of the activity requests.)

In addition, we specify group policies of a generic company and home device. We omit their system policy, as it remains the same as defined above. One can imagine that all devices, which are part of the work and home social group have these policies installed within the CASS, respectively. Tables B.1 and B.2 lists the templates from which these policies are composed of.

Table B.1: Group policies of a generic company device. Each field lists the templates from which the policies are specified.

Interaction/Social group	Work	Public
Subject-Object	Closed subject and object Shared channel with system subject	Closed subject and object Shared channel with system subject
Subject-User	Closed subject and focus Shared focus with system subject	Closed subject and focus Shared focus with system subject
Subject-Participant	Closed subject and participant Shared participant with system subject	Open subject and participant
Subject-Observer	Closed subject and observer Shared observer with system subject Shared subject with public observer	Open subject and observer

Table B.2: Group policies of a generic home device. Each field lists the templates from which the policies are specified.

Interaction/Social group	Work	Public
Subject-Object	Closed subject and object Shared channel with system subject	Closed subject and object Shared channel with system subject
Subject-User	Closed subject and focus Shared focus with system subject	Closed subject and focus Shared focus with system subject
Subject-Participant	Closed subject and participant Shared participant with system subject	Open subject and participant
Subject-Observer	Open subject and observer	Open subject and observer

B.1 Use Case 2

Tables B.3, B.4 and B.5 list the interaction templates from which work, home and public social group policies are composed of, respectively.

Table B.3: Work social group policy.

Interaction	Template
Subject-Object	Closed subject and object
	Shared channel with system subject
Subject-User	Closed subject and focus
	Shared focus with system subject
Subject-Participant	Closed subject and participant
	Shared participant with system subject
	Shared participant with home subject (incoming flow)
Subject-Observer	Open subject and observer

Table B.4: Home social group policy.

Interaction	Template
Subject-Object	Closed subject and object
	Shared channel with system subject
Subject-User	Closed subject and focus
	Shared focus with system subject
Subject-Participant	Closed subject and participant
	Shared participant with system subject
	Shared subject with work participant, with relaxation (outgoing flow)
Subject-Observer	Open subject and observer

Table B.5: Public social group policy.

Interaction	Template
Subject-Object	Closed subject and object
	Shared channel with system subject
Subject-User	Closed subject and focus
	Shared focus with system subject
Subject-Participant	Open subject and participant
Subject-Observer	Open subject and observer

Scene 1. Carol is at home and her focus is her home social group. She wants to check a work document, so she uses the *Launcher* to start the *Launcher_W*, which is responsible for starting the appropriate work application for this purpose. The CASS recognizes this activity request as a change in Carol’s focus, from home to the work social group. Upon this change, it reacts by making new security decision for each continuous activity, with respect to the updated security context. Since the work social group uses the closed and the *Shared focus with system subject* templates for subject-user interaction, the CASS closes all objects originating from other social group. One can imagine that these are the various

home applications that Carol have used just prior the context change, as well as the object they have opened.

Generally, Carol can operate with work apps at home, as if she is at work. Her work social group allows subject-observer interactions in home setting, as it uses the open template for this interaction. This interaction is allowed by the home social group too, as it is also based on the same template. Whether an activity is allowed or not, depends on the other interactions. For example, Carol is allowed to operate with the *Office_W* app over work documents, as all assets involved in these activities, other than observers, are related with the work social group. However, she is not allowed to take photos with the *Camera_W* app, installed for work purposes. This is due to association of this activity is the home social group, and such activities are not allowed from aspect of both work and home social groups, as they use the closed template for the subject-participant interaction in their policy. If Carol wants to take photos, she needs to use the *Camera_H* app, the one whose origin is her home social group.

Scene 2. Carol is in her office and her focus is her public social group. Her employee allows her to use public applications while at work, according to the work group policy for subject-observer interactions, which is based on the open template. The public social group allows this interaction during activities too, as it is based on the same template. Thus, whether an activity is allowed to be performed or not, depends on the other interaction during it.

In this scene, Carol uses: (i) the *NetworkManager_P* app to communicate with the special access point *AP_{W22}*, (ii) the *Installer_P* app to communicate with a public apps store, and (iii) the *Cab_P* app to communicate with a remote server. All these applications have the public social group as their origin and perform activities (e.g., open, write and read) over sensor and network channels they have created themselves. Activities through sensor channels (e.g., screen and touch channels) have public social group in their association, same as activities through network channels. This means that only security policies specified by the public social group apply for subject-object, subject-user and subject-participant interactions during these activities. The public social group uses closed templates for the first two, whereas it uses the open template for the third interaction. According to these templates, all interactions are allowed during these activities. As a result, CASS allows all these activities to be performed.

Scene 3. Carol is in the cab with her colleague Clark and wants to show him some photos of her pet. Her focus is her home social group and setting of all activities through screen and touch channels is the public social group. CASS allows Carol to use home applications in public setting as both home and public social group use the open template for subject-observer interaction.

CASS allows Carol to operate with the *Camera_H* app through touch and screen channel, as long as the association of these activities is her home social group. However, when she shows the screen to Clark, the association of continuous activity of *Camera_H* opening the screen channel becomes her work social group. According to the home social group policy for subject-participants interaction, which is based on the closed template, this interaction is not allowed. Thus, CASS terminates this activity by closing the screen channel.

In order to be able to show home photos with the *Camera_H* app to Clark, Carol needs

to specify an asset policy for this application. Such asset policy is given below.

$$\begin{aligned} uallow(camera_H, sp, S, open, O) \leftarrow &= (camera_H, S), org(home, S), \\ &cctx(C, S, open, O), asc(work, C). \end{aligned} \quad (B.1a)$$

$$\begin{aligned} uallow(camera_H, sp, S, close, O) \leftarrow &= (camera_H, S), org(home, S), \\ &cctx(C, S, close, O), asc(work, C). \end{aligned} \quad (B.1b)$$

$$\begin{aligned} uallow(camera_H, sp, S, write, O) \leftarrow &= (camera_H, S), org(home, S), \\ &cctx(C, S, write, O), asc(work, C). \end{aligned} \quad (B.1c)$$

Home social group policy supports specification of assets for subject-participant interactions, according to the *Shared subject with work participant with relaxation (outgoing flow)*. At the same time, work social group policy allows home applications to perform activities towards its members, according to the *Shared participant with home subject (incoming flow)* template. As a result, *Camera_H* app can open a screen channel and send information to it, after Carol specifies the asset policy.

Scene 4. Carol is at the airport with Clark and her focus is her work social group. She uses the *Launcher_W* to open the *Office_W* app, in order to review the slides for the meeting. CASS allows Carol to operate with work apps in public setting, according to the *Shared subject with public observer* template used in the work group policy. In particular, CASS allows Carol to use sensor channels as usual, but it requires confidentiality and integrity of channels to be achieved in case of network channels. Thus, when Carol wants to send the slides to Clark, the *Office_W* app needs to first establish a secure channel with his device, for example, by using the SSL/TLS protocol. This ensures the communication between them is encrypted, and the exchanged information cannot be intercepted by a malicious party.

B.2 Use Case 3

Tables B.6, B.7 and B.8 list the interaction templates from which work, home and public social group policies are composed of, respectively.

Table B.6: Work social group policy.

Interaction	Template
Subject-Object	Closed subject and object
	Shared channel with system subject
Subject-User	Closed subject and focus
	Shared focus with system subject
	Shared subject with home focus, with relaxation
	Shared subject with public focus, with relaxation
Subject-Participant	Closed subject and participant
	Shared participant with system subject
	Shared subject with public participant (incoming flow)
Subject-Observer	Open subject and observer

Scene 1. Bob is on the field and his focus is his work social group. He uses the *Launcher_W* to open the *NetworkManager_W* app in order to connect to the 3G network

Table B.7: Home social group policy.

Interaction	Template
Subject-Object	Closed subject and object Shared channel with system subject
Subject-User	Closed subject and focus Shared focus with system subject Shared focus with work subject, with relaxation
Subject-Participant	Closed subject and participant Shared participant with system subject
Subject-Observer	Open subject and observer

Table B.8: Public social group policy.

Interaction	Template
Subject-Object	Closed subject and object Shared channel with system subject
Subject-User	Closed subject and focus Shared focus with system subject Shared focus with work subject, with relaxation
Subject-Participant	Open subject and participant
Subject-Observer	Open subject and observer

provide by his mobile operator. As soon as he connects to the Internet, the *Email_W* app starts downloading his emails. Bob uses the *Launcher_W* to open the *Email_W* app in order to read the new emails. CASS allows Bob to operate with work apps in public setting, according to the *Shared subject with public observer* template used in the work group policy. In particular, CASS allows Bob to use sensor channels as usual, but it requires confidentiality and integrity of channels to be achieved in case of network channels. Thus, when the *Email_W* app connects to the ACME3 company mail server, it needs to do this through a secure channel, for example, by using the SSL/TLS protocol. This ensures the communication between the host and the server is encrypted, and the exchanged information cannot be intercepted by a malicious party.

Scene 2. Bob is driving his car towards the public venue where the company meeting is going to be held. His focus is his public social group.

Bob is using the *Traffic_P* app, which has opened four sensor channels—through the screen, touch, speaker and GPS sensors—and one network channel during its operation. The *Traffic_P* app is a public application and all opened objects originate also from the public social group, as they have been created previously by this app. Association and setting for each activity through these channels are the public social group. As a result, CASS takes only group policies specified by the public social group into account for security decisions regarding these activities. According to these policies, CASS allows all these activities.

While Bob uses the *Traffic_P* app, the *Voip_W* app, installed for work purposes is running in background. Bob has defined an asset policy for this application, so that it can operate

while his focus is the public social group. This policy consists of the following rules.

$$\begin{aligned} uallow(voip_W, su, S, create, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, create, O), foc(public, C). \end{aligned} \quad (B.2a)$$

$$\begin{aligned} uallow(voip_W, su, S, destroy, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, destroy, O), foc(public, C). \end{aligned} \quad (B.2b)$$

$$\begin{aligned} uallow(voip_W, su, S, open, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, open, O), foc(public, C). \end{aligned} \quad (B.2c)$$

$$\begin{aligned} uallow(voip_W, su, S, exec, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, exec, O), foc(public, C). \end{aligned} \quad (B.2d)$$

$$\begin{aligned} uallow(voip_W, su, S, close, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, close, O), foc(public, C). \end{aligned} \quad (B.2e)$$

$$\begin{aligned} uallow(voip_W, su, S, read, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, read, O), foc(public, C). \end{aligned} \quad (B.2f)$$

$$\begin{aligned} uallow(voip_W, su, S, write, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, write, O), foc(public, C). \end{aligned} \quad (B.2g)$$

$$\begin{aligned} uallow(voip_W, su, S, exec, O) \leftarrow &= (voip_W, O), org(work, O), \\ &cctx(C, S, exec, O), foc(public, C). \end{aligned} \quad (B.2h)$$

$$\begin{aligned} uallow(voip_W, su, S, close, O) \leftarrow &= (voip_W, O), org(work, O), \\ &cctx(C, S, close, O), foc(public, C). \end{aligned} \quad (B.2i)$$

CASS takes this asset policy into account, because work social group is based on the *Shared subject with public focus, with relaxation* template, which enables relaxation for the subject-user interaction.

In order CASS to allow the activities performed by the *Voip_W* app, Bob also needs to specify an asset policy for the public social group as a focus. Such policy consists of the following rules.

$$\begin{aligned} uallow(public, su, S, create, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, create, O), foc(public, C). \end{aligned} \quad (B.3a)$$

$$\begin{aligned} uallow(public, su, S, destroy, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, destroy, O), foc(public, C). \end{aligned} \quad (B.3b)$$

$$\begin{aligned} uallow(public, su, S, open, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, open, O), foc(public, C). \end{aligned} \quad (B.3c)$$

$$\begin{aligned} uallow(public, su, S, exec, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, exec, O), foc(public, C). \end{aligned} \quad (B.3d)$$

$$\begin{aligned} uallow(public, su, S, close, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, close, O), foc(public, C). \end{aligned} \quad (B.3e)$$

$$\begin{aligned} uallow(public, su, S, read, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, read, O), foc(public, C). \end{aligned} \quad (B.3f)$$

$$\begin{aligned} uallow(public, su, S, write, O) \leftarrow &= (voip_W, S), org(work, S), \\ &cctx(C, S, write, O), foc(public, C). \end{aligned} \quad (B.3g)$$

$$\begin{aligned} uallow(public, su, S, exec, O) \leftarrow &= (voip_W, O), org(work, O), \\ &cctx(C, S, exec, O), foc(public, C). \end{aligned} \quad (B.3h)$$

$$\begin{aligned} uallow(public, su, S, close, O) \leftarrow &= (voip_W, O), org(work, O), \\ &cctx(C, S, close, O), foc(public, C). \end{aligned} \quad (B.3i)$$

CASS takes this asset policy into account, because public social group is based on the *Shared focus with work subject, with relaxation* template, which enables relaxation for the subject-user interaction.

Voip_W app is allowed to operate only with work participants in public focus. This is determined by the closed template for the subject-participants interaction. This means that *Voip_W* app is not allowed to operate through the standard sensor channels, such as screen and touch channels, as association for these activities is the public social group, according to the Bob's focus. As a result, Bob is not able to answer or start a call with the *Voip_W* app. Nevertheless, he can still receive incoming calls from colleagues. This is allowed since *Voip_W* app can operate through a network channel, if association for an activity is the work social group. When this happens, the *Voip_W* app will try to open a screen or touch channel, but CASS will deny such activity. A certain *Notifier* app can send a notification to the screen about such denial, which acts as a notification about missed call.

Bob opens the *Voip_W* app with the *Launcher_W*, which makes his focus to change to work social group. Upon this, CASS closes all public apps, as well as the objects they have opened. This is due to work group policy is based on the closed template for subject-user interaction and does not allow such application to operate in work focus. Now, the *Voip_W* app can open a touch and screen channel, as association of these activities is the work social group. Thus, Bob is able to call his colleague and start a conversation. The *Voip_W* app opens and uses two additional channels, to the microphone and to the rear speaker. The latter channel is because Bob uses the hands-free option. Association for activities through these channels is the work social group, so CASS allows all such activities.

When Bob gets out of his car, he is still having the conversation with his colleague. Now, association for activities through the microphone and the rear speaker becomes the public social group. CASS allows activities through the microphone channel, because work social group also uses the *Shared subject with public participant (incoming flow)* template for subject-participant interaction. However, it does not allow activities through the rear speaker, so it closes this channel. Bob needs to turn off the hands free option in order to continue with the conversation. This is because the association for activities through the front speaker is still the work social group.

Scene 3. Bob is in the conference room where the meeting is going to be held and his focus is his work social group. He is about to start his presentation on the projector, by using the *Office_W* app. He opens the prepared slides and connects to the projector through a wireless network channel. CASS allows Bob to operate with the *Office_W* app in public setting, according to the *Shared subject with public observer* template used in the work group policy. In particular, CASS allows Bob to use sensor channels as usual, but it requires confidentiality and integrity of channels to be achieved in case of network channels. Thus, when *Office_W* app connects to the projector, it needs to do this through a secure

channel, for example, by using the SSL/TLS protocol. This ensures the communication between the host and the projector is encrypted, and the exchanged information cannot be intercepted by a malicious party.

When the security guard enters the room, the association of the continuous activity of the *Office_W* app opening the network channel to the projector changes to the public social group. This activity is no longer allowed by the *Closed subject and participant* template used in the work group policy. Thus, CASS terminates this activity by closing the channel. In this way, the security guard is not able to read the confidential information from Bob's presentation on the projector.

Scene 4. Bob is at home and his focus is his home social group. He is watching a movie on his TV set. The movie is streamed from his mobile device with the help of the *Video_H* app, through a wireless channel. CASS allows the home app *Video_H* to operate through standard screen and touch channels and through a network channel to the TV set. All these channels are also home object, and association and setting for these activities is the home social group. According to the home group policies, which are based on the closed and opened templates, all these activities are allowed.

While watching the movie, the *Email_W* app for work purposes is operating in background. Similar to the *Voip_W* app, Bob has also specified an asset policy for this application, which consists of the following rules.

$$\begin{aligned} uallow(email_W, su, S, create, O) \leftarrow & (email_W, S), org(work, S), \\ & cctx(C, S, create, O), foc(home, C). \end{aligned} \quad (B.4a)$$

$$\begin{aligned} uallow(email_W, su, S, destroy, O) \leftarrow & (email_W, S), org(work, S), \\ & cctx(C, S, destroy, O), foc(home, C). \end{aligned} \quad (B.4b)$$

$$\begin{aligned} uallow(email_W, su, S, open, O) \leftarrow & (email_W, S), org(work, S), \\ & cctx(C, S, open, O), foc(home, C). \end{aligned} \quad (B.4c)$$

$$\begin{aligned} uallow(email_W, su, S, exec, O) \leftarrow & (email_W, S), org(work, S), \\ & cctx(C, S, exec, O), foc(home, C). \end{aligned} \quad (B.4d)$$

$$\begin{aligned} uallow(email_W, su, S, close, O) \leftarrow & (email_W, S), org(work, S), \\ & cctx(C, S, close, O), foc(home, C). \end{aligned} \quad (B.4e)$$

$$\begin{aligned} uallow(email_W, su, S, read, O) \leftarrow & (email_W, S), org(work, S), \\ & cctx(C, S, read, O), foc(home, C). \end{aligned} \quad (B.4f)$$

$$\begin{aligned} uallow(email_W, su, S, write, O) \leftarrow & (email_W, S), org(work, S), \\ & cctx(C, S, write, O), foc(home, C). \end{aligned} \quad (B.4g)$$

$$\begin{aligned} uallow(email_W, su, S, exec, O) \leftarrow & (email_W, O), org(work, O), \\ & cctx(C, S, exec, O), foc(home, C). \end{aligned} \quad (B.4h)$$

$$\begin{aligned} uallow(email_W, su, S, close, O) \leftarrow & (email_W, O), org(work, O), \\ & cctx(C, S, close, O), foc(home, C). \end{aligned} \quad (B.4i)$$

CASS allows *Email_W* app to operate in home focus, same as it allows *Voip_W* app to operate in public focus, according to the *Shared subject with home focus, with relaxation* template used in the work group policy, the *Shared focus with work subject, with relaxation*

template used in the home policy and the asset policy for the home focus, consisted of the following rules.

$$\begin{aligned} uallow(home, su, S, create, O) \leftarrow &= (email_W, S), org(work, S), \\ &cctx(C, S, create, O), foc(home, C). \end{aligned} \quad (B.5a)$$

$$\begin{aligned} uallow(home, su, S, destroy, O) \leftarrow &= (email_W, S), org(work, S), \\ &cctx(C, S, destroy, O), foc(home, C). \end{aligned} \quad (B.5b)$$

$$\begin{aligned} uallow(home, su, S, open, O) \leftarrow &= (email_W, S), org(work, S), \\ &cctx(C, S, open, O), foc(home, C). \end{aligned} \quad (B.5c)$$

$$\begin{aligned} uallow(home, su, S, exec, O) \leftarrow &= (email_W, S), org(work, S), \\ &cctx(C, S, exec, O), foc(home, C). \end{aligned} \quad (B.5d)$$

$$\begin{aligned} uallow(home, su, S, close, O) \leftarrow &= (email_W, S), org(work, S), \\ &cctx(C, S, close, O), foc(home, C). \end{aligned} \quad (B.5e)$$

$$\begin{aligned} uallow(home, su, S, read, O) \leftarrow &= (email_W, S), org(work, S), \\ &cctx(C, S, read, O), foc(home, C). \end{aligned} \quad (B.5f)$$

$$\begin{aligned} uallow(home, su, S, write, O) \leftarrow &= (email_W, S), org(work, S), \\ &cctx(C, S, write, O), foc(home, C). \end{aligned} \quad (B.5g)$$

$$\begin{aligned} uallow(home, su, S, exec, O) \leftarrow &= (email_W, O), org(work, O), \\ &cctx(C, S, exec, O), foc(home, C). \end{aligned} \quad (B.5h)$$

$$\begin{aligned} uallow(home, su, S, close, O) \leftarrow &= (email_W, O), org(work, O), \\ &cctx(C, S, close, O), foc(home, C). \end{aligned} \quad (B.5i)$$

Same as for the *Voip_W* app, CASS allows *Email_W* app to only operate through network channels. This enables it to receive emails. When a new email arrives, the *Notifier* app notifies Bob, in a similar way as it notifies him about a missed call from the *Voip_W* app. When Bob wants to open the *Email_W* app with the *Launcher_W*, his focus changes to the work social group, and CASS allows him to read the new email.

Appendix C

Evaluation of the Prototype

In this appendix, the operation of a prototype is described in more details by applying it in the use cases presented in Section 3.4. This prototype is the context-aware security system (CASS) based on the proposed conceptual model of security context and policy model and language, which was used during the evaluations in Chapter 4 and Chapter 5. The prototype operates as a state machine and makes security decisions upon various input events. Below, the security decisions made upon the actual input events according to the use cases are described, together with the contents of the set of continuous activities, A_C . However, the latter are described in a “relative” manner, meaning that only activities which are added or removed to this set upon each security decision are described. In order to facilitate the presentation of the flow of activities in the use cases, the activities of creation of the resources and execution of many of the applications are omitted. Also, some of the terms used throughout the tables are shortened (e.g., applications names) because of the space limitations. Table C.1 explains these shortenings.

Table C.1: Some terms and their shortened versions.

Term	Shortened term	Term	Shortened term
asst	Assistant app	nfc	NFCManager
cab	Cab app	ntfr	Notifier app
cam	Camera app	off	Office app
cch	camera channel	os	operating system
ch	network channel	P	public social group
chDr	network driver app	scr	screen channel
E	empty social group	scrDr	screen driver app
email	Email app	spk	speaker channel
fm	FileManager app	tch	touch channel
geo	geolocation channel	tchDr	touch driver app
H	home social group	traff	Traffic app
inst	Installer app	video	Video app
lnch	Launcher app	voip	Voip app
mic	microphone channel	W	work social group
netm	NetworkManager app		

Table C.2: Decision making in Use case 1, Scene 1.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, W, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, W, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, W, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, W, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, W, -1, -1)		allow
6	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, W, -1, -1)		allow
7	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, W, -1, -1)		allow
8	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, W, -1, -1)		allow
9	req(lnchW, execute, offW)	(lnchW, execute, offW), W, E, E, 0, 0)	->(lnchW, execute, offW)	allow
10	req(offW, open, scrW7)	(offW, open, scrW7), W, W, W, -1, -1)	->(offW, open, scrW7)	allow
11	req(scrDr, open, scrW7)	(scrDr, open, scrW7), W, W, W, -1, -1)	->(scrDr, open, scrW7)	allow
12	req(offW, open, tchW7)	(offW, open, tchW7), W, W, W, -1, -1)	->(offW, open, tchW7)	allow
13	req(tchDr, open, tchW7)	(tchDr, open, tchW7), W, W, W, -1, -1)	->(tchDr, open, tchW7)	allow
14	req(offW, write, scrW7)	(offW, write, scrW7), W, W, W, -1, -1)		allow
15	req(scrDr, read, scrW7)	(scrDr, read, scrW7), W, W, W, -1, -1)		allow
16	req(tchDr, write, tchW7)	(tchDr, write, tchW7), W, W, W, -1, -1)		allow
17	req(offW, read, tchW7)	(offW, read, tchW7), W, W, W, -1, -1)		allow
18	req(offW, open, docW1)	(offW, open, docW1), W, E, E, 0, 0)	->(offW, open, docW1)	allow
19	req(tchDr, write, tchW7)	(tchDr, write, tchW7), W, W, W, -1, -1)		allow
20	req(offW, read, tchW7)	(offW, read, tchW7), W, W, W, -1, -1)		allow
21	req(offW, write, docW1)	(offW, write, docW1), W, E, E, 0, 0)		allow
22	req(offW, read, docW1)	(offW, read, docW1), W, E, E, 0, 0)		allow
23	req(offW, write, scrW7)	(offW, write, scrW7), W, W, W, -1, -1)		allow
24	req(scrDr, read, scrW7)	(scrDr, read, scrW7), W, W, W, -1, -1)		allow
25	req(offW, open, chW7)	(offW, open, chW7), W, P, P, 1, 1)		deny
26	req(offW, close, docW1)	(offW, close, docW1), W, E, E, 0, 0)	<-(offW, open, docW1)	allow
27	req(offW, open, docH1)	(offW, open, docH1), W, E, E, 0, 0)		deny

Table C.3: Decision making in Use case 1, Scene 2.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, W, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, W, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, W, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, W, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, W, -1, -1)		allow
6	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, W, -1, -1)		allow
7	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, W, -1, -1)		allow
8	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, W, -1, -1)		allow
9	req(lnchW, execute, fmW)	(lnchW, execute, fmW), W, E, E, 0, 0)	->(lnchW, execute, fmW)	allow
10	req(fmW, open, scrW8)	(fmW, open, scrW8), W, W, W, -1, -1)	->(fmW, open, scrW8)	allow
11	req(scrDr, open, scrW8)	(scrDr, open, scrW8), W, W, W, -1, -1)	->(scrDr, open, scrW8)	allow
12	req(fmW, open, tchW8)	(fmW, open, tchW8), W, W, W, -1, -1)	->(fmW, open, tchW8)	allow
13	req(tchDr, open, tchW8)	(tchDr, open, tchW8), W, W, W, -1, -1)	->(tchDr, open, tchW8)	allow
14	req(fmW, write, scrW8)	(fmW, write, scrW8), W, W, W, -1, -1)		allow
15	req(scrDr, read, scrW8)	(scrDr, read, scrW8), W, W, W, -1, -1)		allow
16	req(tchDr, write, tchW8)	(tchDr, write, tchW8), W, W, W, -1, -1)		allow
17	req(fmW, read, tchW8)	(fmW, read, tchW8), W, W, W, -1, -1)		allow
18	req(fmW, open, chW8)	(fmW, open, chW8), W, W, W, 0, 0)	->(fmW, open, chW8)	allow
19	req(chDr, open, chW8)	(chDr, open, chW8), W, W, W, 0, 0)	->(chDr, open, chW8)	allow
20	req(chDr, write, chW8)	(chDr, write, chW8), W, W, W, 0, 0)		allow
21	req(fmW, read, chW8)	(fmW, read, chW8), W, W, W, 0, 0)		allow
22	req(fmW, write, chW8)	(fmW, write, chW8), W, W, W, 0, 0)		allow
23	req(chDr, read, chW8)	(chDr, read, chW8), W, W, W, 0, 0)		allow
24	req(lnchW, execute, netmW)	(lnchW, execute, netmW), W, E, E, 0, 0)	->(lnchW, execute, netmW)	allow
25	req(netmW, open, scrW3)	(netmW, open, scrW3), W, W, W, -1, -1)	->(netmW, open, scrW3)	allow
26	req(scrDr, open, scrW3)	(scrDr, open, scrW3), W, W, W, -1, -1)	->(scrDr, open, scrW3)	allow
27	req(netmW, open, tchW3)	(netmW, open, tchW3), W, W, W, -1, -1)	->(netmW, open, tchW3)	allow
28	req(tchDr, open, tchW3)	(tchDr, open, tchW3), W, W, W, -1, -1)	->(tchDr, open, tchW3)	allow
29	req(netmW, write, scrW3)	(netmW, write, scrW3), W, W, W, -1, -1)		allow
30	req(scrDr, read, scrW3)	(scrDr, read, scrW3), W, W, W, -1, -1)		allow
31	req(tchDr, write, tchW3)	(tchDr, write, tchW3), W, W, W, -1, -1)		allow
32	req(netmW, read, tchW3)	(netmW, read, tchW3), W, W, W, -1, -1)		allow
33	req(netmW, open, chW3)	(netmW, open, chW3), W, W, W, 0, 0)	->(netmW, open, chW3)	allow
34	req(chDr, open, chW3)	(chDr, open, chW3), W, W, W, 0, 0)	->(chDr, open, chW3)	allow
35	req(netmW, write, chW3)	(netmW, write, chW3), W, W, W, 0, 0)		allow
36	req(chDr, read, chW3)	(chDr, read, chW3), W, W, W, 0, 0)		allow
37	req(chDr, write, chW3)	(chDr, write, chW3), W, W, W, 0, 0)		allow
38	req(netmW, read, chW3)	(netmW, read, chW3), W, W, W, 0, 0)		allow
39	req(lnchW, execute, instW)	(lnchW, execute, instW), W, E, E, 0, 0)	->(lnchW, execute, instW)	allow
40	req(instW, open, scrW2)	(instW, open, scrW2), W, W, W, -1, -1)	->(instW, open, scrW2)	allow
41	req(scrDr, open, scrW2)	(scrDr, open, scrW2), W, W, W, -1, -1)	->(scrDr, open, scrW2)	allow
42	req(instW, open, tchW2)	(instW, open, tchW2), W, W, W, -1, -1)	->(instW, open, tchW2)	allow
43	req(tchDr, open, tchW2)	(tchDr, open, tchW2), W, W, W, -1, -1)	->(tchDr, open, tchW2)	allow
44	req(instW, write, scrW2)	(instW, write, scrW2), W, W, W, -1, -1)		allow
45	req(scrDr, read, scrW2)	(scrDr, read, scrW2), W, W, W, -1, -1)		allow
46	req(tchDr, write, tchW2)	(tchDr, write, tchW2), W, W, W, -1, -1)		allow
47	req(instW, read, tchW2)	(instW, read, tchW2), W, W, W, -1, -1)		allow
48	req(instW, open, chW2)	(instW, open, chW2), W, W, W, 0, 0)	->(instW, open, chW2)	allow
49	req(chDr, open, chW2)	(chDr, open, chW2), W, W, W, 0, 0)	->(chDr, open, chW2)	allow
50	req(instW, write, chW2)	(instW, write, chW2), W, W, W, 0, 0)		allow
51	req(chDr, read, chW2)	(chDr, read, chW2), W, W, W, 0, 0)		allow
52	req(chDr, write, chW2)	(chDr, write, chW2), W, W, W, 0, 0)		allow
53	req(instW, read, chW2)	(instW, read, chW2), W, W, W, 0, 0)		allow

Table C.4: Decision making in Use case 1, Scene 3.

ID	Input Event	Activity Context	Continuous activity	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, W, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, W, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, W, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, W, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, W, -1, -1)		allow
6	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, W, -1, -1)		allow
7	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, W, -1, -1)		allow
8	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, W, -1, -1)		allow
9	req(lnchH, open, scrH1)	(lnchH, open, scrH1), W, W, W, -1, -1)		deny
focus change: W->H				
10	chg(lnchW, open, scrW1)	(lnchW, open, scrW1), H, H, W, -1, -1)	<-(lnchW, open, scrW1)	terminate
11	chg(lnchW, open, tchW1)	(lnchW, open, tchW1), H, H, W, -1, -1)	<-(lnchW, open, tchW1)	terminate
12	chg(scrDr, open, scrW1)	(scrDr, open, scrW1), H, H, W, -1, -1)	<-(scrDr, open, scrW1)	terminate
13	chg(tchDr, open, tchW1)	(tchDr, open, tchW1), H, H, W, -1, -1)	<-(tchDr, open, tchW1)	terminate
14	req(lnchH, open, scrH1)	(lnchH, open, scrH1), H, H, W, -1, -1)		deny
15	req(lnchW, open, scrW1)	(lnchW, open, scrW1), H, H, W, -1, -1)		deny
focus change: H->W				
16	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, W, -1, -1)	->(lnchW, open, scrW1)	allow
17	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, W, -1, -1)	->(scrDr, open, scrW1)	allow
18	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, W, -1, -1)	->(lnchW, open, tchW1)	allow
19	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, W, -1, -1)	->(tchDr, open, tchW1)	allow
20	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, W, -1, -1)		allow
21	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, W, -1, -1)		allow
22	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, W, -1, -1)		allow
23	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, W, -1, -1)		allow
24	req(lnchW, execute, nfcW)	(lnchW, execute, nfcW), W, E, E, 0, 0)	->(lnchW, execute, nfcW)	allow
25	req(nfcW, open, chW4)	(nfcW, open, chW4), W, W, W, -1, -1)	->(nfcW, open, chW4)	allow
26	req(chDr, open, chW4)	(chDr, open, chW4), W, W, W, -1, -1)	->(chDr, open, chW4)	allow
27	req(chDr, write, chW4)	(chDr, write, chW4), W, W, W, -1, -1)		allow
28	req(nfcW, read, chW4)	(nfcW, read, chW4), W, W, W, -1, -1)		allow
29	req(nfcW, write, chW4)	(nfcW, write, chW4), W, W, W, -1, -1)		allow
30	req(chDr, read, chW4)	(chDr, read, chW4), W, W, W, -1, -1)		allow
focus change: W->P				
31	chg(lnchW, open, scrW1)	(lnchW, open, scrW1), P, P, W, -1, -1)	<-(lnchW, open, scrW1)	terminate
32	chg(lnchW, open, tchW1)	(lnchW, open, tchW1), P, P, W, -1, -1)	<-(lnchW, open, tchW1)	terminate
33	chg(lnchW, execute, nfcW)	(lnchW, execute, nfcW), P, E, E, 0, 0)	<-(lnchW, execute, nfcW)	terminate
34	chg(nfcW, open, chW4)	(nfcW, open, chW4), P, P, W, -1, -1)	<-(nfcW, open, chW4)	terminate
35	chg(scrDr, open, scrW1)	(scrDr, open, scrW1), P, P, W, -1, -1)	<-(scrDr, open, scrW1)	terminate
36	chg(tchDr, open, tchW1)	(tchDr, open, tchW1), P, P, W, -1, -1)	<-(tchDr, open, tchW1)	terminate
37	chg(chDr, open, chW4)	(chDr, open, chW4), P, P, W, -1, -1)	<-(chDr, open, chW4)	terminate
38	req(lnchH, open, scrH1)	(lnchH, open, scrH1), P, P, P, -1, -1)		deny
focus change: P->H				
39	req(lnchH, open, scrH1)	(lnchH, open, scrH1), H, H, P, -1, -1)	->(lnchH, open, scrH1)	allow
40	req(scrDr, open, scrH1)	(scrDr, open, scrH1), H, H, P, -1, -1)	->(scrDr, open, scrH1)	allow
41	req(lnchH, open, tchH1)	(lnchH, open, tchH1), H, H, P, -1, -1)	->(lnchH, open, tchH1)	allow
42	req(tchDr, open, tchH1)	(tchDr, open, tchH1), H, H, P, -1, -1)	->(tchDr, open, tchH1)	allow
43	req(lnchH, write, scrH1)	(lnchH, write, scrH1), H, H, P, -1, -1)		allow
44	req(scrDr, read, scrH1)	(scrDr, read, scrH1), H, H, P, -1, -1)		allow
45	req(tchDr, write, tchH1)	(tchDr, write, tchH1), H, H, P, -1, -1)		allow
46	req(lnchH, read, tchH1)	(lnchH, read, tchH1), H, H, P, -1, -1)		allow
47	req(lnchH, execute, netmH)	(lnchH, execute, netmH), H, E, E, 0, 0)	->(lnchH, execute, netmH)	allow
48	req(netmH, open, chH4)	(netmH, open, chH4), H, P, P, 0, 0)	->(netmH, open, chH4)	allow
49	req(chDr, open, chH4)	(chDr, open, chH4), H, P, P, 0, 0)	->(chDr, open, chH4)	allow
50	req(netmH, write, chH4)	(netmH, write, chH4), H, P, P, 0, 0)		allow
51	req(chDr, read, chH4)	(chDr, read, chH4), H, P, P, 0, 0)		allow
52	req(chDr, write, chH4)	(chDr, write, chH4), H, P, P, 0, 0)		allow
53	req(netmH, read, chH4)	(netmH, read, chH4), H, P, P, 0, 0)		allow
54	req(lnchH, write, scrH1)	(lnchH, write, scrH1), H, H, P, -1, -1)		allow
55	req(scrDr, read, scrH1)	(scrDr, read, scrH1), H, H, P, -1, -1)		allow
56	req(tchDr, write, tchH1)	(tchDr, write, tchH1), H, H, P, -1, -1)		allow
57	req(lnchH, read, tchH1)	(lnchH, read, tchH1), H, H, P, -1, -1)		allow
58	req(lnchH, execute, voipH)	(lnchH, execute, voipH), H, E, E, 0, 0)	->(lnchH, execute, voipH)	allow
59	req(voipH, open, scrH5)	(voipH, open, scrH5), H, H, P, -1, -1)	->(voipH, open, scrH5)	allow
60	req(scrDr, open, scrH5)	(scrDr, open, scrH5), H, H, P, -1, -1)	->(scrDr, open, scrH5)	allow
61	req(voipH, open, tchH5)	(voipH, open, tchH5), H, H, P, -1, -1)	->(voipH, open, tchH5)	allow
62	req(tchDr, open, tchH5)	(tchDr, open, tchH5), H, H, P, -1, -1)	->(tchDr, open, tchH5)	allow
63	req(voipH, open, chH5)	(voipH, open, chH5), H, H, P, 1, 1)	->(voipH, open, chH5)	allow
64	req(chDr, open, chH5)	(chDr, open, chH5), H, H, P, 1, 1)	->(chDr, open, chH5)	allow
65	req(voipH, open, micH5)	(voipH, open, micH5), H, P, P, -1, -1)	->(voipH, open, micH5)	allow
66	req(micDr, open, micH5)	(micDr, open, micH5), H, P, P, -1, -1)	->(micDr, open, micH5)	allow
67	req(voipH, open, spkH5)	(voipH, open, spkH5), H, H, P, -1, -1)	->(voipH, open, spkH5)	allow
68	req(spkrDr, open, spkH5)	(spkrDr, open, spkH5), H, H, P, -1, -1)	->(spkrDr, open, spkH5)	allow
69	req(chDr, write, chH5)	(chDr, write, chH5), H, H, P, 1, 1)		allow
70	req(voipH, read, chH5)	(voipH, read, chH5), H, H, P, 1, 1)		allow
71	req(voipH, write, spkH5)	(voipH, write, spkH5), H, H, P, -1, -1)		allow
72	req(spkrDr, read, spkH5)	(spkrDr, read, spkH5), H, H, P, -1, -1)		allow
73	req(micDr, write, micH5)	(micDr, write, micH5), H, P, P, -1, -1)		allow
74	req(voipH, read, micH5)	(voipH, read, micH5), H, P, P, -1, -1)		allow
75	req(voipH, write, chH5)	(voipH, write, chH5), H, H, P, 1, 1)		allow
76	req(chDr, read, chH5)	(chDr, read, chH5), H, H, P, 1, 1)		allow

Table C.5: Decision making in Use case 1, Scene 4.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchP, open, scrP1)	(lnchP, open, scrP1), P, P, P, -1, -1)	->(lnchP, open, scrP1)	allow
2	req(scrDr, open, scrP1)	(scrDr, open, scrP1), P, P, P, -1, -1)	->(scrDr, open, scrP1)	allow
3	req(lnchP, open, tchP1)	(lnchP, open, tchP1), P, P, P, -1, -1)	->(lnchP, open, tchP1)	allow
4	req(tchDr, open, tchP1)	(tchDr, open, tchP1), P, P, P, -1, -1)	->(tchDr, open, tchP1)	allow
5	req(lnchP, write, scrP1)	(lnchP, write, scrP1), P, P, P, -1, -1)		allow
6	req(scrDr, read, scrP1)	(scrDr, read, scrP1), P, P, P, -1, -1)		allow
7	req(tchDr, write, tchP1)	(tchDr, write, tchP1), P, P, P, -1, -1)		allow
8	req(lnchP, read, tchP1)	(lnchP, read, tchP1), P, P, P, -1, -1)		allow
9	req(lnchP, execute, netmP)	(lnchP, execute, netmP), P, E, E, 0, 0)	->(lnchP, execute, netmP)	allow
10	req(netmP, open, scrP3)	(netmP, open, scrP3), P, P, P, -1, -1)	->(netmP, open, scrP3)	allow
11	req(scrDr, open, scrP3)	(scrDr, open, scrP3), P, P, P, -1, -1)	->(scrDr, open, scrP3)	allow
12	req(netmP, open, tchP3)	(netmP, open, tchP3), P, P, P, -1, -1)	->(netmP, open, tchP3)	allow
13	req(tchDr, open, tchP3)	(tchDr, open, tchP3), P, P, P, -1, -1)	->(tchDr, open, tchP3)	allow
14	req(netmP, write, scrP3)	(netmP, write, scrP3), P, P, P, -1, -1)		allow
15	req(scrDr, read, scrP3)	(scrDr, read, scrP3), P, P, P, -1, -1)		allow
16	req(tchDr, write, tchP3)	(tchDr, write, tchP3), P, P, P, -1, -1)		allow
17	req(netmP, read, tchP3)	(netmP, read, tchP3), P, P, P, -1, -1)		allow
18	req(netmP, open, chP3)	(netmP, open, chP3), P, P, P, 0, 0)	->(netmP, open, chP3)	allow
19	req(chDr, open, chP3)	(chDr, open, chP3), P, P, P, 0, 0)	->(chDr, open, chP3)	allow
20	req(netmP, write, chP3)	(netmP, write, chP3), P, P, P, 0, 0)		allow
21	req(chDr, read, chP3)	(chDr, read, chP3), P, P, P, 0, 0)		allow
22	req(chDr, write, chP3)	(chDr, write, chP3), P, P, P, 0, 0)		allow
23	req(netmP, read, chP3)	(netmP, read, chP3), P, P, P, 0, 0)		allow
24	req(lnchP, write, scrP1)	(lnchP, write, scrP1), P, P, P, -1, -1)		allow
25	req(scrDr, read, scrP1)	(scrDr, read, scrP1), P, P, P, -1, -1)		allow
26	req(tchDr, write, tchP1)	(tchDr, write, tchP1), P, P, P, -1, -1)		allow
27	req(lnchP, read, tchP1)	(lnchP, read, tchP1), P, P, P, -1, -1)		allow
28	req(lnchP, execute, asstP)	(lnchP, execute, asstP), P, E, E, 0, 0)	->(lnchP, execute, asstP)	allow
29	req(asstP, open, scrP7)	(asstP, open, scrP7), P, P, P, -1, -1)	->(asstP, open, scrP7)	allow
30	req(scrDr, open, scrP7)	(scrDr, open, scrP7), P, P, P, -1, -1)	->(scrDr, open, scrP7)	allow
31	req(asstP, open, tchP7)	(asstP, open, tchP7), P, P, P, -1, -1)	->(asstP, open, tchP7)	allow
32	req(tchDr, open, tchP7)	(tchDr, open, tchP7), P, P, P, -1, -1)	->(tchDr, open, tchP7)	allow
33	req(asstP, open, micP7)	(asstP, open, micP7), P, P, P, -1, -1)	->(asstP, open, micP7)	allow
34	req(micDr, open, micP7)	(micDr, open, micP7), P, P, P, -1, -1)	->(micDr, open, micP7)	allow
35	req(asstP, open, spkP7)	(asstP, open, spkP7), P, P, P, -1, -1)	->(asstP, open, spkP7)	allow
36	req(spkDr, open, spkP7)	(spkDr, open, spkP7), P, P, P, -1, -1)	->(spkDr, open, spkP7)	allow
37	req(asstP, open, chP7)	(asstP, open, chP7), P, P, P, 1, 1)	->(asstP, open, chP7)	allow
38	req(chDr, open, chP7)	(chDr, open, chP7), P, P, P, 1, 1)	->(chDr, open, chP7)	allow
39	req(asstP, write, scrP7)	(asstP, write, scrP7), P, P, P, -1, -1)		allow
40	req(scrDr, read, scrP7)	(scrDr, read, scrP7), P, P, P, -1, -1)		allow
41	req(micDr, write, micP7)	(micDr, write, micP7), P, P, P, -1, -1)		allow
42	req(asstP, read, micP7)	(asstP, read, micP7), P, P, P, -1, -1)		allow
43	req(asstP, write, chP7)	(asstP, write, chP7), P, P, P, 1, 1)		allow
44	req(chDr, read, chP7)	(chDr, read, chP7), P, P, P, 1, 1)		allow
45	req(chDr, write, chP7)	(chDr, write, chP7), P, P, P, 1, 1)		allow
46	req(asstP, read, chP7)	(asstP, read, chP7), P, P, P, 1, 1)		allow
47	req(asstP, write, spkP7)	(asstP, write, spkP7), P, P, P, -1, -1)		allow
48	req(spkDr, read, spkP7)	(spkDr, read, spkP7), P, P, P, -1, -1)		allow
49	req(asstP, open, docW1)	(asstP, open, docW1), P, E, E, 0, 0)		deny
50	req(lnchP, write, scrP1)	(lnchP, write, scrP1), P, P, P, -1, -1)		allow
51	req(scrDr, read, scrP1)	(scrDr, read, scrP1), P, P, P, -1, -1)		allow
52	req(tchDr, write, tchP1)	(tchDr, write, tchP1), P, P, P, -1, -1)		allow
53	req(lnchP, read, tchP1)	(lnchP, read, tchP1), P, P, P, -1, -1)		allow
54	req(lnchP, execute, nfcP)	(lnchP, execute, nfcP), P, E, E, 0, 0)	->(lnchP, execute, nfcP)	allow
55	req(nfcP, open, chP4)	(nfcP, open, chP4), P, H, P, 0, 0)		deny
focus change: P->H				
56	chg(lnchP, open, scrP1)	(lnchP, open, scrP1), H, H, P, -1, -1)	<-(lnchP, open, scrP1)	terminate
57	chg(lnchP, open, tchP1)	(lnchP, open, tchP1), H, H, P, -1, -1)	<-(lnchP, open, tchP1)	terminate
58	chg(lnchP, execute, netmP)	(lnchP, execute, netmP), H, E, E, 0, 0)	<-(lnchP, execute, netmP)	terminate
59	chg(netmP, open, scrP3)	(netmP, open, scrP3), H, H, P, -1, -1)	<-(netmP, open, scrP3)	terminate
60	chg(netmP, open, tchP3)	(netmP, open, tchP3), H, H, P, -1, -1)	<-(netmP, open, tchP3)	terminate
61	chg(netmP, open, chP3)	(netmP, open, chP3), H, P, P, 0, 0)	<-(netmP, open, chP3)	terminate
62	chg(lnchP, execute, asstP)	(lnchP, execute, asstP), H, E, E, 0, 0)	<-(lnchP, execute, asstP)	terminate
63	chg(asstP, open, scrP7)	(asstP, open, scrP7), H, H, P, -1, -1)	<-(asstP, open, scrP7)	terminate
64	chg(asstP, open, tchP7)	(asstP, open, tchP7), H, H, P, -1, -1)	<-(asstP, open, tchP7)	terminate
65	chg(asstP, open, micP7)	(asstP, open, micP7), H, H, P, -1, -1)	<-(asstP, open, micP7)	terminate
66	chg(asstP, open, spkP7)	(asstP, open, spkP7), H, H, P, -1, -1)	<-(asstP, open, spkP7)	terminate
67	chg(asstP, open, chP7)	(asstP, open, chP7), H, P, P, 1, 1)	<-(asstP, open, chP7)	terminate
68	chg(lnchP, execute, nfcP)	(lnchP, execute, nfcP), H, E, E, 0, 0)	<-(lnchP, execute, nfcP)	terminate
69	chg(scrDr, open, scrP1)	(scrDr, open, scrP1), H, H, P, -1, -1)	<-(scrDr, open, scrP1)	terminate
70	chg(tchDr, open, tchP1)	(tchDr, open, tchP1), H, H, P, -1, -1)	<-(tchDr, open, tchP1)	terminate
71	chg(scrDr, open, scrP3)	(scrDr, open, scrP3), H, H, P, -1, -1)	<-(scrDr, open, scrP3)	terminate
72	chg(tchDr, open, tchP3)	(tchDr, open, tchP3), H, H, P, -1, -1)	<-(tchDr, open, tchP3)	terminate
73	chg(chDr, open, chP3)	(chDr, open, chP3), H, P, P, 0, 0)	<-(chDr, open, chP3)	terminate
74	chg(scrDr, open, scrP7)	(scrDr, open, scrP7), H, H, P, -1, -1)	<-(scrDr, open, scrP7)	terminate
75	chg(tchDr, open, tchP7)	(tchDr, open, tchP7), H, H, P, -1, -1)	<-(tchDr, open, tchP7)	terminate
76	chg(micDr, open, micP7)	(micDr, open, micP7), H, H, P, -1, -1)	<-(micDr, open, micP7)	terminate
77	chg(spkDr, open, spkP7)	(spkDr, open, spkP7), H, H, P, -1, -1)	<-(spkDr, open, spkP7)	terminate
78	chg(chDr, open, chP7)	(chDr, open, chP7), H, P, P, 1, 1)	<-(chDr, open, chP7)	terminate
79	req(lnchH, open, scrH1)	(lnchH, open, scrH1), H, H, P, -1, -1)	->(lnchH, open, scrH1)	allow
80	req(scrDr, open, scrH1)	(scrDr, open, scrH1), H, H, P, -1, -1)	->(scrDr, open, scrH1)	allow
81	req(lnchH, open, tchH1)	(lnchH, open, tchH1), H, H, P, -1, -1)	->(lnchH, open, tchH1)	allow
82	req(tchDr, open, tchH1)	(tchDr, open, tchH1), H, H, P, -1, -1)	->(tchDr, open, tchH1)	allow
83	req(lnchH, write, scrH1)	(lnchH, write, scrH1), H, H, P, -1, -1)		allow
84	req(scrDr, read, scrH1)	(scrDr, read, scrH1), H, H, P, -1, -1)		allow
85	req(tchDr, write, tchH1)	(tchDr, write, tchH1), H, H, P, -1, -1)		allow
86	req(lnchH, read, tchH1)	(lnchH, read, tchH1), H, H, P, -1, -1)		allow
87	req(lnchH, execute, nfcH)	(lnchH, execute, nfcH), H, E, E, 0, 0)	->(lnchH, execute, nfcH)	allow
88	req(nfcH, open, chH4)	(nfcH, open, chH4), H, H, P, 0, 0)	->(nfcH, open, chH4)	allow
89	req(chDr, open, chH4)	(chDr, open, chH4), H, H, P, 0, 0)	->(chDr, open, chH4)	allow
90	req(chDr, write, chH4)	(chDr, write, chH4), H, H, P, 0, 0)		allow

Continued on next page

Table C.5 – Continued from previous page

ID	Input Event	Activity Context	Continuous Activities	Decision
91	req(nfcH, read, chH4)	(nfcH, read, chH4), H, H, P, 0, 0)		allow
92	req(nfcH, write, chH4)	(nfcH, write, chH4), H, H, P, 0, 0)		allow
93	req(chDr, read, chH4)	(chDr, read, chH4), H, H, P, 0, 0)		allow
94	req(lnchH, write, scrH1)	(lnchH, write, scrH1), H, H, H, -1, -1)		allow
95	req(scrDr, read, scrH1)	(scrDr, read, scrH1), H, H, H, -1, -1)		allow
96	req(tchDr, write, tchH1)	(tchDr, write, tchH1), H, H, H, -1, -1)		allow
97	req(lnchH, read, tchH1)	(lnchH, read, tchH1), H, H, H, -1, -1)		allow
98	req(lnchH, execute, playH)	(lnchH, execute, playH), H, E, E, 0, 0)	->(lnchH, execute, playH)	allow
99	req(playH, open, spkH7)	(playH, open, spkH7), H, H, H, -1, -1)	->(playH, open, spkH7)	allow
100	req(playH, open, mp3H1)	(playH, open, mp3H1), H, E, E, 0, 0)	->(playH, open, mp3H1)	allow
101	req(playH, write, spkH7)	(playH, write, spkH7), H, H, H, -1, -1)		allow
102	req(lnchW, open, scrW1)	(lnchW, open, scrW1), H, H, H, -1, -1)		deny
focus change: H->W				
103	chg(lnchH, open, scrH1)	(lnchH, open, scrH1), W, W, P, -1, -1)	<-(lnchH, open, scrH1)	terminate
104	chg(lnchH, open, tchH1)	(lnchH, open, tchH1), W, W, P, -1, -1)	<-(lnchH, open, tchH1)	terminate
105	chg(lnchH, execute, nfcH)	(lnchH, execute, nfcH), W, E, E, 0, 0)	<-(lnchH, execute, nfcH)	terminate
106	chg(nfcH, open, chH4)	(nfcH, open, chH4), W, H, P, 0, 0)	<-(nfcH, open, chH4)	terminate
107	chg(lnchH, execute, playH)	(lnchH, execute, playH), W, E, E, 0, 0)	<-(lnchH, execute, playH)	terminate
108	chg(playH, open, spkH7)	(playH, open, spkH7), W, W, H, -1, -1)	<-(playH, open, spkH7)	terminate
109	chg(playH, open, mp3H1)	(playH, open, mp3H1), W, E, E, 0, 0)	<-(playH, open, mp3H1)	terminate
110	chg(scrDr, open, scrH1)	(scrDr, open, scrH1), W, W, P, -1, -1)	<-(scrDr, open, scrH1)	terminate
111	chg(tchDr, open, tchH1)	(tchDr, open, tchH1), W, W, P, -1, -1)	<-(tchDr, open, tchH1)	terminate
112	chg(chDr, open, chH4)	(chDr, open, chH4), W, H, P, 0, 0)	<-(chDr, open, chH4)	terminate
113	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, H, -1, -1)		deny

Table C.6: Decision making in Use case 2, Scene 1.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchH, open, scrH1)	(lnchH, open, scrH1), H, H, H, -1, -1)	->(lnchH, open, scrH1)	allow
2	req(scrDr, open, scrH1)	(scrDr, open, scrH1), H, H, H, -1, -1)	->(scrDr, open, scrH1)	allow
3	req(lnchH, open, tchH1)	(lnchH, open, tchH1), H, H, H, -1, -1)	->(lnchH, open, tchH1)	allow
4	req(tchDr, open, tchH1)	(tchDr, open, tchH1), H, H, H, -1, -1)	->(tchDr, open, tchH1)	allow
5	req(lnchW, open, scrW1)	(lnchW, open, scrW1), H, H, H, -1, -1)	->(lnchW, open, scrW1)	deny
focus change: H->W				
6	chg(lnchH, open, scrH1)	(lnchH, open, scrH1), W, W, H, -1, -1)	<-(lnchH, open, scrH1)	terminate
7	chg(lnchH, open, tchH1)	(lnchH, open, tchH1), W, W, H, -1, -1)	<-(lnchH, open, tchH1)	terminate
8	chg(scrDr, open, scrH1)	(scrDr, open, scrH1), W, W, H, -1, -1)	<-(scrDr, open, scrH1)	terminate
9	chg(tchDr, open, tchH1)	(tchDr, open, tchH1), W, W, H, -1, -1)	<-(tchDr, open, tchH1)	terminate
10	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, H, -1, -1)	->(lnchW, open, scrW1)	allow
11	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, H, -1, -1)	->(scrDr, open, scrW1)	allow
12	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, H, -1, -1)	->(lnchW, open, tchW1)	allow
13	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, H, -1, -1)	->(tchDr, open, tchW1)	allow
14	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, H, -1, -1)	->(lnchW, write, scrW1)	allow
15	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, H, -1, -1)	->(scrDr, read, scrW1)	allow
16	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, H, -1, -1)	->(tchDr, write, tchW1)	allow
17	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, H, -1, -1)	->(lnchW, read, tchW1)	allow
18	req(lnchW, execute, offW)	(lnchW, execute, offW), W, E, E, 0, 0)	->(lnchW, execute, offW)	allow
19	req(offW, open, scrW7)	(offW, open, scrW7), W, W, H, -1, -1)	->(offW, open, scrW7)	allow
20	req(scrDr, open, scrW7)	(scrDr, open, scrW7), W, W, H, -1, -1)	->(scrDr, open, scrW7)	allow
21	req(offW, open, tchW7)	(offW, open, tchW7), W, W, H, -1, -1)	->(offW, open, tchW7)	allow
22	req(tchDr, open, tchW7)	(tchDr, open, tchW7), W, W, H, -1, -1)	->(tchDr, open, tchW7)	allow
23	req(offW, write, scrW7)	(offW, write, scrW7), W, W, H, -1, -1)	->(offW, write, scrW7)	allow
24	req(scrDr, read, scrW7)	(scrDr, read, scrW7), W, W, H, -1, -1)	->(scrDr, read, scrW7)	allow
25	req(tchDr, write, tchW7)	(tchDr, write, tchW7), W, W, H, -1, -1)	->(tchDr, write, tchW7)	allow
26	req(offW, read, tchW7)	(offW, read, tchW7), W, W, H, -1, -1)	->(offW, read, tchW7)	allow
27	req(offW, open, docW1)	(offW, open, docW1), W, E, E, 0, 0)	->(offW, open, docW1)	allow
28	req(offW, read, docW1)	(offW, read, docW1), W, E, E, 0, 0)	->(offW, read, docW1)	allow
29	req(offW, write, scrW7)	(offW, write, scrW7), W, W, H, -1, -1)	->(offW, write, scrW7)	allow
30	req(scrDr, read, scrW7)	(scrDr, read, scrW7), W, W, H, -1, -1)	->(scrDr, read, scrW7)	allow
31	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, H, -1, -1)	->(lnchW, write, scrW1)	allow
32	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, H, -1, -1)	->(scrDr, read, scrW1)	allow
33	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, H, -1, -1)	->(tchDr, write, tchW1)	allow
34	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, H, -1, -1)	->(lnchW, read, tchW1)	allow
35	req(lnchW, execute, camW)	(lnchW, execute, camW), W, E, E, 0, 0)	->(lnchW, execute, camW)	allow
36	req(camW, open, scrW6)	(camW, open, scrW6), W, W, H, -1, -1)	->(camW, open, scrW6)	allow
37	req(scrDr, open, scrW6)	(scrDr, open, scrW6), W, W, H, -1, -1)	->(scrDr, open, scrW6)	allow
38	req(camW, open, tchW6)	(camW, open, tchW6), W, W, H, -1, -1)	->(camW, open, tchW6)	allow
39	req(tchDr, open, tchW6)	(tchDr, open, tchW6), W, W, H, -1, -1)	->(tchDr, open, tchW6)	allow
40	req(camW, open, cchW6)	(camW, open, cchW6), W, W, H, -1, -1)	->(camW, open, cchW6)	allow
41	req(camDr, open, cchW6)	(camDr, open, cchW6), W, W, H, -1, -1)	->(camDr, open, cchW6)	allow
42	req(camW, write, scrW6)	(camW, write, scrW6), W, W, H, -1, -1)	->(camW, write, scrW6)	allow
43	req(scrDr, read, scrW6)	(scrDr, read, scrW6), W, W, H, -1, -1)	->(scrDr, read, scrW6)	allow
44	req(tchDr, write, tchW6)	(tchDr, write, tchW6), W, W, H, -1, -1)	->(tchDr, write, tchW6)	allow
45	req(camW, read, tchW6)	(camW, read, tchW6), W, W, H, -1, -1)	->(camW, read, tchW6)	allow
46	req(camW, read, cchW6)	(camW, read, cchW6), W, W, H, -1, -1)	->(camW, read, cchW6)	deny
47	req(lnchH, open, scrH1)	(lnchH, open, scrH1), W, W, H, -1, -1)	->(lnchH, open, scrH1)	deny
focus change: W->H				
48	chg(lnchW, open, scrW1)	(lnchW, open, scrW1), H, H, H, -1, -1)	<-(lnchW, open, scrW1)	terminate
49	chg(lnchW, open, tchW1)	(lnchW, open, tchW1), H, H, H, -1, -1)	<-(lnchW, open, tchW1)	terminate
50	chg(lnchW, execute, offW)	(lnchW, execute, offW), H, E, E, 0, 0)	<-(lnchW, execute, offW)	terminate
51	chg(offW, open, scrW7)	(offW, open, scrW7), H, H, H, -1, -1)	<-(offW, open, scrW7)	terminate
52	chg(offW, open, tchW7)	(offW, open, tchW7), H, H, H, -1, -1)	<-(offW, open, tchW7)	terminate
53	chg(offW, open, docW1)	(offW, open, docW1), H, E, E, 0, 0)	<-(offW, open, docW1)	terminate
54	chg(lnchW, execute, camW)	(lnchW, execute, camW), H, E, E, 0, 0)	<-(lnchW, execute, camW)	terminate
55	chg(camW, open, scrW6)	(camW, open, scrW6), H, H, H, -1, -1)	<-(camW, open, scrW6)	terminate
56	chg(camW, open, tchW6)	(camW, open, tchW6), H, H, H, -1, -1)	<-(camW, open, tchW6)	terminate
57	chg(camW, open, cchW6)	(camW, open, cchW6), H, H, H, -1, -1)	<-(camW, open, cchW6)	terminate
58	chg(scrDr, open, scrW1)	(scrDr, open, scrW1), H, H, H, -1, -1)	<-(scrDr, open, scrW1)	terminate
59	chg(tchDr, open, tchW1)	(tchDr, open, tchW1), H, H, H, -1, -1)	<-(tchDr, open, tchW1)	terminate
60	chg(scrDr, open, scrW7)	(scrDr, open, scrW7), H, H, H, -1, -1)	<-(scrDr, open, scrW7)	terminate
61	chg(tchDr, open, tchW7)	(tchDr, open, tchW7), H, H, H, -1, -1)	<-(tchDr, open, tchW7)	terminate
62	chg(scrDr, open, scrW6)	(scrDr, open, scrW6), H, H, H, -1, -1)	<-(scrDr, open, scrW6)	terminate
63	chg(tchDr, open, tchW6)	(tchDr, open, tchW6), H, H, H, -1, -1)	<-(tchDr, open, tchW6)	terminate
64	chg(camDr, open, cchW6)	(camDr, open, cchW6), H, H, H, -1, -1)	<-(camDr, open, cchW6)	terminate
65	req(lnchH, open, scrH1)	(lnchH, open, scrH1), H, H, H, -1, -1)	->(lnchH, open, scrH1)	allow
66	req(scrDr, open, scrH1)	(scrDr, open, scrH1), H, H, H, -1, -1)	->(scrDr, open, scrH1)	allow
67	req(lnchH, open, tchH1)	(lnchH, open, tchH1), H, H, H, -1, -1)	->(lnchH, open, tchH1)	allow
68	req(tchDr, open, tchH1)	(tchDr, open, tchH1), H, H, H, -1, -1)	->(tchDr, open, tchH1)	allow
69	req(lnchH, write, scrH1)	(lnchH, write, scrH1), H, H, H, -1, -1)	->(lnchH, write, scrH1)	allow
70	req(scrDr, read, scrH1)	(scrDr, read, scrH1), H, H, H, -1, -1)	->(scrDr, read, scrH1)	allow
71	req(tchDr, write, tchH1)	(tchDr, write, tchH1), H, H, H, -1, -1)	->(tchDr, write, tchH1)	allow
72	req(lnchH, read, tchH1)	(lnchH, read, tchH1), H, H, H, -1, -1)	->(lnchH, read, tchH1)	allow
73	req(lnchH, execute, camH)	(lnchH, execute, camH), H, E, E, 0, 0)	->(lnchH, execute, camH)	allow
74	req(camH, open, scrH6)	(camH, open, scrH6), H, H, H, -1, -1)	->(camH, open, scrH6)	allow
75	req(scrDr, open, scrH6)	(scrDr, open, scrH6), H, H, H, -1, -1)	->(scrDr, open, scrH6)	allow
76	req(camH, open, tchH6)	(camH, open, tchH6), H, H, H, -1, -1)	->(camH, open, tchH6)	allow
77	req(tchDr, open, tchH6)	(tchDr, open, tchH6), H, H, H, -1, -1)	->(tchDr, open, tchH6)	allow
78	req(camH, open, cchH6)	(camH, open, cchH6), H, H, H, -1, -1)	->(camH, open, cchH6)	allow
79	req(camDr, open, cchH6)	(camDr, open, cchH6), H, H, H, -1, -1)	->(camDr, open, cchH6)	allow
80	req(camH, write, scrH6)	(camH, write, scrH6), H, H, H, -1, -1)	->(camH, write, scrH6)	allow
81	req(scrDr, read, scrH6)	(scrDr, read, scrH6), H, H, H, -1, -1)	->(scrDr, read, scrH6)	allow
82	req(tchDr, write, tchH6)	(tchDr, write, tchH6), H, H, H, -1, -1)	->(tchDr, write, tchH6)	allow
83	req(camH, read, tchH6)	(camH, read, tchH6), H, H, H, -1, -1)	->(camH, read, tchH6)	allow
84	req(camDr, write, cchH6)	(camDr, write, cchH6), H, H, H, -1, -1)	->(camDr, write, cchH6)	allow
85	req(camH, read, cchH6)	(camH, read, cchH6), H, H, H, -1, -1)	->(camH, read, cchH6)	allow
86	req(camH, open, imgH1)	(camH, open, imgH1), H, E, E, 0, 0)	->(camH, open, imgH1)	allow
87	req(camH, write, imgH1)	(camH, write, imgH1), H, E, E, 0, 0)	->(camH, write, imgH1)	allow

Table C.7: Decision making in Use case 2, Scene 2.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, W, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, W, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, W, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, W, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchP, open, scrP1)	(lnchP, open, scrP1), W, W, W, -1, -1)		deny
focus change: W->P				
6	chg(lnchW, open, scrW1)	(lnchW, open, scrW1), P, P, W, -1, -1)	<-(lnchW, open, scrW1)	terminate
7	chg(lnchW, open, tchW1)	(lnchW, open, tchW1), P, P, W, -1, -1)	<-(lnchW, open, tchW1)	terminate
8	chg(scrDr, open, scrW1)	(scrDr, open, scrW1), P, P, W, -1, -1)	<-(scrDr, open, scrW1)	terminate
9	chg(tchDr, open, tchW1)	(tchDr, open, tchW1), P, P, W, -1, -1)	<-(tchDr, open, tchW1)	terminate
10	req(lnchP, open, scrP1)	(lnchP, open, scrP1), P, P, W, -1, -1)	->(lnchP, open, scrP1)	allow
11	req(scrDr, open, scrP1)	(scrDr, open, scrP1), P, P, W, -1, -1)	->(scrDr, open, scrP1)	allow
12	req(lnchP, open, tchP1)	(lnchP, open, tchP1), P, P, W, -1, -1)	->(lnchP, open, tchP1)	allow
13	req(tchDr, open, tchP1)	(tchDr, open, tchP1), P, P, W, -1, -1)	->(tchDr, open, tchP1)	allow
14	req(lnchP, write, scrP1)	(lnchP, write, scrP1), P, P, W, -1, -1)		allow
15	req(scrDr, read, scrP1)	(scrDr, read, scrP1), P, P, W, -1, -1)		allow
16	req(tchDr, write, tchP1)	(tchDr, write, tchP1), P, P, W, -1, -1)		allow
17	req(lnchP, read, tchP1)	(lnchP, read, tchP1), P, P, W, -1, -1)		allow
18	req(lnchP, execute, netmP)	(lnchP, execute, netmP), P, E, E, 0, 0)	->(lnchP, execute, netmP)	allow
19	req(netmP, open, scrP3)	(netmP, open, scrP3), P, P, W, -1, -1)	->(netmP, open, scrP3)	allow
20	req(scrDr, open, scrP3)	(scrDr, open, scrP3), P, P, W, -1, -1)	->(scrDr, open, scrP3)	allow
21	req(netmP, open, tchP3)	(netmP, open, tchP3), P, P, W, -1, -1)	->(netmP, open, tchP3)	allow
22	req(tchDr, open, tchP3)	(tchDr, open, tchP3), P, P, W, -1, -1)	->(tchDr, open, tchP3)	allow
23	req(netmP, write, scrP3)	(netmP, write, scrP3), P, P, W, -1, -1)		allow
24	req(scrDr, read, scrP3)	(scrDr, read, scrP3), P, P, W, -1, -1)		allow
25	req(tchDr, write, tchP3)	(tchDr, write, tchP3), P, P, W, -1, -1)		allow
26	req(netmP, read, tchP3)	(netmP, read, tchP3), P, P, W, -1, -1)		allow
27	req(netmP, open, chP3)	(netmP, open, chP3), P, P, W, 0, 0)	->(netmP, open, chP3)	allow
28	req(chDr, open, chP3)	(chDr, open, chP3), P, P, W, 0, 0)	->(chDr, open, chP3)	allow
29	req(netmP, write, chP3)	(netmP, write, chP3), P, P, W, 0, 0)		allow
30	req(chDr, read, chP3)	(chDr, read, chP3), P, P, W, 0, 0)		allow
31	req(chDr, write, chP3)	(chDr, write, chP3), P, P, W, 0, 0)		allow
32	req(netmP, read, chP3)	(netmP, read, chP3), P, P, W, 0, 0)		allow
33	req(lnchP, write, scrP1)	(lnchP, write, scrP1), P, P, W, -1, -1)		allow
34	req(scrDr, read, scrP1)	(scrDr, read, scrP1), P, P, W, -1, -1)		allow
35	req(tchDr, write, tchP1)	(tchDr, write, tchP1), P, P, W, -1, -1)		allow
36	req(lnchP, read, tchP1)	(lnchP, read, tchP1), P, P, W, -1, -1)		allow
37	req(lnchP, execute, instP)	(lnchP, execute, instP), P, E, E, 0, 0)	->(lnchP, execute, instP)	allow
38	req(instP, open, scrP2)	(instP, open, scrP2), P, P, W, -1, -1)	->(instP, open, scrP2)	allow
39	req(scrDr, open, scrP2)	(scrDr, open, scrP2), P, P, W, -1, -1)	->(scrDr, open, scrP2)	allow
40	req(instP, open, tchP2)	(instP, open, tchP2), P, P, W, -1, -1)	->(instP, open, tchP2)	allow
41	req(tchDr, open, tchP2)	(tchDr, open, tchP2), P, P, W, -1, -1)	->(tchDr, open, tchP2)	allow
42	req(instP, write, scrP2)	(instP, write, scrP2), P, P, W, -1, -1)		allow
43	req(scrDr, read, scrP2)	(scrDr, read, scrP2), P, P, W, -1, -1)		allow
44	req(tchDr, write, tchP2)	(tchDr, write, tchP2), P, P, W, -1, -1)		allow
45	req(instP, read, tchP2)	(instP, read, tchP2), P, P, W, -1, -1)		allow
46	req(instP, open, chP2)	(instP, open, chP2), P, P, P, 1, 1)	->(instP, open, chP2)	allow
47	req(chDr, open, chP2)	(chDr, open, chP2), P, P, P, 1, 1)	->(chDr, open, chP2)	allow
48	req(instP, write, chP2)	(instP, write, chP2), P, P, P, 1, 1)		allow
49	req(chDr, read, chP2)	(chDr, read, chP2), P, P, P, 1, 1)		allow
50	req(chDr, write, chP2)	(chDr, write, chP2), P, P, P, 1, 1)		allow
51	req(instP, read, chP2)	(instP, read, chP2), P, P, P, 1, 1)		allow
52	req(lnchP, execute, cabP)	(lnchP, execute, cabP), P, E, E, 0, 0)	->(lnchP, execute, cabP)	allow
53	req(cabP, open, scrP7)	(cabP, open, scrP7), P, P, W, -1, -1)	->(cabP, open, scrP7)	allow
54	req(scrDr, open, scrP7)	(scrDr, open, scrP7), P, P, W, -1, -1)	->(scrDr, open, scrP7)	allow
55	req(cabP, open, tchP7)	(cabP, open, tchP7), P, P, W, -1, -1)	->(cabP, open, tchP7)	allow
56	req(tchDr, open, tchP7)	(tchDr, open, tchP7), P, P, W, -1, -1)	->(tchDr, open, tchP7)	allow
57	req(cabP, open, chP7)	(cabP, open, chP7), P, P, P, 0, 0)	->(cabP, open, chP7)	allow
58	req(chDr, open, chP7)	(chDr, open, chP7), P, P, P, 0, 0)	->(chDr, open, chP7)	allow
59	req(chDr, write, chP7)	(chDr, write, chP7), P, P, P, 0, 0)		allow
60	req(cabP, read, chP7)	(cabP, read, chP7), P, P, P, 0, 0)		allow
61	req(cabP, write, scrP7)	(cabP, write, scrP7), P, P, W, -1, -1)		allow
62	req(scrDr, read, scrP7)	(scrDr, read, scrP7), P, P, W, -1, -1)		allow
63	req(tchDr, write, tchP7)	(tchDr, write, tchP7), P, P, W, -1, -1)		allow
64	req(cabP, read, tchP7)	(cabP, read, tchP7), P, P, W, -1, -1)		allow
65	req(cabP, write, chP7)	(cabP, write, chP7), P, P, P, 0, 0)		allow
66	req(chDr, read, chP7)	(chDr, read, chP7), P, P, P, 0, 0)		allow
67	req(chDr, write, chP7)	(chDr, write, chP7), P, P, P, 0, 0)		allow
68	req(cabP, read, chP7)	(cabP, read, chP7), P, P, P, 0, 0)		allow

Table C.8: Decision making in Use case 2, Scene 3.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchH, open, scrH1)	(lnchH, open, scrH1), H, H, P, -1, -1)	->(lnchH, open, scrH1)	allow
2	req(scrDr, open, scrH1)	(scrDr, open, scrH1), H, H, P, -1, -1)	->(scrDr, open, scrH1)	allow
3	req(lnchH, open, tchH1)	(lnchH, open, tchH1), H, H, P, -1, -1)	->(lnchH, open, tchH1)	allow
4	req(tchDr, open, tchH1)	(tchDr, open, tchH1), H, H, P, -1, -1)	->(tchDr, open, tchH1)	allow
5	req(lnchH, write, scrH1)	(lnchH, write, scrH1), H, H, P, -1, -1)		allow
6	req(scrDr, read, scrH1)	(scrDr, read, scrH1), H, H, P, -1, -1)		allow
7	req(tchDr, write, tchH1)	(tchDr, write, tchH1), H, H, P, -1, -1)		allow
8	req(lnchH, read, tchH1)	(lnchH, read, tchH1), H, H, P, -1, -1)		allow
9	req(lnchH, execute, camH)	(lnchH, execute, camH), H, E, E, 0, 0)	->(lnchH, execute, camH)	allow
10	req(camH, open, scrH6)	(camH, open, scrH6), H, H, P, -1, -1)	->(camH, open, scrH6)	allow
11	req(scrDr, open, scrH6)	(scrDr, open, scrH6), H, H, P, -1, -1)	->(scrDr, open, scrH6)	allow
12	req(camH, open, tchH6)	(camH, open, tchH6), H, H, P, -1, -1)	->(camH, open, tchH6)	allow
13	req(tchDr, open, tchH6)	(tchDr, open, tchH6), H, H, P, -1, -1)	->(tchDr, open, tchH6)	allow
14	req(camH, write, scrH6)	(camH, write, scrH6), H, H, P, -1, -1)		allow
15	req(scrDr, read, scrH6)	(scrDr, read, scrH6), H, H, P, -1, -1)		allow
16	req(tchDr, write, tchH6)	(tchDr, write, tchH6), H, H, P, -1, -1)		allow
17	req(camH, read, tchH6)	(camH, read, tchH6), H, H, P, -1, -1)		allow
18	req(camH, open, imgH1)	(camH, open, imgH1), H, E, E, 0, 0)	->(camH, open, imgH1)	allow
19	req(camH, read, imgH1)	(camH, read, imgH1), H, E, E, 0, 0)		allow
20	req(camH, write, scrH6)	(camH, write, scrH6), H, H, P, -1, -1)		allow
21	req(scrDr, read, scrH6)	(scrDr, read, scrH6), H, H, P, -1, -1)		allow
22	req(camH, write, scrH6)	(camH, write, scrH6), H, W, P, -1, -1)		allow
23	req(scrDr, read, scrH6)	(scrDr, read, scrH6), H, W, P, -1, -1)		allow
24	req(tchDr, write, tchH6)	(tchDr, write, tchH6), H, H, P, -1, -1)		allow
25	req(camH, read, tchH6)	(camH, read, tchH6), H, H, P, -1, -1)		allow
26	req(camH, read, imgH1)	(camH, read, imgH1), H, E, E, 0, 0)		allow
27	req(camH, write, scrH6)	(camH, write, scrH6), H, W, P, -1, -1)		allow
28	req(scrDr, read, scrH6)	(scrDr, read, scrH6), H, W, P, -1, -1)		allow

Table C.9: Decision making in Use case 2, Scene 4.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, P, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, P, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, P, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, P, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, execute, fmW)	(lnchW, execute, fmW), W, E, E, 0, 0)	->(lnchW, execute, fmW)	allow
6	req(fmW, open, scrW8)	(fmW, open, scrW8), W, W, P, -1, -1)	->(fmW, open, scrW8)	allow
7	req(scrDr, open, scrW8)	(scrDr, open, scrW8), W, W, P, -1, -1)	->(scrDr, open, scrW8)	allow
8	req(fmW, open, tchW8)	(fmW, open, tchW8), W, W, P, -1, -1)	->(fmW, open, tchW8)	allow
9	req(tchDr, open, tchW8)	(tchDr, open, tchW8), W, W, P, -1, -1)	->(tchDr, open, tchW8)	allow
10	req(fmW, write, scrW8)	(fmW, write, scrW8), W, W, P, -1, -1)		allow
11	req(scrDr, read, scrW8)	(scrDr, read, scrW8), W, W, P, -1, -1)		allow
12	req(tchDr, write, tchW8)	(tchDr, write, tchW8), W, W, P, -1, -1)		allow
13	req(fmW, read, tchW8)	(fmW, read, tchW8), W, W, P, -1, -1)		allow
14	req(fmW, open, docW1)	(fmW, open, docW1), W, E, E, 0, 0)	->(fmW, open, docW1)	allow
15	req(fmW, read, docW1)	(fmW, read, docW1), W, E, E, 0, 0)		allow
16	req(fmW, open, chW8)	(fmW, open, chW8), W, W, P, 0, 0)		deny
secure channel established				
17	req(fmW, open, chW8)	(fmW, open, chW8), W, W, P, 1, 1)	->(fmW, open, chW8)	allow
18	req(chDr, open, chW8)	(chDr, open, chW8), W, W, P, 1, 1)	->(chDr, open, chW8)	allow
19	req(fmW, write, chW8)	(fmW, write, chW8), W, W, P, 1, 1)		allow
20	req(chDr, read, chW8)	(chDr, read, chW8), W, W, P, 1, 1)		allow

Table C.10: Decision making in Use case 3, Scene 1.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, P, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, P, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, P, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, P, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, P, -1, -1)		allow
6	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, P, -1, -1)		allow
7	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, P, -1, -1)		allow
8	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, P, -1, -1)		allow
9	req(lnchW, execute, netmW)	(lnchW, execute, netmW), W, E, E, 0, 0)	->(lnchW, execute, netmW)	allow
10	req(netmW, open, scrW3)	(netmW, open, scrW3), W, W, P, -1, -1)	->(netmW, open, scrW3)	allow
11	req(scrDr, open, scrW3)	(scrDr, open, scrW3), W, W, P, -1, -1)	->(scrDr, open, scrW3)	allow
12	req(netmW, open, tchW3)	(netmW, open, tchW3), W, W, P, -1, -1)	->(netmW, open, tchW3)	allow
13	req(tchDr, open, tchW3)	(tchDr, open, tchW3), W, W, P, -1, -1)	->(tchDr, open, tchW3)	allow
14	req(netmW, write, scrW3)	(netmW, write, scrW3), W, W, P, -1, -1)		allow
15	req(scrDr, read, scrW3)	(scrDr, read, scrW3), W, W, P, -1, -1)		allow
16	req(tchDr, write, tchW3)	(tchDr, write, tchW3), W, W, P, -1, -1)		allow
17	req(netmW, read, tchW3)	(netmW, read, tchW3), W, W, P, -1, -1)		allow
18	req(netmW, open, chW3)	(netmW, open, chW3), W, P, P, 0, 0)	->(netmW, open, chW3)	allow
19	req(chDr, open, chW3)	(chDr, open, chW3), W, P, P, 0, 0)	->(chDr, open, chW3)	allow
20	req(netmW, write, chW3)	(netmW, write, chW3), W, P, P, 0, 0)		allow
21	req(chDr, read, chW3)	(chDr, read, chW3), W, P, P, 0, 0)		allow
22	req(chDr, write, chW3)	(chDr, write, chW3), W, P, P, 0, 0)		allow
23	req(netmW, read, chW3)	(netmW, read, chW3), W, P, P, 0, 0)		allow
24	req(lnchW, execute, emailW)	(lnchW, execute, emailW), W, E, E, 0, 0)	->(lnchW, execute, emailW)	allow
25	req(emailW, open, scrW8)	(emailW, open, scrW8), W, W, P, -1, -1)	->(emailW, open, scrW8)	allow
26	req(scrDr, open, scrW8)	(scrDr, open, scrW8), W, W, P, -1, -1)	->(scrDr, open, scrW8)	allow
27	req(emailW, open, tchW8)	(emailW, open, tchW8), W, W, P, -1, -1)	->(emailW, open, tchW8)	allow
28	req(tchDr, open, tchW8)	(tchDr, open, tchW8), W, W, P, -1, -1)	->(tchDr, open, tchW8)	allow
29	req(emailW, write, scrW8)	(emailW, write, scrW8), W, W, P, -1, -1)		allow
30	req(scrDr, read, scrW8)	(scrDr, read, scrW8), W, W, P, -1, -1)		allow
31	req(tchDr, write, tchW8)	(tchDr, write, tchW8), W, W, P, -1, -1)		allow
32	req(emailW, read, tchW8)	(emailW, read, tchW8), W, W, P, -1, -1)		allow
33	req(emailW, open, chW8)	(emailW, open, chW8), W, W, P, 1, 1)	->(emailW, open, chW8)	allow
34	req(chDr, open, chW8)	(chDr, open, chW8), W, W, P, 1, 1)	->(chDr, open, chW8)	allow
35	req(emailW, write, chW8)	(emailW, write, chW8), W, W, P, 1, 1)		allow
36	req(chDr, read, chW8)	(chDr, read, chW8), W, W, P, 1, 1)		allow
37	req(chDr, write, chW8)	(chDr, write, chW8), W, W, P, 1, 1)		allow
38	req(emailW, read, chW8)	(emailW, read, chW8), W, W, P, 1, 1)		allow
39	req(emailW, open, dbW8)	(emailW, open, dbW8), W, E, E, 0, 0)	->(emailW, open, dbW8)	allow
40	req(emailW, read, dbW8)	(emailW, read, dbW8), W, E, E, 0, 0)		allow
41	req(emailW, write, dbW8)	(emailW, write, dbW8), W, E, E, 0, 0)		allow

Table C.11: Decision making in Use case 3, Scene 2.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, P, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, P, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, P, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, P, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, P, -1, -1)		allow
6	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, P, -1, -1)		allow
7	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, P, -1, -1)		allow
8	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, P, -1, -1)		allow
9	req(lnchW, execute, voipW)	(lnchW, execute, voipW), W, E, E, 0, 0)	->(lnchW, execute, voipW)	allow
10	req(voipW, open, scrW5)	(voipW, open, scrW5), W, W, P, -1, -1)	->(voipW, open, scrW5)	allow
11	req(scrDr, open, scrW5)	(scrDr, open, scrW5), W, W, P, -1, -1)	->(scrDr, open, scrW5)	allow
12	req(voipW, open, tchW5)	(voipW, open, tchW5), W, W, P, -1, -1)	->(voipW, open, tchW5)	allow
13	req(tchDr, open, tchW5)	(tchDr, open, tchW5), W, W, P, -1, -1)	->(tchDr, open, tchW5)	allow
14	req(voipW, open, micW5)	(voipW, open, micW5), W, W, P, -1, -1)	->(voipW, open, micW5)	allow
15	req(micDr, open, micW5)	(micDr, open, micW5), W, W, P, -1, -1)	->(micDr, open, micW5)	allow
16	req(voipW, open, spkW5)	(voipW, open, spkW5), W, W, P, -1, -1)	->(voipW, open, spkW5)	allow
17	req(spKDr, open, spkW5)	(spKDr, open, spkW5), W, W, P, -1, -1)	->(spKDr, open, spkW5)	allow
18	req(voipW, open, chW5)	(voipW, open, chW5), W, W, P, 1, 1)	->(voipW, open, chW5)	allow
19	req(chDr, open, chW5)	(chDr, open, chW5), W, W, P, 1, 1)	->(chDr, open, chW5)	allow
20	req(ntfr, open, scrS5)	(ntfr, open, scrS5), W, W, P, -1, -1)	->(ntfr, open, scrS5)	allow
21	req(scrDr, open, scrS5)	(scrDr, open, scrS5), W, W, P, -1, -1)	->(scrDr, open, scrS5)	allow
22	req(lnchP, open, scrP1)	(lnchP, open, scrP1), W, W, P, -1, -1)		deny
focus change: W->P				
23	chg(lnchW, open, scrW1)	(lnchW, open, scrW1), P, P, P, -1, -1)	<-(lnchW, open, scrW1)	terminate
24	chg(lnchW, open, tchW1)	(lnchW, open, tchW1), P, P, P, -1, -1)	<-(lnchW, open, tchW1)	terminate
25	chg(scrDr, open, scrW1)	(scrDr, open, scrW1), P, P, P, -1, -1)	<-(scrDr, open, scrW1)	terminate
26	chg(tchDr, open, tchW1)	(tchDr, open, tchW1), P, P, P, -1, -1)	<-(tchDr, open, tchW1)	terminate
27	req(lnchP, open, scrP1)	(lnchP, open, scrP1), P, P, P, -1, -1)	->(lnchP, open, scrP1)	allow
28	req(scrDr, open, scrP1)	(scrDr, open, scrP1), P, P, P, -1, -1)	->(scrDr, open, scrP1)	allow
29	req(lnchP, open, tchP1)	(lnchP, open, tchP1), P, P, P, -1, -1)	->(lnchP, open, tchP1)	allow
30	req(tchDr, open, tchP1)	(tchDr, open, tchP1), P, P, P, -1, -1)	->(tchDr, open, tchP1)	allow
31	req(lnchP, write, scrP1)	(lnchP, write, scrP1), P, P, P, -1, -1)		allow
32	req(scrDr, read, scrP1)	(scrDr, read, scrP1), P, P, P, -1, -1)		allow
33	req(tchDr, write, tchP1)	(tchDr, write, tchP1), P, P, P, -1, -1)		allow
34	req(lnchP, read, tchP1)	(lnchP, read, tchP1), P, P, P, -1, -1)		allow
35	req(lnchP, execute, traffP)	(lnchP, execute, traffP), P, E, E, 0, 0)	->(lnchP, execute, traffP)	allow
36	req(traffP, open, scrP7)	(traffP, open, scrP7), P, P, P, -1, -1)	->(traffP, open, scrP7)	allow
37	req(scrDr, open, scrP7)	(scrDr, open, scrP7), P, P, P, -1, -1)	->(scrDr, open, scrP7)	allow
38	req(traffP, open, tchP7)	(traffP, open, tchP7), P, P, P, -1, -1)	->(traffP, open, tchP7)	allow
39	req(tchDr, open, tchP7)	(tchDr, open, tchP7), P, P, P, -1, -1)	->(tchDr, open, tchP7)	allow
40	req(traffP, open, geoP7)	(traffP, open, geoP7), P, P, P, -1, -1)	->(traffP, open, geoP7)	allow
41	req(geoDr, open, geoP7)	(geoDr, open, geoP7), P, P, P, -1, -1)	->(geoDr, open, geoP7)	allow

Continued on next page

Table C.11 – Continued from previous page

ID	Input Event	Activity Context	Continuous Activities	Decision
42	req(traffP, open, chP7)	(traffP, open, chP7), P, P, P, 0, 0)	->(traffP, open, chP7)	allow
43	req(chDr, open, chP7)	(chDr, open, chP7), P, P, P, 0, 0)	->(chDr, open, chP7)	allow
44	req(traffP, open, spkP7)	(traffP, open, spkP7), P, P, P, -1, -1)	->(traffP, open, spkP7)	allow
45	req(spKDr, open, spkP7)	(spkDr, open, spkP7), P, P, P, -1, -1)	->(spkDr, open, spkP7)	allow
46	req(traffP, write, scrP7)	(traffP, write, scrP7), P, P, P, -1, -1)		allow
47	req(scrDr, read, scrP7)	(scrDr, read, scrP7), P, P, P, -1, -1)		allow
48	req(geoDr, write, geoP7)	(geoDr, write, geoP7), P, P, P, -1, -1)		allow
49	req(traffP, read, geoP7)	(traffP, read, geoP7), P, P, P, -1, -1)		allow
50	req(traffP, write, chP7)	(traffP, write, chP7), P, P, P, 0, 0)		allow
51	req(chDr, read, chP7)	(chDr, read, chP7), P, P, P, 0, 0)		allow
52	req(chDr, write, chP7)	(chDr, write, chP7), P, P, P, 0, 0)		allow
53	req(traffP, read, chP7)	(traffP, read, chP7), P, P, P, 0, 0)		allow
54	req(traffP, write, scrP7)	(traffP, write, scrP7), P, P, P, -1, -1)		allow
55	req(scrDr, read, scrP7)	(scrDr, read, scrP7), P, P, P, -1, -1)		allow
56	req(traffP, write, spkP7)	(traffP, write, spkP7), P, P, P, -1, -1)		allow
57	req(spKDr, read, spkP7)	(spkDr, read, spkP7), P, P, P, -1, -1)		allow
58	req(chDr, write, chW5)	(chDr, write, chW5), P, W, P, 1, 1)		allow
59	req(voipW, read, chW5)	(voipW, read, chW5), P, W, P, 1, 1)		allow
60	req(voipW, write, chW5)	(voipW, write, chW5), P, W, P, 1, 1)		allow
61	req(chDr, read, chW5)	(chDr, read, chW5), P, W, P, 1, 1)		allow
62	req(voipW, write, scrW5)	(voipW, write, scrW5), P, P, P, -1, -1)		deny
63	req(ntfr, write, scrS5)	(ntfr, write, scrS5), P, P, P, -1, -1)		allow
64	req(scrDr, read, scrS5)	(scrDr, read, scrS5), P, P, P, -1, -1)		allow
65	req(lnchW, open, scrW1)	(lnchW, open, scrW1), P, P, P, -1, -1)		deny
focus change: P->W				
66	chg(lnchP, open, scrP1)	(lnchP, open, scrP1), W, W, P, -1, -1)	<-(lnchP, open, scrP1)	terminate
67	chg(lnchP, open, tchP1)	(lnchP, open, tchP1), W, W, P, -1, -1)	<-(lnchP, open, tchP1)	terminate
68	chg(lnchP, execute, traffP)	(lnchP, execute, traffP), W, E, E, 0, 0)	<-(lnchP, execute, traffP)	terminate
69	chg(traffP, open, scrP7)	(traffP, open, scrP7), W, W, P, -1, -1)	<-(traffP, open, scrP7)	terminate
70	chg(traffP, open, tchP7)	(traffP, open, tchP7), W, W, P, -1, -1)	<-(traffP, open, tchP7)	terminate
71	chg(traffP, open, geoP7)	(traffP, open, geoP7), W, W, P, -1, -1)	<-(traffP, open, geoP7)	terminate
72	chg(traffP, open, chP7)	(traffP, open, chP7), W, P, P, 0, 0)	<-(traffP, open, chP7)	terminate
73	chg(traffP, open, spkP7)	(traffP, open, spkP7), W, W, P, -1, -1)	<-(traffP, open, spkP7)	terminate
74	chg(scrDr, open, scrP1)	(scrDr, open, scrP1), W, W, P, -1, -1)	<-(scrDr, open, scrP1)	terminate
75	chg(tchDr, open, tchP1)	(tchDr, open, tchP1), W, W, P, -1, -1)	<-(tchDr, open, tchP1)	terminate
76	chg(scrDr, open, scrP7)	(scrDr, open, scrP7), W, W, P, -1, -1)	<-(scrDr, open, scrP7)	terminate
77	chg(tchDr, open, tchP7)	(tchDr, open, tchP7), W, W, P, -1, -1)	<-(tchDr, open, tchP7)	terminate
78	chg(geoDr, open, geoP7)	(geoDr, open, geoP7), W, W, P, -1, -1)	<-(geoDr, open, geoP7)	terminate
79	chg(chDr, open, chP7)	(chDr, open, chP7), W, P, P, 0, 0)	<-(chDr, open, chP7)	terminate
80	chg(spKDr, open, spkP7)	(spkDr, open, spkP7), W, W, P, -1, -1)	<-(spkDr, open, spkP7)	terminate
81	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, P, -1, -1)	->(lnchW, open, scrW1)	allow
82	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, P, -1, -1)	->(scrDr, open, scrW1)	allow
83	req(tchDr, write, tchW5)	(tchDr, write, tchW5), W, W, P, -1, -1)		allow
84	req(voipW, read, tchW5)	(voipW, read, tchW5), W, W, P, -1, -1)		allow
85	req(voipW, write, scrW5)	(voipW, write, scrW5), W, W, P, -1, -1)		allow
86	req(scrDr, read, scrW5)	(scrDr, read, scrW5), W, W, P, -1, -1)		allow
87	req(voipW, write, chW5)	(voipW, write, chW5), W, W, P, 1, 1)		allow
88	req(chDr, read, chW5)	(chDr, read, chW5), W, W, P, 1, 1)		allow
89	req(chDr, write, chW5)	(chDr, write, chW5), W, W, P, 1, 1)		allow
90	req(voipW, read, chW5)	(voipW, read, chW5), W, W, P, 1, 1)		allow
91	req(micDr, write, micW5)	(micDr, write, micW5), W, W, P, -1, -1)		allow
92	req(voipW, read, micW5)	(voipW, read, micW5), W, W, P, -1, -1)		allow
93	req(voipW, write, spkW5)	(voipW, write, spkW5), W, W, P, -1, -1)		allow
94	req(spKDr, read, spkW5)	(spkDr, read, spkW5), W, W, P, -1, -1)		allow
95	req(micDr, write, micW5)	(micDr, write, micW5), W, P, P, -1, -1)		allow
96	req(voipW, read, micW5)	(voipW, read, micW5), W, P, P, -1, -1)		allow
97	req(voipW, write, spkW5)	(voipW, write, spkW5), W, P, P, -1, -1)		deny
98	req(voipW, close, spkW5)	(voipW, close, spkW5), W, P, P, -1, -1)	<-(voipW, open, spkW5)	allow
99	req(spKDr, close, spkW5)	(spkDr, close, spkW5), W, P, P, -1, -1)	<-(spkDr, open, spkW5)	allow
100	req(voipW, open, spkW51)	(voipW, open, spkW51), W, W, P, -1, -1)	->(voipW, open, spkW51)	allow
101	req(spKDr, open, spkW51)	(spkDr, open, spkW51), W, W, P, -1, -1)	->(spkDr, open, spkW51)	allow
102	req(voipW, write, spkW51)	(voipW, write, spkW51), W, W, P, -1, -1)		allow
103	req(spKDr, read, spkW51)	(spkDr, read, spkW51), W, W, P, -1, -1)		allow
104	req(micDr, write, micW5)	(micDr, write, micW5), W, P, P, -1, -1)		allow
105	req(voipW, read, micW5)	(voipW, read, micW5), W, P, P, -1, -1)		allow
106	req(voipW, write, chW5)	(voipW, write, chW5), W, W, P, 1, 1)		allow
107	req(chDr, read, chW5)	(chDr, read, chW5), W, W, P, 1, 1)		allow

Table C.12: Decision making in Use case 3, Scene 3.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, P, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, P, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, P, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, P, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, P, -1, -1)		allow
6	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, P, -1, -1)		allow
7	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, P, -1, -1)		allow
8	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, P, -1, -1)		allow
9	req(lnchW, execute, offW)	(lnchW, execute, offW), W, E, E, 0, 0)	->(lnchW, execute, offW)	allow
10	req(offW, open, scrW7)	(offW, open, scrW7), W, W, P, -1, -1)	->(offW, open, scrW7)	allow
11	req(scrDr, open, scrW7)	(scrDr, open, scrW7), W, W, P, -1, -1)	->(scrDr, open, scrW7)	allow
12	req(offW, open, tchW7)	(offW, open, tchW7), W, W, P, -1, -1)	->(offW, open, tchW7)	allow
13	req(tchDr, open, tchW7)	(tchDr, open, tchW7), W, W, P, -1, -1)	->(tchDr, open, tchW7)	allow
14	req(offW, write, scrW7)	(offW, write, scrW7), W, W, P, -1, -1)		allow
15	req(scrDr, read, scrW7)	(scrDr, read, scrW7), W, W, P, -1, -1)		allow
16	req(tchDr, write, tchW7)	(tchDr, write, tchW7), W, W, P, -1, -1)		allow
17	req(offW, read, tchW7)	(offW, read, tchW7), W, W, P, -1, -1)		allow
18	req(offW, open, sldW7)	(offW, open, sldW7), W, E, E, 0, 0)	->(offW, open, sldW7)	allow
19	req(offW, read, sldW7)	(offW, read, sldW7), W, E, E, 0, 0)		allow
20	req(offW, open, chW7)	(offW, open, chW7), W, W, P, 1, 1)	->(offW, open, chW7)	allow

Continued on next page

Table C.12 – Continued from previous page

ID	Input Event	Activity Context	Continuous Activities	Decision
21	req(chDr, open, chW7)	(chDr, open, chW7), W, W, P, 1, 1)	->(chDr, open, chW7)	allow
22	req(offW, write, chW7)	(offW, write, chW7), W, W, P, 1, 1)		allow
23	req(chDr, read, chW7)	(chDr, read, chW7), W, W, P, 1, 1)		allow
24	req(chDr, write, chW7)	(chDr, write, chW7), W, W, P, 1, 1)		allow
25	req(offW, read, chW7)	(offW, read, chW7), W, W, P, 1, 1)		allow
26	req(offW, write, chW7)	(offW, write, chW7), W, W, P, 1, 1)		deny

Table C.13: Decision making in Use case 3, Scene 4.

ID	Input Event	Activity Context	Continuous Activities	Decision
1	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, P, -1, -1)	->(lnchW, open, scrW1)	allow
2	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, P, -1, -1)	->(scrDr, open, scrW1)	allow
3	req(lnchW, open, tchW1)	(lnchW, open, tchW1), W, W, P, -1, -1)	->(lnchW, open, tchW1)	allow
4	req(tchDr, open, tchW1)	(tchDr, open, tchW1), W, W, P, -1, -1)	->(tchDr, open, tchW1)	allow
5	req(lnchW, write, scrW1)	(lnchW, write, scrW1), W, W, P, -1, -1)		allow
6	req(scrDr, read, scrW1)	(scrDr, read, scrW1), W, W, P, -1, -1)		allow
7	req(tchDr, write, tchW1)	(tchDr, write, tchW1), W, W, P, -1, -1)		allow
8	req(lnchW, read, tchW1)	(lnchW, read, tchW1), W, W, P, -1, -1)		allow
9	req(lnchW, execute, emailW)	(lnchW, execute, emailW), W, E, E, 0, 0)	->(lnchW, execute, emailW)	allow
10	req(emailW, open, scrW8)	(emailW, open, scrW8), W, W, P, -1, -1)	->(emailW, open, scrW8)	allow
11	req(scrDr, open, scrW8)	(scrDr, open, scrW8), W, W, P, -1, -1)	->(scrDr, open, scrW8)	allow
12	req(emailW, open, tchW8)	(emailW, open, tchW8), W, W, P, -1, -1)	->(emailW, open, tchW8)	allow
13	req(tchDr, open, tchW8)	(tchDr, open, tchW8), W, W, P, -1, -1)	->(tchDr, open, tchW8)	allow
14	req(emailW, open, chW8)	(emailW, open, chW8), W, W, P, 1, 1)	->(emailW, open, chW8)	allow
15	req(chDr, open, chW8)	(chDr, open, chW8), W, W, P, 1, 1)	->(chDr, open, chW8)	allow
16	req(emailW, open, dbW8)	(emailW, open, dbW8), W, E, E, 0, 0)	->(emailW, open, dbW8)	allow
17	req(ntfr, open, scrS5)	(ntfr, open, scrS5), W, W, P, -1, -1)	->(ntfr, open, scrS5)	allow
18	req(scrDr, open, scrS5)	(scrDr, open, scrS5), W, W, P, -1, -1)	->(scrDr, open, scrS5)	allow
19	req(lnchP, open, scrP1)	(lnchP, open, scrP1), W, W, P, -1, -1)		deny
focus change: W->P				
20	chg(lnchW, open, scrW1)	(lnchW, open, scrW1), P, P, P, -1, -1)	<-(lnchW, open, scrW1)	terminate
21	chg(lnchW, open, tchW1)	(lnchW, open, tchW1), P, P, P, -1, -1)	<-(lnchW, open, tchW1)	terminate
22	chg(scrDr, open, scrW1)	(scrDr, open, scrW1), P, P, P, -1, -1)	<-(scrDr, open, scrW1)	terminate
23	chg(tchDr, open, tchW1)	(tchDr, open, tchW1), P, P, P, -1, -1)	<-(tchDr, open, tchW1)	terminate
24	req(lnchP, open, scrP1)	(lnchP, open, scrP1), P, P, P, -1, -1)	->(lnchP, open, scrP1)	allow
25	req(scrDr, open, scrP1)	(scrDr, open, scrP1), P, P, P, -1, -1)	->(scrDr, open, scrP1)	allow
26	req(lnchP, open, tchP1)	(lnchP, open, tchP1), P, P, P, -1, -1)	->(lnchP, open, tchP1)	allow
27	req(tchDr, open, tchP1)	(tchDr, open, tchP1), P, P, P, -1, -1)	->(tchDr, open, tchP1)	allow
28	req(lnchP, write, scrP1)	(lnchP, write, scrP1), P, P, P, -1, -1)		allow
29	req(scrDr, read, scrP1)	(scrDr, read, scrP1), P, P, P, -1, -1)		allow
30	req(tchDr, write, tchP1)	(tchDr, write, tchP1), P, P, P, -1, -1)		allow
31	req(lnchP, read, tchP1)	(lnchP, read, tchP1), P, P, P, -1, -1)		allow
32	req(lnchP, execute, nfcP)	(lnchP, execute, nfcP), P, E, E, 0, 0)	->(lnchP, execute, nfcP)	allow
33	req(nfcP, open, chP4)	(nfcP, open, chP4), P, H, P, 0, 0)		deny
focus change: P->H				
34	chg(emailW, open, scrW8)	(emailW, open, scrW8), H, H, P, -1, -1)	<-(emailW, open, scrW8)	terminate
35	chg(emailW, open, tchW8)	(emailW, open, tchW8), H, H, P, -1, -1)	<-(emailW, open, tchW8)	terminate
36	chg(lnchP, open, scrP1)	(lnchP, open, scrP1), H, H, P, -1, -1)	<-(lnchP, open, scrP1)	terminate
37	chg(lnchP, open, tchP1)	(lnchP, open, tchP1), H, H, P, -1, -1)	<-(lnchP, open, tchP1)	terminate
38	chg(lnchP, execute, nfcP)	(lnchP, execute, nfcP), H, E, E, 0, 0)	<-(lnchP, execute, nfcP)	terminate
39	chg(scrDr, open, scrW8)	(scrDr, open, scrW8), H, H, P, -1, -1)	<-(scrDr, open, scrW8)	terminate
40	chg(tchDr, open, tchW8)	(tchDr, open, tchW8), H, H, P, -1, -1)	<-(tchDr, open, tchW8)	terminate
41	chg(scrDr, open, scrP1)	(scrDr, open, scrP1), H, H, P, -1, -1)	<-(scrDr, open, scrP1)	terminate
42	chg(tchDr, open, tchP1)	(tchDr, open, tchP1), H, H, P, -1, -1)	<-(tchDr, open, tchP1)	terminate
43	req(lnchH, open, scrH1)	(lnchH, open, scrH1), H, H, P, -1, -1)	->(lnchH, open, scrH1)	allow
44	req(scrDr, open, scrH1)	(scrDr, open, scrH1), H, H, P, -1, -1)	->(scrDr, open, scrH1)	allow
45	req(lnchH, open, tchH1)	(lnchH, open, tchH1), H, H, P, -1, -1)	->(lnchH, open, tchH1)	allow
46	req(tchDr, open, tchH1)	(tchDr, open, tchH1), H, H, P, -1, -1)	->(tchDr, open, tchH1)	allow
47	req(lnchH, write, scrH1)	(lnchH, write, scrH1), H, H, P, -1, -1)		allow
48	req(scrDr, read, scrH1)	(scrDr, read, scrH1), H, H, P, -1, -1)		allow
49	req(tchDr, write, tchH1)	(tchDr, write, tchH1), H, H, P, -1, -1)		allow
50	req(lnchH, read, tchH1)	(lnchH, read, tchH1), H, H, P, -1, -1)		allow
51	req(lnchH, execute, nfcH)	(lnchH, execute, nfcH), H, E, E, 0, 0)	->(lnchH, execute, nfcH)	allow
52	req(nfcH, open, chH4)	(nfcH, open, chH4), H, H, P, 0, 0)	->(nfcH, open, chH4)	allow
53	req(chDr, open, chH4)	(chDr, open, chH4), H, H, P, 0, 0)	->(chDr, open, chH4)	allow
54	req(chDr, write, chH4)	(chDr, write, chH4), H, H, P, 0, 0)		allow
55	req(nfcH, read, chH4)	(nfcH, read, chH4), H, H, P, 0, 0)		allow
56	req(nfcH, write, chH4)	(nfcH, write, chH4), H, H, P, 0, 0)		allow
57	req(chDr, read, chH4)	(chDr, read, chH4), H, H, P, 0, 0)		allow
58	req(lnchH, execute, videoH)	(lnchH, execute, videoH), H, E, E, 0, 0)	->(lnchH, execute, videoH)	allow
59	req(videoH, open, scrH7)	(videoH, open, scrH7), H, H, H, 0, 0)	->(videoH, open, scrH7)	allow
60	req(scrDr, open, scrH7)	(scrDr, open, scrH7), H, H, H, 0, 0)	->(scrDr, open, scrH7)	allow
61	req(videoH, open, tchH7)	(videoH, open, tchH7), H, H, H, 0, 0)	->(videoH, open, tchH7)	allow
62	req(tchDr, open, tchH7)	(tchDr, open, tchH7), H, H, H, 0, 0)	->(tchDr, open, tchH7)	allow
63	req(videoH, open, chH7)	(videoH, open, chH7), H, H, H, 0, 0)	->(videoH, open, chH7)	allow
64	req(chDr, open, chH7)	(chDr, open, chH7), H, H, H, 0, 0)	->(chDr, open, chH7)	allow
65	req(videoH, open, aviH7)	(videoH, open, aviH7), H, E, E, 0, 0)	->(videoH, open, aviH7)	allow
66	req(videoH, read, aviH7)	(videoH, read, aviH7), H, E, E, 0, 0)		allow
67	req(videoH, write, chH7)	(videoH, write, chH7), H, H, H, 0, 0)		allow
68	req(chDr, read, chH7)	(chDr, read, chH7), H, H, H, 0, 0)		allow
69	req(chDr, write, chH7)	(chDr, write, chH7), H, H, H, 0, 0)		allow
70	req(videoH, read, chH7)	(videoH, read, chH7), H, H, H, 0, 0)		allow
71	req(chDr, write, chW8)	(chDr, write, chW8), H, W, P, 1, 1)		allow
72	req(emailW, read, chW8)	(emailW, read, chW8), H, W, P, 1, 1)		allow
73	req(emailW, write, dbW8)	(emailW, write, dbW8), H, E, E, 0, 0)		allow
74	req(emailW, open, scrW8)	(emailW, open, scrW8), H, H, H, 0, 0)		deny
75	req(ntfr, write, scrS5)	(ntfr, write, scrS5), H, H, H, 0, 0)		allow
76	req(scrDr, read, scrS5)	(scrDr, read, scrS5), H, H, H, 0, 0)		allow
77	req(lnchW, open, scrW1)	(lnchW, open, scrW1), H, H, H, -1, -1)		deny
focus change: H->W				

Continued on next page

Table C.13 – Continued from previous page

ID	Input Event	Activity Context	Continuous Activities	Decision
78	chg(lnchH, open, scrH1)	(lnchH, open, scrH1), W, W, H, -1, -1)	<-(lnchH, open, scrH1)	terminate
79	chg(lnchH, open, tchH1)	(lnchH, open, tchH1), W, W, H, -1, -1)	<-(lnchH, open, tchH1)	terminate
80	chg(lnchH, execute, nfcH)	(lnchH, execute, nfcH), W, E, E, 0, 0)	<-(lnchH, execute, nfcH)	terminate
81	chg(nfcH, open, chH4)	(nfcH, open, chH4), W, H, P, 0, 0)	<-(nfcH, open, chH4)	terminate
82	chg(lnchH, execute, videoH)	(lnchH, execute, videoH), W, E, E, 0, 0)	<-(lnchH, execute, videoH)	terminate
83	chg(videoH, open, scrH7)	(videoH, open, scrH7), W, H, H, 0, 0)	<-(videoH, open, scrH7)	terminate
84	chg(videoH, open, tchH7)	(videoH, open, tchH7), W, H, H, 0, 0)	<-(videoH, open, tchH7)	terminate
85	chg(videoH, open, chH7)	(videoH, open, chH7), W, H, H, 0, 0)	<-(videoH, open, chH7)	terminate
86	chg(videoH, open, aviH7)	(videoH, open, aviH7), W, E, E, 0, 0)	<-(videoH, open, aviH7)	terminate
87	chg(scrDr, open, scrH1)	(scrDr, open, scrH1), W, W, H, -1, -1)	<-(scrDr, open, scrH1)	terminate
88	chg(tchDr, open, tchH1)	(tchDr, open, tchH1), W, W, H, -1, -1)	<-(tchDr, open, tchH1)	terminate
89	chg(chDr, open, chH4)	(chDr, open, chH4), W, H, P, 0, 0)	<-(chDr, open, chH4)	terminate
90	chg(scrDr, open, scrH7)	(scrDr, open, scrH7), W, H, H, 0, 0)	<-(scrDr, open, scrH7)	terminate
91	chg(tchDr, open, tchH7)	(tchDr, open, tchH7), W, H, H, 0, 0)	<-(tchDr, open, tchH7)	terminate
92	chg(chDr, open, chH7)	(chDr, open, chH7), W, H, H, 0, 0)	<-(chDr, open, chH7)	terminate
93	req(lnchW, open, scrW1)	(lnchW, open, scrW1), W, W, H, -1, -1)	->(lnchW, open, scrW1)	allow
94	req(scrDr, open, scrW1)	(scrDr, open, scrW1), W, W, H, -1, -1)	->(scrDr, open, scrW1)	allow
95	req(emailW, open, scrW8)	(emailW, open, scrW8), W, W, H, -1, -1)	->(emailW, open, scrW8)	allow
96	req(scrDr, open, scrW8)	(scrDr, open, scrW8), W, W, H, -1, -1)	->(scrDr, open, scrW8)	allow
97	req(emailW, open, tchW8)	(emailW, open, tchW8), W, W, H, -1, -1)	->(emailW, open, tchW8)	allow
98	req(tchDr, open, tchW8)	(tchDr, open, tchW8), W, W, H, -1, -1)	->(tchDr, open, tchW8)	allow
99	req(emailW, write, scrW8)	(emailW, write, scrW8), W, W, H, -1, -1)		allow
100	req(scrDr, read, scrW8)	(scrDr, read, scrW8), W, W, H, -1, -1)		allow
101	req(tchDr, write, tchW8)	(tchDr, write, tchW8), W, W, H, -1, -1)		allow
102	req(emailW, read, tchW8)	(emailW, read, tchW8), W, W, H, -1, -1)		allow

References

- [1] M. Weiser, “The computer for the 21st century,” *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.
- [2] A. Schmidt, B. Pfleging, F. Alt, A. Sahami, and G. Fitzpatrick, “Interacting with 21st-century computers,” *Pervasive Computing*, vol. 11, no. 1, pp. 22–31, 2012.
- [3] E. Papadopoulou, S. Gallacher, N. K. Taylor, and M. H. Williams, “A personal smart space approach to realising ambient ecologies,” *Pervasive and Mobile Computing*, vol. 8, no. 4, pp. 485–499, 2012.
- [4] R. C. Basole, “The value and impact of mobile information and communication technologies,” in *Proceedings of the IFAC Symposium on Analysis, Modeling & Evaluation of Human-Machine Systems*, 2004, pp. 1–7.
- [5] R. C. Basole, “The emergence of the mobile enterprise: A value-driven perspective,” in *International Conference on the Management of Mobile Business, ICMB 2007*, IEEE, 2007, pp. 41–41.
- [6] B. Niehaves, S. Köffer, and K. Ortbach, “IT consumerization—A theory and practice review,” in *Americas Conference on Information Systems, AMCIS 2012*, 2012.
- [7] J. Clarke, M. G. Hidalgo, A. Lioy, M. Petkovic, C. Vishik, and J. Ward, “Consumerization of IT: Top risks and opportunities,” European Network and Information Security Agency (ENISA), Tech. Rep., 2012.
- [8] J. Harris, B. Ives, and I. Junglas, “IT consumerization: When gadgets turn into enterprise IT tools,” *MIS Quarterly Executive*, vol. 11, no. 3, pp. 99–111, 2012.
- [9] P. D’Arcy and L. E. Marketing, *CIO strategies for consumerization: The future of enterprise mobile computing*, 2011.
- [10] A. Hines and C. Carbone, “The future of knowledge work,” *Employment Relations Today*, vol. 40, no. 1, pp. 1–17, 2013.
- [11] V. Ramakrishna, K. Eustice, and M. Schnaider, “Approaches for ensuring security and privacy in unplanned ubiquitous computing interactions,” in *Mobile and Wireless Network Security and Privacy*, S. Makki, P. Reiher, K. Makki, N. Pissinou, and S. Makki, Eds., Springer US, 2007, pp. 167–189.
- [12] R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. D. Mickunas, “Towards security and privacy for pervasive computing,” in *Software Security - Theories and Systems*, M. Okada, B. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, Eds., ser. Lecture Notes in Computer Science, vol. 2609, Springer Berlin Heidelberg, 2003, pp. 1–15.
- [13] P. Robinson and M. Beigl, “Trust context spaces: An infrastructure for pervasive security in context-aware environments,” in *Security in Pervasive Computing*, D. Hutter, G. Müller, W. Stephan, and M. Ullmann, Eds., ser. Lecture Notes in Computer Science, vol. 2802, Springer Berlin Heidelberg, 2004, pp. 157–172.

- [14] M. La Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 1, pp. 446–471, 2013.
- [15] M. Haque and S. I. Ahamed, "Security in pervasive computing: Current status and open issues," *International Journal of Network Security*, vol. 3, no. 3, pp. 203–214, 2006.
- [16] R. Sandhu, "Good-enough security," *Internet Computing*, vol. 7, no. 1, pp. 66–68, 2003.
- [17] R. K. Thomas and R. Sandhu, "Models, protocols, and architectures for secure pervasive computing: Challenges and research directions," in *Proceedings of the IEEE Conference on Pervasive Computing and Communications, PerCom '04*, IEEE Computer Society, 2004, pp. 164–168.
- [18] M. Bazire and P. Brézillon, "Understanding context before using it," in *Modeling and Using Context*, A. Dey, B. Kokinov, D. Leake, and R. Turner, Eds., ser. Lecture Notes in Computer Science, vol. 3554, Springer Berlin Heidelberg, 2005, pp. 29–40.
- [19] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dept. of Computer Science, Dartmouth College, Tech. Rep., 2000.
- [20] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.
- [21] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd, "Securing context-aware applications using environment roles," in *Proceedings of the ACM Symposium on Access Control Models and Technologies, SACMAT '01*, ACM, 2001, pp. 10–20.
- [22] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M. D. Mickunas, "Cerberus: A context-aware security scheme for smart spaces," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications, PerCom '03*, IEEE Computer Society, 2003, pp. 489–496.
- [23] D. Kulkarni and A. Tripathi, "Context-aware role-based access control in pervasive computing systems," in *Proceedings of the ACM Symposium on Access Control Models and Technologies, SACMAT '08*, ACM, 2008, pp. 113–122.
- [24] A. Ahmed and N. Zhang, "Towards the realisation of context-risk-aware access control in pervasive computing," *Telecommunication Systems*, vol. 45, no. 2-3, pp. 127–137, 2010.
- [25] F. Cuppens and N. Cuppens-Boulahia, "Modeling contextual security policies," *International Journal of Information Security*, vol. 7, no. 4, pp. 285–305, 2008.
- [26] G. Johnson, P. Shakarian, N. Gupta, and A. Agrawala, "Towards shrink-wrapped security: Practically incorporating context into security services," *Procedia Computer Science*, vol. 5, pp. 782–787, 2011.
- [27] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila, "Proteus: A semantic context-aware adaptive policy model," in *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY '07*, IEEE Computer Society, 2007, pp. 129–140.

- [28] D. Feth and C. Jung, "Context-aware, data-driven policy enforcement for smart mobile devices in business environments," in *Security and Privacy in Mobile Information and Communication Systems*, A. Schmidt, G. Russello, I. Krontiris, and S. Lian, Eds., ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 107, Springer Berlin Heidelberg, 2012, pp. 69–80.
- [29] B. Ksiezopolski and Z. Kotulski, "Adaptable security mechanism for dynamic environments," *Computers & Security*, vol. 26, no. 3, pp. 246–255, 2007.
- [30] 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*, A. Schmidt and S. Lian, Eds., ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 17, Springer Berlin Heidelberg, 2009, pp. 134–147.
- [31] S. Hachem, A. Toninelli, A. Pathak, and V. Issarny, "Policy-based access control in mobile social ecosystems," in *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY '11*, IEEE Computer Society, 2011, pp. 57–64.
- [32] G. Bai, L. Gu, T. Feng, Y. Guo, and X. Chen, "Context-aware usage control for Android," in *Security and Privacy in Communication Networks*, S. Jajodia and J. Zhou, Eds., ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 50, Springer Berlin Heidelberg, 2010, pp. 326–343.
- [33] G. Spanoudakis and S. Kokolakis, *Security and dependability for Ambient Intelligence*. Springer, 2009.
- [34] G. K. Mostefaoui, "Towards a conceptual and software framework for integrating context-based security in pervasive environments," PhD thesis, University of Fribourg, 2004.
- [35] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos, "Progressive authentication: Deciding when to authenticate on mobile phones," in *Proceedings of the USENIX Security Symposium, Security '12*, USENIX Association, 2012, pp. 15–15.
- [36] C. Bettini, O. Brdiczka, K. Henriksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.
- [37] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.
- [38] J. Ye, S. Dobson, and S. McKeever, "Situation identification techniques in pervasive computing: A review," *Pervasive and Mobile Computing*, vol. 8, no. 1, pp. 36–66, 2012.
- [39] W. Stallings and L. V. Brown, *Computer security: Principles and practice*. Prentice-Hall, 2008.
- [40] M. A. Bishop, *The Art and Science of Computer Security*. Addison-Wesley, 2002.
- [41] G. S. Graham and P. J. Denning, "Protection: Principles and practice," in *Proceedings of the May 16-18, 1972, spring joint computer conference*, ACM, 1972, pp. 417–429.
- [42] D. E. Bell and L. J. La Padula, "Secure computer system: Unified exposition and multics interpretation," The MITRE Corporation, Tech. Rep., 1976.

- [43] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.
- [44] T. Y. Woo and S. S. Lam, "Authorization in distributed systems: A new approach," *Journal of Computer Security*, vol. 2, no. 2-3, pp. 107–136, 1993.
- [45] S. Jajodia, P. Samarati, M. L. Sapino, and V. Subrahmanian, "Flexible support for multiple access control policies," *ACM Transactions on Database Systems (TODS)*, vol. 26, no. 2, pp. 214–260, 2001.
- [46] J. Park and R. Sandhu, "The UCON ABC usage control model," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 128–174, 2004.
- [47] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder policy specification language," in *Policies for Distributed Systems and Networks*, Springer, 2001, pp. 18–38.
- [48] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, IEEE, 2003, pp. 63–74.
- [49] Shirey, Robert W, "Internet security glossary, version 2," The Internet Engineering Task Force (IETF), Tech. Rep. RFC: 4949, 2007.
- [50] ITU-T, "Security architecture for open systems interconnection for CCITT applications," International Telecommunications Union (ITU), Tech. Rep. Recommendation X.800, 1991.
- [51] D. F. Sterne, "On the buzzword security policy," in *Research in security and privacy, 1991. proceedings., 1991 ieee computer society symposium on*, IEEE, 1991, pp. 219–230.
- [52] ISO/IEC, "ISO/IEC 27002:2005 - Information technology - Security techniques - Code of practice for information security management," ISO/IEC, Tech. Rep. 27002:2005, 2005.
- [53] P. Dourish, "What we talk about when we talk about context," *Personal and Ubiquitous Computing*, vol. 8, no. 1, pp. 19–30, 2004.
- [54] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and Ubiquitous Computing*, H.-W. Gellersen, Ed., ser. Lecture Notes in Computer Science, vol. 1707, Springer Berlin Heidelberg, 1999, pp. 304–307.
- [55] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.
- [56] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—Degrees, models, and applications," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, p. 7, 2008.
- [57] V. Vieira, P. Tedesco, and A. C. Salgado, "Designing context-sensitive systems: An integrated approach," *Expert Systems with Applications*, vol. 38, no. 2, pp. 1119–1138, 2011, Intelligent Collaboration and Design.
- [58] T. Winograd, "Architectures for context," *Human-Computer Interaction*, vol. 16, no. 2, pp. 401–419, 2001.
- [59] R. Boutaba and I. Aib, "Policy-based management: A historical perspective," *Journal of Network and Systems Management*, vol. 15, no. 4, pp. 447–480, 2007.

- [60] M. Sloman, "Policy driven management for distributed systems," *Journal of network and Systems Management*, vol. 2, no. 4, pp. 333–360, 1994.
- [61] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *1st International workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, England*, 2004.
- [62] M. Perttunen, J. Riekki, and O. Lassila, "Context representation and reasoning in pervasive computing: A review," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 4, no. 4, 2009.
- [63] C. Hoareau and I. Satoh, "Modeling and processing information for context-aware computing: A survey," *New Generation Computing*, vol. 27, no. 3, pp. 177–196, 2009.
- [64] R. Caceres and A. Friday, "UbiComp systems at 20: Progress, opportunities, and challenges," *IEEE Pervasive Computing*, no. 1, pp. 14–21, 2011.
- [65] A. Giessmann, K. Stanoevska-Slabeva, and B. De Visser, "Mobile enterprise applications - current state and future directions," in *Proceedings of the 45th Hawaii International Conference on System Science, HICSS 2012*, IEEE, 2012, pp. 1363–1372.
- [66] A. Alkassar, S. Schulz, C. Stübke, and S. Wohlgemuth, "Securing smartphone compartments: approaches and solutions," in *ISSE 2012 Securing Electronic Business Processes*, Springer, 2012, pp. 260–268.
- [67] Y. Zhauniarovich, G. Russello, M. Conti, B. Crispo, and E. Fernandes, "MOSES: Supporting and enforcing security profiles on smartphones," *Dependable and Secure Computing, IEEE Transactions on*, vol. 11, no. 3, pp. 211–223, 2014.
- [68] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastri, "Practical and lightweight domain isolation on Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ACM, 2011, pp. 51–62.
- [69] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: A virtual mobile smartphone architecture," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ACM, 2011, pp. 173–187.
- [70] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2010.
- [71] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.
- [72] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat, "Practical domain and type enforcement for UNIX," in *Security and Privacy, 1995. Proceedings., 1995 IEEE Symposium on*, IEEE, 1995, pp. 66–77.
- [73] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat, "A domain and type enforcement UNIX prototype," *Computing Systems*, vol. 9, no. 1, pp. 47–83, 1996.
- [74] The Australian Signals Directorate, "Risk management of enterprise mobility including bring your own device," Australian Government, Department of Defence, Intelligence and Security, Tech. Rep., 2013.

- [75] G. M. Johnson, "Towards shrink-wrapped security: A taxonomy of security-relevant context," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications, PerCom '09*, IEEE Computer Society, 2009, pp. 1–2.
- [76] A. Evesti and S. Pantsar-Syv niemi, "Towards micro architecture for security adaptation," in *Proceedings of the European Conference on Software Architecture: Companion Volume, ECSA '10*, ACM, 2010, pp. 181–188.
- [77] V. Jovanovikj, D. Gabrijel i , and T. Klobu ar, "A conceptual model of security context," *International Journal of Information Security*, vol. 13, no. 6, pp. 571–581, 2014.
- [78] H. F. Nissenbaum, *Privacy in context: Technology, policy, and the integrity of social life*. Stanford Law & Politics, 2010.
- [79] Y. Engestr m *et al.*, "Activity theory and individual and social transformation," *Perspectives on activity theory*, pp. 19–38, 1999.
- [80] M. Strimpakou, I. Roussaki, C. Pils, M. Angermann, P. Robertson, and M. Anagnostou, "Context modelling and management in ambient-aware pervasive environments," in *Location- and Context-Awareness*, T. Strang and C. Linnhoff-Popien, Eds., ser. Lecture Notes in Computer Science, vol. 3479, Springer Berlin Heidelberg, 2005, pp. 83–94.
- [81] K. Frank, N. Kalatzis, I. Roussaki, and N. Liampotis, "Challenges for context management systems imposed by context inference," in *Proceedings of the International Workshop on Managing Ubiquitous Communications and Services, MUCS '09*, ACM, 2009, pp. 27–34.
- [82] B. P. Rocha, D. N. Costa, R. A. Moreira, C. G. Rezende, A. A. Loureiro, and A. Boukerche, "Adaptive security protocol selection for mobile computing," *Journal of Network and Computer Applications*, vol. 33, no. 5, pp. 569–587, 2010.
- [83] A. Saxena, M. Lacoste, T. Jarboui, U. L cking, and B. Steinke, "A software framework for autonomic security in pervasive environments," in *Information Systems Security*, P. McDaniel and S. Gupta, Eds., ser. Lecture Notes in Computer Science, vol. 4812, Springer Berlin Heidelberg, 2007, pp. 91–109.
- [84] G. R. Ganger, "Authentication confidences," in *Proceedings of the Workshop on Hot Topics in Operating Systems, HOTOS '01*, IEEE Computer Society, 2001, p. 169.
- [85] R. Hulsebosch, M. Bargh, G. Lenzi, P. Ebben, and S. Iacob, "Context sensitive adaptive authentication," in *Smart Sensing and Context*, G. Kortuem, J. Finney, R. Lea, and V. Sundramoorthy, Eds., ser. Lecture Notes in Computer Science, vol. 4793, Springer Berlin Heidelberg, 2007, pp. 93–109.
- [86] R. Zhang, F. Giunchiglia, B. Crispo, and L. Song, "Relation-based access control: An access control model for context-aware computing environment," *Wireless Personal Communications*, vol. 55, no. 1, pp. 5–17, 2010.
- [87] A. Sabzevar and J. Sousa, "Chameleon: A model of identification, authorization and accountability for ubicomp," in *Ubiquitous Intelligence and Computing*, C.-H. Hsu, L. Yang, J. Ma, and C. Zhu, Eds., ser. Lecture Notes in Computer Science, vol. 6905, Springer Berlin Heidelberg, 2011, pp. 326–339.
- [88] M. Lacoste, G. Privat, and F. Ramparany, "Evaluating confidence in context for context-aware security," in *Ambient Intelligence*, Springer, 2007, pp. 211–229.

- [89] T. Buchholz and M. Schiffers, “Quality of context: What it is and why we need it,” in *In Proceedings of the 10th Workshop of the OpenView University Association: OVUA’03*, 2003.
- [90] B. W. Lampson, “Protection,” *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.
- [91] W. E. Boebert and R. Y. Kain, “A practical alternative to hierarchical integrity policies,” *NIST SPECIAL PUBLICATION SP*, A–10, 1989.
- [92] S. Ceri, G. Gottlob, and L. Tanca, “What you always wanted to know about Datalog (and never dared to ask),” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 1, no. 1, pp. 146–166, 1989.
- [93] P. Bonatti, C. Galdi, and D. Torres, “ERBAC: Event-driven RBAC,” in *Proceedings of the 18th ACM symposium on Access control models and technologies*, ACM, 2013, pp. 125–136.
- [94] K. J. Biba, “Integrity considerations for secure computer systems,” DTIC Document, Tech. Rep., 1977.
- [95] J. H. Jafarian and M. Amini, “CAMAC: A context-aware mandatory access control model,” *The ISC International Journal of Information Security*, vol. 1, no. 1, 2015.
- [96] V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “Access control in BitTorrent P2P networks using the Enhanced Closed Swarms protocol,” in *Proceedings of the International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2011)*, IARIA, 2011, pp. 97–102.
- [97] V. Jovanovikj, “Towards a novel security model for development of security policies,” in *Proceedings of the UNITE 2nd Doctoral Symposium: R & D in Future Internet and Enterprise Interoperability, Sofia, Bulgaria*, R. Jardim-Gonçalves and K. Stefanova, Eds., 2012, pp. 35–39.
- [98] V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “Security policy model for ubiquitous social systems,” in *Modeling and Using Context: Proceedings of the 10th International and Interdisciplinary Conference, (CONTEXT 2017)*, P. Brézillon, R. Turner, and C. Penco, Eds., ser. Lecture Notes in Artificial Intelligence, vol. 10257, Springer, 2017, pp. 302–314.
- [99] P. A. Bonatti, “Datalog for security, privacy and trust,” in *Datalog Reloaded*, Springer, 2011, pp. 21–36.
- [100] M. Y. Becker and P. Sewell, “Cassandra: Distributed access control policies with tunable expressiveness,” in *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, IEEE, 2004, pp. 159–168.
- [101] P. Bonatti and D. Olmedilla, “Driving and monitoring provisional trust negotiation with metapolicies,” in *null*, IEEE, 2005, pp. 14–23.
- [102] N. Peter Loscocco, “Integrating flexible support for security policies into the Linux operating system,” in *Proceedings of the FREENIX track: USENIX Annual Technical Conference*, 2001.
- [103] S. Smalley and R. Craig, “Security Enhanced (SE) Android: Bringing flexible MAC to Android,” in *NDSS*, vol. 310, 2013, pp. 20–38.
- [104] *Android*, <http://www.android.com>, 2016.
- [105] W. Stallings, *Operating systems: Internals and design principles*. Prentice-Hall, 1998.

- [106] W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android security," *IEEE Security & Privacy*, no. 1, pp. 50–57, 2009.
- [107] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ACM, 2010, pp. 328–332.
- [108] M. Conti, B. Crispo, E. Fernandes, and Y. Zhauniarovich, "Crêpe: A system for enforcing fine-grained context-related policies on Android," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 5, pp. 1426–1438, 2012.
- [109] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. Android and Mr. Hide: Fine-grained permissions in Android applications," in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, ACM, 2012, pp. 3–14.
- [110] R. Xu, H. Sa'di, and R. Anderson, "Aurasium: Practical policy enforcement for Android applications," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 539–552.
- [111] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky, "Appguard—fine-grained policy enforcement for untrusted Android applications," in *Data Privacy Management and Autonomous Spontaneous Security*, J. Garcia-Alfaro, G. Lioudakis, N. Cuppens-Boulahia, S. Foley, and M. W. Fitzgerald, Eds., Springer Berlin Heidelberg, 2014, pp. 213–231.
- [112] S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi, "ASM: A programmable interface for extending Android security," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 1005–1019.
- [113] J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security modules: General security support for the Linux kernel," in *USENIX Security Symposium*, 2002.
- [114] V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, "Context modelling in context-aware security systems," in *Proceedings of the 20th International Electrotechnical and Computer Science Conference ERK 2011, Portorož, Slovenia*, B. Zajc and A. Trost, Eds., IEEE, 2011, pp. 51–54.
- [115] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, 2011.

Bibliography

Publications Related to the Thesis

Journal Article

- V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “A conceptual model of security context,” *International Journal of Information Security*, vol. 13, no. 6, pp. 571–581, 2014, **IF: 1.279**.

Conference Paper

- V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “Security policy model for ubiquitous social systems,” in *Modeling and Using Context: Proceedings of the 10th International and Interdisciplinary Conference, (CONTEXT 2017)*, P. Brézillon, R. Turner, and C. Penco, Eds., ser. Lecture Notes in Artificial Intelligence, vol. 10257, Springer, 2017, pp. 302–314.
- V. Jovanovikj, “Towards a novel security model for development of security policies,” in *Proceedings of the UNITE 2nd Doctoral Symposium: R & D in Future Internet and Enterprise Interoperability, Sofia, Bulgaria*, R. Jardim-Gonçalves and K. Stefanova, Eds., 2012, pp. 35–39.
- V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “Access control in BitTorrent P2P networks using the Enhanced Closed Swarms protocol,” in *Proceedings of the International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2011)*, IARIA, 2011, pp. 97–102.
- V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “Access control in BitTorrent P2P networks using the Enhanced Closed Swarms protocol,” in *Proceedings of the 3rd Jožef Stefan International Postgraduate School Students Conference, Ljubljana, Slovenia*, D. Petelin, Ed., 2011, pp. 68–73.
- V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “Context modelling in context-aware security systems,” in *Proceedings of the 20th International Electrotechnical and Computer Science Conference ERK 2011, Portorož, Slovenia*, B. Zajc and A. Trost, Eds., IEEE, 2011, pp. 51–54.
- V. Jovanovikj, D. Gabrijelčič, and T. Klobučar, “Security issues of the Future Internet with focus on access control in the content delivery platforms,” in *Proceedings of the 19th International Electrotechnical and Computer Science Conference ERK 2010, Portorož, Slovenia*, B. Zajc and A. Trost, Eds., IEEE, 2010, pp. 151–154.

Biography

Vladimir Jovanovikj was born on February 16, 1985 in Strumica, Macedonia. He finished secondary school at the local gymnasium in 2003. He continued his education at the Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and Methodius University in Skopje, Macedonia. During his studies, he majored in computer science, information technology and automation. He obtained his university degree with honors in 2008. His graduation thesis covered topics from optimization algorithms and data mining, and their usage in medicine.

While still at university, Vladimir started working as a software developer at Seavus, an international company with premises in Skopje. Within Seavus, he was responsible for maintaining the billing systems of leading telecom providers in Europe, such as Hutchison 3G, Vodafone and Tele2. In particular, he analyzed and fixed existing bugs and problems, as well as designing and developing new business requirements. After almost two years in this role, he decided to pursue his aspiration towards a doctoral degree.

Vladimir enrolled as a doctoral student at the Jožef Stefan International Postgraduate School in 2009. He was awarded a scholarship for doctoral studies and research training by the Slovene Research Agency, after which he joined the Laboratory for Open Systems and Networks (E5) at the Jožef Stefan Institute as a young researcher. During his PhD journey, he has been working under the supervision of Asst. Prof. Dr. Tomaž Klobučar. Vladimir's research focuses on security and privacy issues within complex modern environments such as ubiquitous computing and peer-to-peer systems. In particular, this involves the specification and enforcement of security policies for access control. He is primarily interested in discovering contextual information which is important for security and using this in the development of context-aware security solutions. He has participated in two research projects: P2P Next (FP7 EU project) and Security and Trust in the New Generation of P2P networks (a national project). He also received the Best Papers award for his work at the International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2011).

