



UNIVERSIDAD DE MÁLAGA

TESIS DOCTORAL

A TRUST-BY-DESIGN FRAMEWORK FOR THE INTERNET OF THINGS

Author:

Davide Ferraris

PROGRAMA DE DOCTORADO EN TECNOLOGÍAS INFORMÁTICAS

PHD IN COMPUTER SCIENCE

UNIVERSITY OF MALAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

2022

Advisors:

María del Carmen Fernández-Gago

Associate Professor at University of Malaga

and


Francisco Javier López Muñoz

Full Professor at University of Malaga



UNIVERSIDAD
DE MÁLAGA

AUTOR: Davide Ferraris

 <https://orcid.org/0000-0003-3035-3774>

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga



Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:

<http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>

Cualquier parte de esta obra se puede reproducir sin autorización

pero con el reconocimiento y atribución de los autores.

No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): riuma.uma.es



DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña Davide Ferraris

Estudiante del programa de doctorado en Tecnologías Informáticas de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: A Trust-by-Design Framework for the Internet of Things

Realizada bajo la tutorización de Javier Lopez Muñoz y dirección de Javier Lopez Muñoz y Carmen Fernandez-Gago (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 16 de Mayo de 2022

Fdo.: Davide Ferraris Doctorando/a	Fdo.: Javier Lopez Muñoz Tutor/a
Fdo.: Carmen Fernandez Gago y Javier Lopez Muñoz Director/es de tesis	



D. Fco. Javier López Muñoz, Catedrático de Universidad del área de Ingeniería Telemática del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, y Dña. Carmen Fernandez Gago, Profesora Contratada Doctora del Departamento de Matemática Aplicada de la Universidad de Málaga.

CERTIFICAN QUE:

Don Davide Ferraris, Ingeniero en Informática, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, bajo nuestra dirección, el trabajo de investigación correspondiente a su Tesis Doctoral, titulada:

A Trust-by-Design Framework for the Internet of Things

Revisado el presente trabajo, estimamos que puede ser presentado al tribunal que ha de juzgarlo, y autorizamos la presentación de esta Tesis Doctoral en la Universidad de Málaga.

Además autorizamos la lectura de la Tesis y declaramos que las publicaciones que avalan la Tesis no han sido utilizadas en tesis anteriores.

Málaga, a 16 de Mayo de 2022

Fdo: Dña. Carmen Fernandez Gago
Profesora Contratada Doctora
Departamento de Matemática Aplicada

Fdo: D. Fco. Javier López Muñoz
Catedrático de Universidad
Area de Ingeniería Telemática

Table of Contents

Table of Contents	iii
List of Figures	viii
List of Tables	xii
1 Introduction	9
1.1 Research Scope	10
1.2 Goals of the thesis	11
1.3 Outline of this Dissertation	13
1.4 Publications and Funding	14
2 State of the Art	17
2.1 Trust	18
2.1.1 Trustworthiness	20
2.1.2 Characteristics of trust	21
2.1.3 Trust and Reputation	23
2.1.4 Trust connected properties	25
2.1.5 Trust Management, Metrics and Models	32
2.2 System Development Life Cycle (SDLC)	36
2.2.1 Requirements Engineering	36
2.2.2 Modeling Languages	41
2.3 Internet of Things	43

2.3.1	Trust in the IoT	47
2.3.2	Frameworks for Trust and IoT	53
2.4	Conclusion	57
3	The K-Model: A Model for the Development of Trust in the IoT	59
3.1	General Overview of the K-Model	60
3.2	Need	62
3.3	Requirements - TrUStAPIS: A Method for IoT Requirements Elicitation	64
3.3.1	Trust Requirements	70
3.3.2	Usability Requirements	72
3.3.3	Security Requirements	73
3.3.4	Availability Requirements	74
3.3.5	Privacy Requirements	75
3.3.6	Identity Requirements	76
3.3.7	Safety Requirements	78
3.3.8	Requirements Database Refinement	79
3.3.9	Step-by-Step Methodology	80
3.3.10	TrUStAPIS methodology discussion	82
3.4	Model Phase - Trust Model-driven Approach	83
3.4.1	Use Case Diagram	84
3.4.2	Class Diagram	85
3.4.3	Activity Diagram	87
3.4.4	Sequence Diagram	89
3.4.5	State Machine Diagram	91
3.4.6	Requirement Diagram	92
3.4.7	Context Diagram	96
3.4.8	Traceability Diagram	98
3.4.9	Methodology	99
3.5	Development	101
3.5.1	Top-Down approach	102

3.5.2	Bottom-Up approach	103
3.5.3	Trusted Block Development	104
3.5.4	Implementation Approaches	105
3.6	Verification and Validation	106
3.6.1	Verification	106
3.6.2	Validation	110
3.7	Utilization	114
3.7.1	Adaptive Trust Model	116
3.7.2	Trust Estimation - Join, Stay and Leave	118
3.8	Conclusion	122
4	Transversal Activities of the K-Model	123
4.1	General Overview	124
4.2	Traceability	125
4.3	Documentation	125
4.4	Metrics	127
4.5	Gates	129
4.6	Threat Analysis	131
4.7	Risk Management	133
4.8	Decision-Making	138
4.8.1	Analytic Hierarchy Process (AHP)	138
4.8.2	Methodology: Pairwise Ordination Method (POM)	141
4.8.3	Procedure	143
4.8.4	POM vs AHP	147
4.9	Conclusion	148
5	Implementation of the Framework in an IoT Scenario	149
5.1	Need	150
5.2	Requirements	154
5.2.1	Step 1	154

5.2.2	Step 2	155
5.2.3	Step 3	163
5.2.4	Step 4	164
5.2.5	From Step 4 to Step 1	166
5.2.6	Decision Making	169
	Goal	172
	Context	173
	Stakeholders	173
	Vendors	174
	Customers	174
	Faster	175
	Cheaper	175
	Traceability	176
	Connected Requirements	176
	Connected Needs	176
	TrUStAPIS	177
	Availability	177
	Privacy	178
	Identity	178
	Final priority	178
5.3	Model	180
5.3.1	Use Case Diagram - UCD1	180
5.3.2	Class Diagram - CD1	182
5.3.3	Activity Diagram - AD1	185
5.3.4	Sequence Diagram - SD1	187
5.3.5	State Machine Diagram - SMD1	188
5.3.6	Requirement Diagram - RD1	189
5.3.7	Context Diagram - XD1	190
5.3.8	Traceability Diagram - TD1	192
	Diagrams Table	196

5.4	Development	199
5.4.1	Top-Down approach	200
5.4.2	Bottom-Up approach	201
5.4.3	Block Development	202
5.4.4	Metrics	204
5.5	Verification and Validation	207
5.5.1	Verification	207
5.5.2	Validation	210
5.6	Utilization	213
5.6.1	Join	215
5.6.2	Stay	217
5.6.3	Threath Analysis	218
5.7	Conclusion	221
6	Conclusions, Future Works and Future Lines of Research	223
6.1	Conclusions	224
6.2	Future work	230
6.3	Future Lines of Research	231
7	Resumen en español	235
7.1	Introducción	235
7.2	Objetivos de la tesis	237
7.3	Diseño de un Marco de Confianza	239
7.3.1	Modelo K	240
7.3.2	Actividades Transversales	248
7.3.3	Futuras líneas de investigación	250

List of Figures

2.1	Previsions of total number of connected device up to 2025 [8]	44
3.1	K-Model: with the <i>Transversal Activities</i> it compounds our framework	60
3.2	Requirements composition	65
3.3	Conceptual Model for TrUStAPIS	67
3.4	TrUStAPIS - JSON template to elicit requirements	68
3.5	Requirements Relationships	80
3.6	Requirements elicitation: step-by-step methodology	81
3.7	Use Case Diagram example	85
3.8	Class Diagram example	86
3.9	Activity Diagram example	89
3.10	Sequence Diagram example	90
3.11	State Machine Diagram example	93
3.12	Requirement Diagram example	95
3.13	Context Diagram example	97
3.14	Traceability Diagram example	98
3.15	Model-driven approach: step-by-step methodology	100
3.16	Functional Domain Breakdown Structure (FDBS)	103
3.17	Bottom-Up Approach (Context)	104
3.18	Block Development (BD): each block is a <i>trust island</i>	104
3.19	Step-by-step methodology	105
3.20	Verification	109
3.21	Test Verification: step-by-step methodology	109

3.22	Validation	113
3.23	Test Validation: step-by-step methodology	114
3.24	Adaptive Trust Model: Join Decision	118
3.25	Joining a network	119
3.26	Adaptive Trust Model: Stay Decision	120
3.27	Leaving a network	121
4.1	Requirements Relationships	135
4.2	General AHP model	140
4.3	Example: ordered branch-tree	144
4.4	Example: ordered and weighted branch-tree	146
5.1	Smart Cake Machine and its relationships with the other IoT entities	151
5.2	Hierarchy Levels of the Segregated Architecture	152
5.3	JSON code part 1	155
5.4	JSON code part 2	158
5.5	JSON code part 3	158
5.6	JSON code part 4	159
5.7	JSON code part 5	161
5.8	JSON code part 6	162
5.9	JSON code part 7	163
5.10	JSON code part 8	164
5.11	Traceability between requirements	167
5.12	JSON code for IDNT01.2	168
5.13	POM model related to our use case scenario	172
5.14	POM model related to our use case scenario	179
5.15	UCD1 - User data	181
5.16	CD1 - SCM and SM service classes	183
5.17	AD1 - Securing user data	186
5.18	SD1 - User, SCM, SF, SH and SM interactions.	187
5.19	SMD1 - Upload a recipe into the SCM	189
5.20	RD1 - Trust, Identity and Privacy Requirements	190

5.21	XD1 - Contexts: Cook, Interaction, Recipes, User Data	191
5.22	TD1 - Traceability among diagrams	192
5.23	Database Chart	194
5.24	Traceability Table	195
5.25	Activity Diagrams Table	196
5.26	Requirement Diagrams Table	197
5.27	Traceability Table (Extended)	198
5.28	Graphical view of the Traceability Table (Extended)	199
5.29	Traceability Table related to SMD4	199
5.30	FDBS - Use Case: Smart Cake Machine	200
5.31	Bottom-Up Approach - Use Case: Smart Cake Machine	202
5.32	Block Development (BD) - Block Cook	203
5.33	JSON code for USAB01	208
5.34	JSON code for TRST01	212
5.35	Smart Home: Segregated Trust Architecture	214
6.1	Phases, Input and Output of the whole Framework	229
7.1	Modelo K	239
7.2	Modelo conceptual - TrUStAPIS	242
7.3	TrUStAPIS - plantilla JSON para obtener requisitos	243

List of Tables

2.1	Characteristics of trust	23
2.2	Sub-properties of trust extracted from [70]	26
2.3	Sub-properties of trust extracted from [124]	28
2.4	Trustor and trustee properties of trust extracted from [160]	31
2.5	Search criteria about trust and IoT and related works analysed	52
2.6	Frameworks Properties	56
3.1	Requirement specification	79
3.2	Context Diagram - Database template	97
3.3	Context Diagram Example - Database view	97
3.4	Traceability Diagram - Database view	99
3.5	Traceability Diagram Example - Database view	99
3.6	Validation tests	112
4.1	Metrics table	129
4.2	Gates table	132
4.3	Threats table	133
4.4	Likelihood	136
4.5	Severity	136
4.6	Detectability	136
4.7	Values of Risks	137
4.8	The fundamental scale for pairwise comparisons [140]	140
4.9	POM number of comparisons	147

5.1	Requirements elicited using TrUsTAPIS - part 1	165
5.2	Requirements elicited using TrUsTAPIS - part 2	165
5.3	Requirements elicited using TrUsTAPIS - part 3	166
5.4	Security requirement: SEC01	166
5.5	Identity requirement: IDNT01	166
5.6	Identity sub-requirement: IDNT01.1	167
5.7	Identity sub-requirement: IDNT01.2	168
5.8	Identity requirement: IDNT01	168
5.9	Trust requirement: TRST02	180
5.10	XD1 - Database view	192
5.11	TD1 Example - Database view	193
5.12	Requirements and model connections.	208
5.13	Requirements and needs connections.	211

List of Acronyms

AD Activity Diagram

AHP Analytic Hierarchy Process

BD Block Development

BLE Bluetooth Low Energy

BT Backward Traceability

BYOD Bring Your Own Device

C Context

CA Certification Authority

CD Class Diagram

CI Consistency Index

D Detectability

DB Database

DDoS Distributed Denial of Service

DF Device Functionality

DM Decision Maker

DMZ Demilitarized Zone

FBS Functional Breakdown Structure

FDBS Functional Domain Breakdown Structure

FIoT Fog based IoT

FT Forward Traceability

GDPR General Data Protection Regulation

GR Gate Reviews

HMI Human Machine Interaction

H2H Humans to Humans

HG House Guest

HM House Member

HO House Owner

IoT Internet of Things

IT Inner Traceability

IVV Independent Verification and Validation

JSON JavaScript Notation Object

L Likelihood

MCDA Multi Criteria Decision Analysis

M2M Machine to Machine

MU Malicious User

NeCS Meaning

PKI Public Key Infrastructure

POM Pairwise Ordination Method

R Role

RA Reputation Agent

RD Requirement Diagram

RE Requirements Engineering

S Severity

Sc Score

SCM Smart Cake Machine

SD Sequence Diagram

SDLC Software Development Life Cycle

SDLC System Development Life Cycle

SF Smart Fridge

SH Smart Hub

SIoT Social Internet of Things

SM Smart Supermarket

SMD State Machine Diagram

SysML System Modeling Language

T Target

TD Traceability Diagram

TM Trust Metric

TTP Trusted Third Party

UCD Use Case Diagram

UML Unified Modeling Language

WBS Work Breakdown Structure

XD Context Diagram

XI Context Importance

Abstract

The Internet of Things (IoT) is an environment where interconnected entities can interact and can be identifiable, usable, and controllable via the Internet. However, in order to interact among them, such IoT entities must trust each other. Trust is difficult to define because it concerns different aspects and is strongly dependent on the context. For this reason, a holistic approach allowing developers to consider and implement trust in the IoT is highly desirable. Nevertheless, trust is usually considered among different IoT entities only when they have to interact among them. In fact, without considering it during the whole System Development Life Cycle (SDLC) there is the possibility that security issues will be raised. In fact, without a clear conception of the possible threats during the development of the IoT entity, the lack of planning can be insufficient in order to protect the IoT entity. For this reason, we believe that it is fundamental to consider trust during the whole SDLC in order to carefully plan how an IoT entity will perform trust decisions and interact with the other IoT entities. To fulfill this goal, in this thesis work, we propose a trust-by-design framework for the IoT that is composed of a K-Model and several transversal activities. On the one hand, the K-Model covers the SDLC from the need phase to the utilization phase. On the other hand, the transversal activities will be implemented differently depending on the phases. A fundamental aspect that we implement in this framework is the relationship that trust has with other related domains such as security and privacy. Thus we will also consider such domains and their characteristics in order to develop a trusted IoT entity.

Keywords: Trust, Security, Internet of Things (IoT), System Development Life Cycle (SDLC), K-Model

Acknowledgments

“Per Aspera Ad Astra”

First of all, I want to thank my supervisors, Carmen Fernandez-Gago and Javier Lopez Muñoz for their support and help during my PhD journey. Then, I must thank the NeCS project and the European Commission for giving me an amazing opportunity to boost up my knowledge in a field such as Computer Security and give me the opportunity to make networking with prestigious researchers and institutes of the World. Then, I want to thank my parents Roberto and Gabriella for helping me even if we are far away. Your support made me what I’m now. I thank also my sister Fabiana, my niece Emma and my brother in law Davide, I wait for you in your beloved Malaga. A life without friends is not the same, so I want to thank all my Ventimiglia’s, Genova’s and Malaga’s friends. I cannot enumerate you all, but you have been always at my side helping me during hard periods. I want also to thank my NICS colleagues. I’ve learned a lot from you, you are an amazing group and without you I could not been the same researcher that I am now. In this adventure, I’ve been also with my PhD students colleagues from Malaga and NeCS project. It has been a pleasure to know you all. Finally, I want to thank all the other persons that I’ve meet during these years and spent days, weeks and years together, even if we are not traveling together anymore.

Thank you All, this Thesis is Mine as Yours.

CHAPTER 1

Introduction

This Chapter sets the scope of the thesis, beginning with an introduction to **Internet of Things (IoT)** and **trust**. Secondly, we discuss how IoT can benefit from trust considerations. Then, we establish the main goal of the thesis, which is to propose a way to consider trust in an IoT entity not only during its utilization or implementation but in its System or Software Development Life Cycle (SDLC). In Section 1.3, we outline the composition of this thesis work. Finally, we present the collection of the papers that we have written to cover the thesis and the funding we have received to work on and publish our results.

1.1 Research Scope

The Internet of Things (IoT) allows humans and smart entities to cooperate among them anyhow and anywhere [133]. The IoT entities developed and used by customers are growing each year, and “it is expected that there will be more than 64B IoT devices worldwide by 2025”¹. This prediction states that the IoT paradigm will define how the world will be connected.

For this reason, many opportunities will arise, but also many problems [31]. A way to mitigate them is offered by trust. In fact, an entity should interact with another only if trust is established between them. Due to the uncertainty, interoperability, and heterogeneity of IoT, achieving trust is still a challenge. Besides, considering that isolated research communities have tackled these aspects separately, a holistic approach is desirable [46].

Trust is difficult to define. It concerns different aspects and topics ranging from Philosophy to Computer Science [46], and it is strongly dependent on the context. In fact, trust “means many things to many people” [43]. This premise is most than true for IoT. In fact, IoT entities can work in several contexts, and if we consider trust in these contexts, we can enhance the protection of these devices.

Hoffman et al. [70] and Pavlidis [124], considered trust strongly dependent on other properties like security and privacy. Ferraris et al. [50] stated that these relations are even more important during an IoT entity development. In fact, as also stated by Mohammadi et al. [109], trust mechanisms can be fundamentals and require more investigation in this field. For this reason, in our opinion, it is crucial to consider trust since the initial phases of the SDLC in order to develop the trust relationships among the smart entities correctly. This approach could help to protect the smart entities and to give important rules of behaviour during the interactions with other smart entities.

In a trust relationship, there are basically two actors involved: the trustor and the trustee. The trustor is the one who actively trusts, and the trustee is the one who keeps the trust. We can state that this collaboration is necessary when the

¹<https://techjury.net/blog/internet-of-things-statistics/>

trustor needs the trustee to perform an action or fulfill a goal considering a particular context. This goal is not achievable by the trustor alone. For this reason, the trustee is needed. Trust metrics are necessary to compute a trust level that helps the trustor to decide if a trustee can be trusted [89]. This value must be computed before the two actors start the collaboration. Moreover, the trust level could change over time positively or negatively due to the right or wrong behaviour of the trustee [65].

For example, in an IoT environment, the trustor can be the user, and the trustee can be the IoT device.

However, trust, security, privacy, and other important aspects are usually considered only during the final phases of the SDLC, and this can raise issues. We want to fill this gap because we believe that it is crucial to consider trust not only during the utilization of an IoT device but also since the earliest phases of the System and Software Development Life Cycle (SDLC).

1.2 Goals of the thesis

As we stated earlier, we believe that to consider trust properly in the IoT, we need to integrate it in the IoT devices not only during the implementation and utilization of such devices but also in the whole SDLC of an IoT entity. In fact, so far, there are no such approaches that cover the whole SDLC with trust, but only a part of it. For this reason, our aim is to redefine the approaches and tools that help developers consider trust in each phase: “from cradle to grave”.

Both in the System Development Life Cycle [67] and Software Development Life Cycle [106], one of the first phases is related to requirements engineering. In fact, collecting requirements in the early phases of the SDLC is an important task that brings benefits to the following phases and avoids problems that could happen in later phases. Developers usually elicit the requirements following stakeholders’ needs. The stakeholders are persons or companies having an interest in the system or software developed.

Existing requirements languages have been widely used with the introduction of Goal-Oriented methodologies [21, 100, 111, 163], but they have not been developed

for IoT and do not consider trust concerning other security domains. Similarly, trust and related domains such as security, identity, usability, and privacy were not appropriately considered in the first phases of SDLC [120]. On the contrary, to guarantee trust, it is important to consider other domains related to it, as Hoffman [70] and Pavlidis [124] stated. Following this premise, Rios et al. [132] have proposed a work considering privacy in trust negotiation, and Gago et al. [46] moved forward considering both identity and privacy connected to trust in the IoT field.

We aim to continue in this direction, considering trust-related domains holistically in IoT and considering requirements engineering as a crucial in our framework to ensure trust in an IoT entity during the whole SDLC.

After the requirements elicitation phase, there is the modeling phase where both UML [137] and SysML [56] are widely used by developers. In fact, these diagrams have been created in order to explore the different functionalities of a generic software/system under development. Anyhow, these original modeling languages had no features to implement security, privacy, or trust. For this reason, it is necessary to define them in order to help the developers properly model trust and related domains.

Moreover, during the development of an IoT entity, the developers can consider several approaches to perform this crucial task. A widely used approach is the so-called top-down. It is basically a way to consider the problem starting from a general perspective to a specific one. Moreover, the top-down approach can be used even for software development through a Functional Breakdown Structure (FBS) or a Work Breakdown Structure (WBS) [77]. However, in our case, it is important to consider not only the functionalities but also their connections to the domains such as trust and security in order to divide and perform the analysis according to their scope.

On the contrary, the bottom-up approach starts from a specific viewpoint to a general overview of the system. It is a method used particularly in software engineering [55, 77], but it can also be used to develop IoT infrastructures [129]. In our paper, we move forward and consider it according to the IoT entities' different contexts and domains.

However, in our opinion, these two approaches alone are not enough for the development of a trusted IoT entity. In fact, it is essential to highlight that an IoT

entity is composed of software, and an effective way to develop the code is following a finite state approach as stated by [156], and [23]. This approach is even more critical in an environment such as the IoT, where usually all the functionalities are performed separately and following a step-by-step process. In this work, we will extend this approach in addition to the bottom-up and top-down approaches.

Then, we can state that verification and validation are two fundamental phases to finish the development of an IoT entity. In fact, through verification is possible to say that *the entity has been built in the right way* that means that the functionalities are working as expected. On the other hand, validation means that *the right entity has been built*. In this case, we can declare that the IoT entity has been developed as it was intended for the originated need.

1.3 Outline of this Dissertation

This thesis is structured as follows.

In chapter 2, we present the State of the Art related to trust considering its definition and characteristics. Then we compare trust to its related properties, and we discuss trust management implementations. Then, we present two essential phases of the SDLC: requirements and modeling. In this part, we discuss which requirements elicitation techniques and model languages have been proposed and extended in previous works. Then, we consider IoT and its connection with trust.

In chapter 3, we propose the K-Model, which is part of our trust-by-design framework. The first phase of the K-Model concerns needs, where the purpose of the IoT entity to be developed is defined. Then, there is the requirements elicitation phase, where we propose a method to elicit requirements related to trust and its related domains (security, privacy, usability, availability, identity, and safety). Then, we define a JavaScript Notation Object² (JSON) template to help developers elicit the requirements and a conceptual model illustrating all the elements that must be taken into consideration during this process. After this critical phase, there is the model phase, where we present a model-driven approach extending UML and SysML in order to

²<https://www.json.org>

implement trust and related domains in the SDLC of an IoT device. Once these two essential phases have been covered, there is the fourth phase: the development phase. Here, we propose several approaches useful to develop the entities according to the outputs of the previous phases. Then, verification and validation must be performed in order to complete the flow of the SDLC that will end in the last phase: utilization.

In chapter 4, we present the seven transversal activities, which with the K-Model compound our proposed framework. These activities are traceability, documentation, metrics, gates, threat analysis, risk management, and decision-making. Traceability is essential for the whole framework because it connects the phases of the K-Model among them. Moreover, it is a crucial tool in requirements and model phases. Documentation is very important, especially in the first phase, where all the needed aspects of the desired IoT entity must be collected in order to elicit the proper requirements. Metrics are important for verifying the requirements and setting of the IoT entity's functionalities. The gates are related to the continuation of the flow from one phase to the following one. Only if the previous phase has been completed, it is possible to proceed. Then, Threat Analysis and Risk Management are fundamental activities that must be considered in order to prevent hazards or attacks on users and devices. Finally, the decision-making process is useful in order to solve conflicts among requirements or to perform trust decisions during the utilization phase.

In chapter 5, we propose a use case scenario in order to present how to implement both the phases of the K-Model and their related transversal activities. Thus, we will present how a new IoT entity will be developed, considering trust and related domains during the whole SDLC.

Finally, there is chapter 6, in which we summarize the insights and the discussions of our thesis work. Moreover, we present our future works and possible future lines of research.

1.4 Publications and Funding

The work presented in this dissertation has been internationally awarded in security conferences, meetings, and journals, where experts have provided their valuable

insights and thoughts.

1. Davide Ferraris, Carmen Fernandez-Gago, and Javier Lopez, “A Trust-by-Design Framework for the Internet of Things” in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1-4, February 2018.
2. Davide Ferraris, Carmen Fernandez-Gago, Joshua Daniel, and Javier Lopez, “A Segregated Architecture for a Trust-based Network of Internet of Things” in *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1-6, January 2019.
3. Davide Ferraris, Daniel Bastos, Carmen Fernandez-Gago, Fadi El-Moussa, and Javier Lopez, “An Analysis of Trust in Smart Home Devices” in *2019 20th World Conference on Information Security Applications (WISA Workshop)*, pp. 1-8, August 2019.
4. Davide Ferraris, and Carmen Fernandez-Gago, “TrUSStAPIS: a Trust Requirements Elicitation Method for IoT” in *2020 Springer International Journal of Information Security*, Volume 19, Number 1, pp. 111-127, February 2020.
5. Davide Ferraris, Daniel Bastos, Carmen Fernandez-Gago, and Fadi El-Moussa, “A trust model for popular smart home devices” in *2020 Springer International Journal of Information Security*, pp. 1-17, August 2020.
6. Davide Ferraris, Carmen Fernandez-Gago, and Javier Lopez, “A model-driven approach to ensure trust in the IoT” in *2020 Springer Human-centric Computing and Information Sciences*, pp. 1-29, December 2020.
7. Davide Ferraris, Carmen Fernandez-Gago, and Javier Lopez, “POM: A Trust-based AHP-like Methodology to Solve Conflict Requirements for the IoT” in *Book: Collaborative approaches for Cyber Security in cyber-physical systems. Springer Series title: Advanced Sciences and Technologies for Security Applications, Accepted.*

8. Davide Ferraris, Carmen Fernandez-Gago, and Javier Lopez, “Novel Approaches for the Development of Trusted IoT Entities” in *2022 37th International Conference on ICT Systems Security and Privacy Protection (IFIP SEC)*, Accepted, June 2022.
9. Davide Ferraris, Carmen Fernandez-Gago, and Javier Lopez, “Verification and Validation Methods for a Trust-by-Design Framework” in *2022 36th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSEC)*, Submitted.

This thesis has been funded by the European Union’s Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 675320 (NeCS project), by the Spanish Ministry of Economy and FEDER through the project PRECISE(TIN2014-54427-JIN), by the EU project H2020-MSCA-RISE-2017 under grant agreement No 777996 (Sealed-GRID), by the CyberSec4Europe project under SU-ICT-03 program grant agreement 830929, and the SMOG project founded by the Spanish Ministry of Economy and Competitiveness (TIN2016-79095-C2-1-R).

CHAPTER 2

State of the Art

This chapter will describe how trust has been defined in state of the art during the years. We will take into consideration different topics and how different authors have defined it. Trust has some characteristics, and it is strongly related to other domains such as privacy and security. Moreover, we have to consider how trust management is performed and which trust models have been proposed over the years. Then, we will introduce requirements engineering and the existing modeling languages. Furthermore, our aim is how to consider trust in the IoT. Therefore, we will discuss IoT and how trust can be considered in it. Finally, we will present the works related to trust management in the IoT.

2.1 Trust

“Trust is a common phenomenon” [98], but it is also a difficult concept to define “because it is a multidimensional, multidisciplinary and multifaceted concept” [160]. Trust is defined in British English by the Cambridge Dictionary as “to believe that someone is good and honest and will not harm you, or that something is safe and reliable”¹. Analysing this definition, there is a distinction respect to people (“someone”) and objects (“something”). The former case refers to the goodness and honesty of the person we trust and that he/she will not harm us. In the latter case, we refer to the object implying that it is safe and reliable and basically that its utilization will not harm us and it will work as we have expected. Thus, we can state that this definition is general, and it can give an idea that trust is strongly related to the context, and it is hard to define.

Indeed, in state of the art, there are many definitions of trust applicable to different aspects. Erickson [43] stated that “trust means many things to many people”. Accordingly, with this definition, we can understand why it is hard to define and explain what trust is. Moreover, many fields of study, such as Sociology, Psychology, Philosophy, and Information Technology have to deal with trust differently. For this reason, McKnight [105] stated that “Trust has been defined in so many ways by so many different researchers across disciplines that a typology of the various types of trust is sorely needed”.

Following this premise, giving a meaning to trust is a challenge that many authors have tackled in the past years.

Mayer et al. [103] defined trust as a “willingness to be vulnerable to another party”.

McKnight and Chervany [104] explained that trust intention is “the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even though negative consequences are possible”.

Gambetta [57] affirmed that “trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent

¹<https://dictionary.cambridge.org/it/dizionario/inglese/trust>

or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action”.

Mui et al. [115] defined trust as “a subjective expectation an agent has about another’s future behavior based on the history of their encounters”.

For Ruohomaa et al. [138] “trust is the extent to which one party is willing to participate in a given action with a given partner, considering the risks and incentives involved”.

Hoffman [70] defined trust “as the expectation that a service will be provided or a commitment will be fulfilled”.

Jøsang [78] stated that “trust is a personal and subjective phenomenon that is based on various factors or evidence” and also that “trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends”.

Finally, Agudo et al. [4] defined that trust is related to “the level of confidence that an entity participating in a network system places on another entity of the same system for performing a given task”.

However, by analysing all these definitions, we can state that trust is strictly dependent on the actors involved in the trust interaction. Typically, they are at least two entities known as the trustor, and the trustee [4, 98, 112, 124].

For a trust interaction, in order to be performed, we can state that it is necessary that “the trustor trusts the trustee”. Analysing this sentence we can state that:

1. “the *trustor*” is the entity which places trust (active trust);
2. “the *trustee*” is the entity on which trust is placed (passive trust);
3. “*trusts*” is the action between the two entities.

The trust action happens when an individual (the “trustor”) requires the service of another individual (the “trustee”). Depending on the fulfillment of the action or how it is performed, the level of trust of the trustor can change positively or negatively. This means that future interactions will depend on the outcome of past interactions affecting the level of trust of the trustor.

2.1.1 Trustworthiness

Trustworthiness can be defined as a characteristic of a person [124], or something [43] that is the object of somebody's trust. In other words, it is a characteristic of the trustee.

Pavlidis [124] stated that “a trustworthy system is a system that has the capability of meeting customer trust and the capability to meet their stated, unstated, and even unanticipated needs”

McKnight and Chervany [105] defined four concepts related to trustworthiness: benevolence, competence, integrity, and predictability.

- **Benevolence:** the trustor is important for the trustee, and for this reason, he wants to act properly not to hurt him.
- **Competence:** the trustee is able to do what the trustor wants (and this can be the reason for the trustor asking help from the trustee)
- **Integrity:** the trustee is honest, and he acts to do what the trustor asks without malicious intentions.
- **Predictability:** the trustor can anticipate the behaviour of the trustee and has knowledge *a-priori* about the exchange.

According to McKnight and Chervany [105], only one of these four concepts is not enough to establish a trust relationship. In fact, if the trustee is honest but has no competence to finalize the action requested by the trustor this one does not want to establish the relationship. Thus, in this case, the trustor cannot trust the trustee to perform that action. Otherwise, if the trustee is competent but he is not honest, probably the relationship is not created because the trustor cannot trust the trustee, fearing a possible betrayal.

Hence, trustworthiness determines if someone (or something) can be trusted. The higher is the trustworthiness, the higher is the possibility to be trusted. When the desired level of trust of the trustor matches the trustworthiness of the trustee, there is no disequilibrium in the relation. The other possibilities are trusting less or

trusting more than the trustee's trustworthiness. In the first case, there is a loss of the opportunities; in the second case, there is a possible loss because the trustor is vulnerable [29, 30].

Trustworthiness is crucial for both humans and things. When we talk of trustworthiness about a thing or software, it is considered a high-quality resource [107]. Moreover, a system can be defined as trustworthy and accepted by the customers if its capability meets the stakeholder needs, not only the ones asked by them but also the ones they did not specify but are important for the system [124].

2.1.2 Characteristics of trust

According to trust, there are some characteristics defined in state of the art by several authors.

Trust can be **direct**. This property means that trust is based on direct experiences between the trustor and the trustee [18]. In this case, we can also say that trust is history-dependent.

Trust can also be **indirect**; this happens when the trustor and the trustee did not have past interactions. In this case, trust is built on the opinion and the recommendation of other entities trusted by the trustor [1]. In this case, we can refer to the possibility that trust is **transitive** [27]. Trust is conditionally transferable, "information about trust can be transmitted/received along a chain (or network) of recommendations. The conditions are often bound to the context and the trustor's objective factors" [160].

Trust is also **directed** because there is an oriented relationship between the trustor and the trustee [160]. This means that if A trusts B, we cannot be sure that B trusts A.

Trust is **not static** over time, it is not strictly time-dependent, but it can change over time. Chang et al. [24] stated that "trust builds with time". In fact, a trustor could trust the trustee about something for a period of time. However, this trust level could change later because something could happen to modify the original trust level [124].

Trust is **dynamic**, “it is non-monotonically changing with time. It may be periodically refreshed or revoked and must be able to adapt to the changing conditions of the context in which the trust decision was made” [65].

Trust is also **context-dependent** [112] and it can change depending on the purpose where it is used. Moreover, Yan et al. [160] stated that “in general, trust is a subjective belief about an entity in a particular context” and more specifically in Information Technology Abdelghani et al. [1] state that “the trust of a node i in a node j varies from one context to another”.

Trust can be **local** [1] because it depends on the considered couple of trustor and trustee (i.e., Alice and Bob) and if we consider other two couples (i.e., Alice and Charlie, and Bob and Charlie), it is possible that Alice distrust Charlie, even if Bob trusts Charlie [27].

Trust can be **global**, it “means that every node has a unique trust value in the network which can be known by all other nodes” [1].

In addition, we can state that trust can be **specific** [110, 83] or **general** [110, 83]. On the one hand, it means that trust is specific if the trustor trusts the trustee only for a specific action or service. On the other hand, trust is general if the trustor trusts the trustee generally and not only for a specific action.

Trust can be **asymmetric**, this means that two entities tied by a relationship may trust each other in different ways, so the fact that A trusts B does not imply that B should trust A [116]. This is connected to the definition of “directed”.

Trust is **subjective** because it is related to a personal opinion based on various factors (i.e., experience), and these factors can have different weights [65]. Trust is different for each individual in a particular situation [160]. On the other hand, trust can also be **objective** “such as when trust is computed based on QoS properties of a device [1]”. Furthermore, an objective parameter to compute trust is also known as **reputation**.

Trust is usually a **composite property**, “trust is really a composition of many different attributes: reliability, dependability, honesty, truthfulness, security, competence, and timeliness, which may have to be considered depending on the environment in which trust has been specified [65]”. Compositionality is an essential feature for

trust calculations [160], and every attribute could have a different weight.

Finally, trust should be **measurable**, “trust values can be used to represent the different degrees of trust an entity may have in another. [160].” This characteristic is the basis for the computation of a final trust value during trust management.

These sixteen characteristics of trust are summarized in Table 2.1.

Table 2.1: Characteristics of trust

Direct	[18]
Indirect	[1]
Transitive	[27, 160]
Directed	[160]
Not Static	[24, 124]
Dynamic	[65]
Context-dependent	[1, 112]
Local	[1, 27]
Global	[1]
Specific	[83, 110]
General	[83, 110]
Asymmetric	[116]
Subjective	[65, 160]
Objective	[1]
Composite-property	[65, 160]
Measurable	[160]

2.1.3 Trust and Reputation

Strongly related to trust, reputation is defined as “the opinion that people, in general, have about someone or something, or how much respect or admiration someone or something receives, based on past behaviour or character”².

Mui [115] stated that “reputation is defined as a perception a party creates through past actions about its intentions and norms”.

²<http://dictionary.cambridge.org/dictionary/english/reputation>

Moreover, reputation is also defined as objective trust³.

We can say that trust and reputation are connected, but they are not the same.

In fact, Jøsang [78] asserted that:

“I trust you because of your good reputation.” (1)

“I trust you despite your bad reputation.” (2)

These are two positive definitions. Hoffman stated that “Metrics must be defined to measure user trust and distrust of a system” [70] and Gambetta [57] defined distrust symmetrical respect trust. For the sake of completeness, in addition to trust, distrust, and no trust, Marsh also defined untrust and mistrust [97].

Following these definitions, we can also define two negative assertions:

“I distrust you despite your good reputation.” (3)

“I distrust you because of your bad reputation.” (4)

With these four definitions, we can understand better that reputation can be a parameter in a trust relationship (because in these definitions, the aspect of reputation is always mentioned), but not the only one that determines the computation of a trust value.

To know the reputation of someone or something, we need to keep this information somewhere by some service. This service can be a third party or any entity participating in the interactions. This structure depends on the system’s architecture; more precisely, it depends on whether the architecture is centralised or distributed. This approach is used by reputation systems (like eBay [130]). In this respect, Ruohomaa stated that “reputation systems provide essential input for computational trust as predictions on future behaviour based on a peer’s past actions. Information about these actions can also be received from other members of a reputation network who

³wiki.p2pfoundation.net/Trust_Metrics

have transacted with the peer. However, the credibility of this third-party information must be critically assessed” [139].

In Information Technology, reputation has been an enabler for internet auction exchanges. As Resnick stated that “reputation system enables trust among strangers on the Internet. This is entirely different from the reputation systems that evolved in human societies across thousands of years” [130].

2.1.4 Trust connected properties

According to Hoffman et al. [70], Lo Presti [128], and Pavlidis [124], trust is strongly dependent on other properties or domains (i.e., privacy, identity, and security). Moreover, in state of the art, there are several works about trust properties that propose a classification of them [128, 160].

Hoffman [70] proposed a trust model which considers the following properties related to trust:

- 1) Reliability & Availability
- 2) Privacy
- 3) Audit & Verification Mechanisms
- 4) Security
- 5) Usability
- 6) User Expectations

There are sub-properties for each of these properties, as we can see in Table 2.2.

The first property is a compound property. *Reliability* indicates a service is operating within its specifications [128] and *availability* means that the actions of the systems are not paused or stopped for long periods.

The second property is *privacy*. Thus, the system has to guarantee some features like granting confidentiality or anonymity. This second sub-property can create conflict with other trust properties not selected by Hoffman (i.e., accountability [117]).

For the author, *Audit & verification* is important because “since no system is perfect, the question “*Who watches the watchers?*” must be addressed. Thus, audit

Table 2.2: Sub-properties of trust extracted from [70]

Trust related property	Sub-Property/Characterisc
Reliability & Availability	<ul style="list-style-type: none"> - Vulnerability to denial of service attacks - Connection to the Internet - Quality of service/performance criteria specific to the application - Use of fault tolerance platform techniques
Privacy	<ul style="list-style-type: none"> - User anonymity - Data confidentiality
Audit & Verification Mechanisms	<ul style="list-style-type: none"> - Cryptographic methods used to verify database integrity - Manual or paper audit trails - Use of trusted agents
Security	<ul style="list-style-type: none"> - Authentication of parties in transaction - Data access control - Data integrity - Software change control procedures - Physical security
Usability	<ul style="list-style-type: none"> - Perception issues - Motor accessibility (i.e., user dexterity, physical strength requirements) - Interaction design issues
User Expectation	<ul style="list-style-type: none"> - Product reputation - Prior user experience - Knowledge of technology - Use of trusted agents

capabilities (both electronic and non-electronic, both by a single user and multiple trusted agents) will be included in the trust model. The trust model will show the effects of security and verification mechanisms on trust levels.”

Security is not only a property of trust. In fact, as Yan stated “Trust is beyond security. It is a solution for enhanced security” [160]. Security is composed of sub-properties like grants that the entities involved in a process are authenticated, have the right to access data, and are not corrupted.

Usability is essential because if a system is challenging to be used and understood, the user trust could be affected by these difficulties [70]. Furthermore, if a system is difficult to be used correctly, it is possible to misuse it. This may lead to problems and, consequently, lower the trustworthiness of the system itself [128].

User expectation is important to be considered. It is strongly related to the reputation of a system. So, the higher the reputation is, the higher the expectation about the system will be.

Pavlidis [124] considered some properties already taken into consideration by Hoffman (privacy, reliability, security, and usability). Moreover, he considered availability as a sub-property of security. In state of the art, other works consider availability as part of the security domain [114]. Besides, Pavlidis took into consideration safety and maintainability. In Table 2.3, we can see the classification made by Pavlidis.

Privacy is important for Pavlidis because nowadays the information systems can collect a considerable amount of personal information very quickly; this aspect raises a risk about the possibility that those data can be accidentally or intentionally disclosed. This situation can affect users’ trust negatively. Besides, privacy has four sub-properties: anonymity, unobservability, pseudonymity, and unlinkability. Anonymity is the ability that avoids being identified, unobservability is related to the possibility of being not detected, and pseudonymity gives the possibility to use aliases. Unlinkability can be derived by anonymity and unobservability.

The second property is *usability*. It has a substantial impact on users’ trust because it affects the system perception of a user. In fact, if the system is easy to use or memorize, the outcome will increase the trustworthiness of the system perceived by the user.

Table 2.3: Sub-properties of trust extracted from [124]

Trust related property	Sub-Property/Characterisc
Privacy	<ul style="list-style-type: none"> - Anonymity - Unobservability - Pseudonymity - Unlinkability
Usability	not defined
Reliability	not defined
Safety	not defined
Security	<ul style="list-style-type: none"> - Integrity - Confidentiality - Availability - Authentication - Authorization
Maintainability	not defined

Reliability is an attribute that is very important for system trustworthiness. In fact, reliability has been defined as “the probability that a system will perform a specified function within prescribed limits, under given environmental conditions, for a specified time” [147]. Anyhow, we consider it as a sub-set of trust.

Safety is strongly related to the physical domain. Preventing a user from being harmed will increase the system’s trustworthiness because the user will perceive the system as safe and trusted.

For the author, *security* must be taken into consideration if we do not consider trust in the design of a system. In this case, we need to make the system secure as much as possible because it is the only defense against malicious entities. On the other hand, if we consider trust, it is possible to relax some security features because the users will be trusted to perform particular activities. In this case, we have five sub-properties. Confidentiality, integrity, and availability are known as the CIA triad [45]. Authentication and authorization are significant properties, also for trust.

Finally, *maintainability* is important because it affects a system to be modified or updated. According to the author, this feature affects trust.

Another author, Lo Presti [128], defined eleven trust-related properties:

- **Source vs. Interpretation:** From raw data sources, if we manipulate them, we obtain one or more interpretations. Because of this manipulation, source data are more trusted than the interpretation.
- **Accuracy:** It represents the level of detail of information determining how precisely trust can be considered and computed. With a higher accuracy level, a user will be more confident.
- **Audit trails:** It is related to the logs of the system where all the information about actions, users, and permissions are stored. If a modification of these data occurs, it should be detected and recorded.
- **Authorization:** It is mandatory to be trusted and authorized in order to perform an action.
- **Identification:** This property allows the possibility to identify users. It is strictly connected to audit and authorization. In the case a level of privacy is needed, it is possible to provide identification under pseudonyms.
- **Reliability:** It indicates that “a service operates according to its specification”. It can also be referred to as the integrity of the data. This property is strongly related to the resilience of a system [49].
- **Availability:** This property guarantees that the services of a system are not stopped, or at least if they are not available, it will be for a short period of time.
- **Personal responsibility:** It is connected to audit and identity. Because bad actions can reduce the trust level of a system, who is liable must be recognized. This property guarantees accountability.
- **Reasoning:** It is connected to source vs. interpretation. In fact, any user can manipulate the data, but this can affect other users’ trust if “the reasoning does not appear correct”.

- **Usability:** It is an important element of trust. In fact, if a system is hard to be used, it can lead to incorrect use of it, reducing the system's trust.
- **Harm:** The author stated that “at the heart of trust is the notion of avoiding harm”. It can include any possible problem concerning a particular system (i.e., security breach, loss of personal data, privacy risk).

Some of these properties are also presented in Hoffman's work [70] (i.e., reliability, availability, audit, and usability). Others that Hoffman has identified as sub-properties of security in Lo Presti's work are distinct properties (identification, authorization).

Identification has also been considered by Mahalle et al. [95]. They have proposed identity features that are important to be taken into consideration, such as authentication and authorization.

Usability has been defined by Baharuddin et al. as “the capability of a product to be understood, learned, operated, and attractive to the users when used to achieve certain goals with effectiveness and efficiency in specific environments” [13]. In this work, the authors identified various characteristics to enhance the usability in the mobile device domain, such as effectiveness, efficiency, satisfaction, and reliability.

Even the actors involved in trust relationships (trustor and trustee) have some properties. About this, Yan et al. [160] elicited subjective and objective properties regarding the trustor and the trustee. We show them in Table 2.4.

Table 2.4: Trustor and trustee properties of trust extracted from [160]

Factors related to trustee's objective properties	Competence; ability; security; dependability; integrity; predictability; reliability; timeliness; (observed) behaviour; strength
Factors related to trustee's subjective properties	Honesty; benevolence; goodness
Factors related to trustor's objective properties	Assessment; a given set of standards; trustor's standards
Factors related to trustor's subjective properties	Confidence; expectations or expectancy; subjective probability; willingness; belief; disposition; attitude; feeling; intention; faith; hope; trustor's dependence and reliance
Context	Entailing risk; structural risk; domain of action

Thus, by analysing these definitions of trust connected properties, we can affirm that trust can be connected to other domains or properties (i.e., privacy or security). Moreover, these domains have characteristics fundamentals to define them. This is a piece of important information that we strongly consider in this thesis work.

2.1.5 Trust Management, Metrics and Models

Trust management systems have been realized to compute trust values and assist entities in interacting with other entities to decide how the interaction should be performed. Their architectures can be centralised or distributed.

More in detail, Louta et al. stated that trust management “can be conceptualized in two ways. Firstly, as a process according to which an entity becomes trustworthy for other entities. Secondly, as a process that enables the assessment of the reliability of other entities, which in turn is exploited in order to automatically adapt its strategy and behaviour to different levels of cooperation and trust” [94].

The first trust management framework was presented by Blaze [20]. His creator described *policyMaker* as a trust management system “that will facilitate the development of security features in a wide range of network services”. This framework can be considered as the most general form of trust management system. Then, Levien stated “certificates and policies can represent arbitrary computations in Turing-complete language. Applications of PolicyMaker tend to focus on the language of assertions rather than trust computations over the graph, but the fully general nature of the system allows the latter to be implemented” [88].

Ruan [135] proposed a trust management system framework that is composed of three context-dependent phases.

- **Trust Modeling:** in this phase, there is a mapping of the available trust raw data from the fields into trust metrics.
- **Trust Inference:** it focuses on propagating and aggregating the obtained trust metrics over the whole network or over the part of interest.
- **Decision Making:** is about the use of the produced trust knowledge to support decision making. This process allows the entity to decide how to act according to the data which have been collected and computed.

Hoffman [70] stated that trust management and trust metrics will be helpful for the success of new technologies distribution.

Beth et al. [18] published the first type of modern trust metric. It is composed of a set of rules used to derive the trustworthiness value of a node between 0 and 1, using subjective and objective trust.

Levien [89] defined the simplest trust metric as the following. There are three elements:

1. a designated “seed” node indicating the root of trust
2. a “target” node (T)
3. a directed graph.

This is considered as a basis for the other trust metrics. All the trust metrics contain at least these three elements. A trust metric is useful to determine whether the node T is trustworthy or not.

As Levien stated “the simplest trust metric is also the weakest. If an attacker can generate an edge from any node reachable from the seed to a node under his control, he can cause arbitrary nodes to be accepted” [89].

For more complicated metrics, the edges can contain rules, weights, or controls.

There are also relatively simple trust metrics “similar to the simplest one, with the additional constraint that path lengths are bounded by some parameter k . Thus, all nodes within a distance of k edges from the seed are accepted. A variation of this trust metric is used in X.509 systems” [89].

Therefore, Chen et al. stated that propagation and aggregation of trust are crucial trust metrics [25]. In addition, trust transitivity is discussed in terms of its two primary operations: concatenation and aggregation.

Together with trust metrics, trust models are a fundamental part of trust management. Moyano et al. [112] made a classification of trust models. This work is essential because even if it should not be useful to compare models of different classes, it is crucial to extract some similar features from them. Following this premise, it is possible to create a general framework containing these features.

The following classification provided by Moyano divides the trust models into two main categories:

- **Decision Models:** “They aim to make more flexible access control decisions, simplifying the two-step authentication and authorization process into a one-step trust decision. Policy models and negotiation models fall into this category. They build on the notions of policies and credentials, restricting the access to resources by means of policies that specify which credentials are required to access them” [112].
- **Evaluation Models:** “Their intent is to evaluate the reliability (or other similar attributes) of an entity by measuring certain factors that have an influence on trust in the case of behaviour models, or by disseminating trust information along trust chains, as it is the case in propagation models. An important sub-type of the former are reputation models, in which entities use other entities’ opinions about a given entity to evaluate their trust on the latter” [112].

Policy models (as Policy Maker [20]) are a sub-type of decision models; they have rules that are used to give or not to give access to a resource. These rules are named policies, and they are written with a policy language [112].

Other decision model types are the negotiation models (as Trust Builder [159]). As Moyano [112] stated, “Trust negotiation models add a protocol, called negotiation strategy, during which two entities perform a step-by-step negotiation-driven exchange of credentials and policies until they decide whether to trust each other or not. This strategy allows for protecting the privacy of the entities as policies and credentials are only revealed when required”.

One type of evaluation model is the behaviour model. These models are often built in a systematic way and through three phases [112]:

1. Assign a trust value to the entities belonging to the system.
2. Monitoring the entities and their attributes.
3. Assign values to the monitored attributes merging them to compute a final result called *trust or reputation score*.

The final score is a value showing how much the trustor trusts the trustee, and it can be unidimensional or multidimensional [78]. In the second case, the values

can come from different aspects of trust. Trust metrics are used to compute these values, and they compute variables like security or utility to give a final total score to relations [112].

Reputation models help compute an initial trust value if the trustor has never had previous interaction with the trustee. These models can be centralised or distributed. In the first case, there is an entity (a trusted third party) that has to collect the reputation of the other entities and share these values among all the entities. In the second case, every entity collects information about other entities and shares this information with the other entities. In both cases “the model might consider how certain or reliable this information is (i.e., the credibility of witnesses), and might also consider the concept of time (i.e., how fresh the trust information is)” [112].

Propagation Models assume that some trust relationships are available in advance. Then this information must be shared and disseminate to other entities. These entities have no knowledge about other entities and if they are trusted or not. In this model, the assumption about the transitivity of trust is fundamental. Some models like Advogato [88] or the E-Bay reputation system [130] are based on transitivity trust.

Regarding simpler trust models, one of “the most basic is the model of password-protected accounts. This mechanism is almost universally deployed but suffers from severe limitations, including the need for servers to manage the accounts, the need for users to keep track of a large number of passwords, and the relative lack of security provided by this model. Thus, there has been a sustained interest in more sophisticated models” [89].

Another model is based on the Public Key Infrastructure (PKI), where a trusted third party is present. In this case, it is known as *Certification Authority* (CA), and it is able to release a digital certificate. The PKI is composed of various CAs and, as Levien [89] stated, this architecture suffers from two main issues:

1. the lack of useful meaning in the PKI’s underlying namespace.
2. the question of which CA to trust.

In addition, Levien stated that “implementations of CA’s have proved themselves

not worthy of absolute trust. Further, as the number of CA deployed scales up, the risk of any one of them being compromised scales accordingly. In part because of these two problems, PKI's have met with limited success at best" [89].

In this thesis, we will consider trust management, trust metrics, and trust models considering the state of the art and moving forward proposing our considerations.

2.2 System Development Life Cycle (SDLC)

System Development Life Cycle (SDLC) is a conceptual paradigm that involves rules and procedures for developing or modifying systems during their life cycles. It is a systematic approach that explicitly divides the work into phases required. In this section, we will focus on how two phases of the SDLC have been considered in state of the art: the requirements and model phases.

2.2.1 Requirements Engineering

Requirements engineering is one of the first phases of the System Development Life Cycle [67], and Software Development Life Cycle [106] (in this thesis, we will refer to both of them as SDLC).

In the State of the Art, many authors have considered this topic. Nuseibh et al. defined an important statement about requirements engineering. They stated that "the primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Broadly speaking, software systems requirements engineering (RE) is the process of discovering that purpose by identifying stakeholders and their needs and documenting these in a form that is amenable to analysis, communication, and subsequent implementation" [118].

Zave gave one of the most straightforward definitions of requirements engineering. She stated that "requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software

families” [164]. This definition is essential for different reasons. Firstly, it focuses on the fact that the “real-world goals” need to be at the center during the development of a system. They represent the *why* and the *what*, critical questions for the motivation of building a system. Secondly, the words “precise specifications” define one of the key points during requirements analysis: the future validation of what the stakeholders originally wanted, and the guide for developers in order to build the right system. Lastly, this definition specifies “evolution over time and across software families”, underlining how the real-world changes frequently, so it is needed to define a requirement taking this change into consideration. Anyhow, this definition focuses only on software engineering, but these aspects can also be taken into consideration for a general system.

Hull, Jackson, and Dick [37] in their book stated that requirements are the basis for every project, defining what the stakeholders (i.e., users, customers, suppliers) and developers in a potential new system need from it and what the system must do to satisfy that need. In order to be understood by all these actors, requirements are generally expressed in natural language. However, this representation can be helpful, but it could have many different interpretations. Here, we have one major challenge that is to capture the need entirely and unambiguously. Moreover, we need to avoid the utilization of specialist jargon. Therefore, a guide on how to write and elicit the proper requirement is needed. Thus, once communicated and agreed upon, requirements drive the project activity. Nevertheless, the stakeholders’ needs may be many and varied, and they can lead to conflicts among them. Besides, these needs may not be clearly defined at the start, maybe constrained by factors outside their control, or maybe influenced by other goals that change over time. Thus, without a stable requirements base, a development project can fail.

The IEEE 830-1993 requirements specification [33] can help in this direction because they defined how a requirement must be written in order to avoid different interpretations. This specification states that a requirement must be: correct, unambiguous, complete, consistent, ranked for importance or stability, verifiable, modifiable, and traceable.

A requirement is *correct* “if, and only if, every requirement stated therein is

one that the software shall meet” [33]. It is hard to prove that a requirement is correct, and usually, it is done considering the ‘why’ of a system and if the elicited requirement satisfies it. Moreover, the correctness of a requirement can be verified by the developers and the final users in order to check if it reflects the needs of the system.

Unambiguous property is satisfied “if, and only if, every requirement stated therein has only one interpretation” [33]. This is a crucial property that fixes the problems related to the utilization of natural language to define a requirement. Following this property, the developer must check if the elicited requirement has only one interpretation, if not, the requirement must be rewritten.

A requirement is *complete* if it covers a specific aspect without leaving any part unspecified. The completeness of a requirement or a set of requirements is a crucial point to cover all the specified needs. If a set of requirements is not complete, the developers must rewrite the requirements or insert new ones in order to satisfy this property.

Consistent is strictly connected to the *correct* one. In addition, it is a crucial property also if different requirements must be compared. In fact, through this property, we can find out if there is conflict among different requirements. For example, this can happen if two requirements defined two different situations and they cannot be implemented together, but only one of them can be satisfied. The decision-making process will be helpful in these situations in order to decide which requirement to preserve and which one to delete or modify.

The property *ranked for importance or stability* is helpful in order to solve conflicts among requirements of the same set.

A requirement is *verifiable* “if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement” [33].

A requirement can be *modifiable* only if changes can be made “easily, completely, and consistently” [33]. So, it is possible to modify a requirement, but the new requirement must meet the other properties.

A requirement must be *traceable*. This means that “the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation” [33]. In the IEEE 830-1993 specification, two types of traceability are proposed: backward and forward traceability. Backward traceability depends “upon each requirement explicitly referencing its source in earlier documents” [33]. On the other hand, forward traceability is satisfied if each requirement is uniquely identified.

Following these premises, we can state that requirements engineering is fundamental in order to perform a reasonable, effective, and efficient development of any system or software.

2. 2. 1. 1 *Requirements and Trust*

In this thesis work, we will also analyse all those previous works that have considered trust or related properties during the requirements engineering process. However, it is important to underline that for many years, trust and related domains such as security, identity, usability, and privacy were not appropriately considered in the first phases of SDLC [120]. Furthermore, Yu et al. [162] stated that trust could be considered a non-functional requirement depending on other aspects in all the SDLC and Giorgini et al. [62] reasoned on how it can be possible to include trust in requirements language.

Indeed, to collect and elicit proper requirements during the earliest phases of the SDLC is a fundamental task that has positive outcomes and minimizes raising problems in the following phases of the SDLC. Usually, as we specified earlier, requirements elicitation is performed by developers following stakeholders’ needs. The latter are persons or companies having an interest in the system or software which is under development.

During the requirements elicitation phase, it is helpful to use appropriate techniques to elicit the proper requirements. Some of them have been widely used with the introduction of Goal-Oriented methodologies [21, 100, 111, 163].

The first one was I* and it has been developed by Yu [163]. This language

introduces the notions of actor, goal, and dependencies. SI* [100] is an extension of I* focusing on security and including notions related to it and secondarily to trust. Another methodology is TROPOS [21]. It is based on the I* framework methodology, and it was developed to support all the design activities during the SDLC. Finally, Mouratidis and Giorgini extended the Tropos methodology creating Secure Tropos [111]. In this work, it is explicit which actor owns a service and is able to provide it. Then, considering other trust-related domains, Rios et al. [131] have highlighted how it might be necessary considering also privacy characteristics in the requirements elicitation phase. This process enhances trust, especially during trust negotiation processes. Finally, Mavropoulos et al. [102] proposed a methodology to elicit security requirements for the Internet of Things. They stated that “using JSON format the process of requirements elicitation can be automated, thus making the analysis of large IoT networks more efficient”.

Although, none of these works have considered trust holistically with other trust-related domains. On the contrary, to guarantee trust, it is essential to consider other domains related to it, as Hoffman [70] and Pavlidis [124] stated. Following this premise, one of the works previously cited proposed by Rios et al. [132] have considered privacy during trust negotiation. Besides, Gago et al. [46] have considered trust, privacy, and identity as requirements to be taken into consideration during the development of a system, particularly an Internet of Things system.

To summarize, there are plenty of works related to the requirements elicitation process in state of the art, but none of them considers trust properly alongside other domains. Moreover, these works focus more only in the earliest phases of the SDLC without an holistic perception during its whole period. Our thesis work aims to fill this gap taking these works into consideration and going further, considering trust strongly related to other domains such as security, usability, and identity. Moreover, we will propose an innovative requirement elicitation method for the Internet of Things.

2.2.2 Modeling Languages

During the SDLC, the modeling phase is essential in order to define the model specifications that a system will need and how to implement important features elicited in the previous phases.

Two widely implemented modeling languages are known as Unified Modeling Language (UML) [137], and System Modeling Language (SysML) [56]. These languages are useful to explore the different functionalities of a software/system. In addition, the UML/SysML diagrams help developers to define the proper software/system. However, these modeling languages have not been designed with features related to security, privacy, or trust. Nevertheless, in the state of the art, several authors tried to solve this issue by expanding UML, implementing extra features. However, a few authors have proposed a way to consider trust in the first phases of the SDLC.

About security, Jürjens [80] considered security policy validation and encryption extending UML in UMLsec. Furthermore, Basin et al. [15] and Lodderstedt et al. [93] extended UML in secureUML in order to implement access control rules. A limitation related to this work is that they have not considered scenarios where the access control rules are violated. Moreover, they have not deeply considered the requirements phase focusing only on the design phase.

Concerning risk analysis, the following UML extensions implement risk and threats adding some features in order to consider them during the modeling phase. Thus, Vraalsen et al. [155] used CORAS [38] to implement threat and risk modeling. Furthermore, Hussein et al. [74] extended UML in UMLintr to include intrusion detection into the models.

Related to trust, an interesting paper that considers trust in UML has been proposed by Uddin and Zulkernine [152]. UMLTrust focuses on the system design and specification phases of the SDLC. In particular, they enhanced the class, state machine, and use case diagrams with trust stereotypes. However, they did not consider other properties related to trust (i.e., security or privacy). Neither they consider important phases of the SDLC such as needs, verification, and validation. Finally, there is no backtracking between the phases, and this is a considerable limitation in

order to implement modifications.

Previously, only Gorski et al. defined stereotypes to implement trust in UML use case diagrams [63]. They considered only pieces of evidence as claims in order to modify the trust level of the trustor. Even though this was one of the first works including trust in UML, a considerable limitation of this proposal is that they did not implement other trust characteristics.

About SysML, only several authors have enriched it by implementing security properties, but none have expanded these diagrams with trust stereotypes. Maskani et al. [99] have extended SysML considering especially the requirements diagram. The authors included security stereotypes in order to consider them also during the requirements elicitation process. However, their work does not consider security-related properties such as privacy or trust. Moreover, it is related only to the requirements phase. Some years earlier, other authors worked on implementing security stereotypes in SysML. Thus, Apvrille et al. [9] developed a framework called SysML-Sec, and it was designed for embedded systems. They aimed to extend SysML to cover requirements, design, and validation phases. In order to elicit security requirements, they took into consideration threats analysis and risk assessment. Nevertheless, they did not implement traceability in order to connect needs and requirements. Another limitation is that they did not consider other properties related to security.

To summarize, no one of the aforementioned modeling language extensions was intended to be used specifically for IoT. Thus, considering that in the state of the art there are not modeling languages proposed for IoT and that there is still little effort to consider trust during the modeling phase of the SDLC, our thesis work aims to fill this gap by merging the trust models domain with modeling languages such as UML and SysML. We propose a model-driven approach useful to consider trust and its related properties in the model phase of the SDLC.

In order to perform this task, we consider features identified by Moyano et al. [112] useful to enrich our work with trust. These features can be implemented in several models.

Firstly, there are the **common features** belonging to every trust model. The actors involved in a trust interaction can be:

- A *witness* that can provide information to other actors.
- A *trustor* or a *trustee* that are dependent on the context (i.e., a requester of a service and a service provider).
- A *trusted third party* (TTP) that can provide credentials (i.e., certificates) to the other actors.

In addition, the purpose (i.e., why to establish trust?) is important. According to Grandison and Sloman, Jøsang, and Moyano [65, 78, 112], it is possible to identify four primary purposes: access trust, provision trust, identity trust, and infrastructure trust. Besides, trust is always context-dependent [1, 112].

Concerning the **decision models**, the critical feature is the credentials (i.e., provided by a TTP), policies, and evidence. In addition, it is a particularity of these models to provide a step-by-step authentication that preserves the privacy of the entities. In fact, policies and credentials are revealed only when they are required, avoiding the disclosure of extra information when it is not needed.

Finally, regarding the **evaluation models**, Moyano identified that a trust level is always present, and it could be uni-dimensional or multi-dimensional. According to Jøsang [78], it might have different degrees of objectivity or scope. In order to compute these values, trust metrics are necessary, and they need attributes to be computed through engines (i.e., simple summation, fuzzy, bayesian). One crucial attribute could be reputation. In the propagation models (a subset of the evaluation models), a trust path is important and how to propagate trust through the nodes of the path is a crucial point.

2.3 Internet of Things

Internet of things (IoT) is composed of two words: Internet and things. With these two words, we can understand the scope of this technology, which is about the connection of things among them through the Internet. Undoubtedly, the Internet brings many possibilities (i.e., providing communication anywhere in the world), but

many problems can arise (e.g., threats or cyber-attacks). The word thing is generic; these things can be inanimate or humans (i.e., connected by smart-phones, laptops, or tablets). In fact, through the IoT, we can connect different types of things and how to connect them in a protected and trusted way is one of the main challenges in this area. In this thesis, we will use the term things, devices, or entities for the same purpose.

IoT analytics [8] forecasted that the number of connected IoT devices will be around 21.5 Billion in 2025, as it is shown in Figure 2.1.

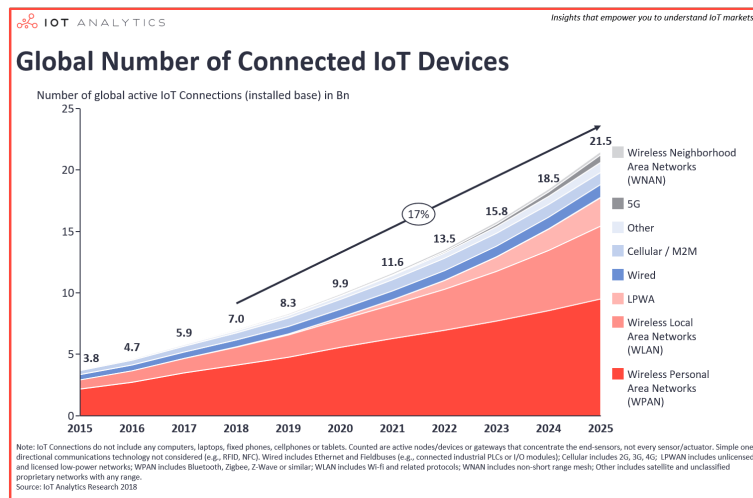


Figure 2.1: Previsions of total number of connected device up to 2025 [8]

Gazis stated that “IoT is understood as the revolutionary transition into an era where physical assets and virtual assets will be treated uniformly and, for all intents and purposes, be mostly indistinguishable to the processes involving them. The sheer scale of IoT suggests that harmonized global standards will be paramount in realizing a seamless treatment across the physical facet and the virtual facet of things” [58].

Before the IoT, one of the first technologies used to allow things to communicate among them was called Machine to Machine (M2M). As Watson stated, M2M “is a term used to describe the technologies that enable computers, embedded processors, smart sensors, actuators, and mobile devices to communicate with one another, take measurements and make decisions - often without human intervention” [158]. However, M2M is where the *Machines* use a network to communicate with remote

application infrastructure only for purposes of monitoring or control the machine itself or its environment. IoT is an upgrade that permits the objects to interact on their own and with the environment. Thus, IoT is a concept and a paradigm that considers pervasive presence in the environment of various things that, through wireless/wired connections and unique addressing schemes, can interact with each other cooperating to create new applications/services and reach common goals. In this context, the research and development challenges to create a smart world are numerous and hard to implement. A smart world where the real, digital, and virtual are converging to create smart environments that provide energy, transport, and services among the smart entities. Moreover, according to the heterogeneity of the IoT, we can state that it is composed of different entities developed by different vendors, each of them with a different purpose and a different life-cycle. We want to focus on the word different to make clear that it is an entirely heterogeneous environment in every aspect.

The goal of the Internet of Things is to enable smart entities in order to be connected anytime, anyplace, with anything and anyone ideally using any path network and any service [133]. Things can make themselves recognizable, and they become “intelligent” by making or enabling context-related decisions. They can provide information about themselves or access information provided by other things. Moreover, together with other smart entities, they can be components of complex services. Anyhow, it is expected that these entities have to interact with each other often under unclear conditions. Mechanisms useful to address this need for information can consider trust in order to overcome uncertainty.

With the IoT enabling smart homes and smart cities, it is possible to connect everyday entities and control them remotely. To ease this deployment, the manufacturers of the IoT devices allow the owners of such devices to control them even when they are away from their home network. This functionality enables connected devices to be synchronized and take instructions from other smart entities devices ⁴ and services ⁵.

⁴<http://www2.meethue.com/en-gb/>

⁵<https://developer.amazon.com/alexa>

However, several issues can appear. One is related to the fact that manufacturers of smart things usually include different communication technologies, such as Zigbee or Zwave [125]. These technologies are embedded with either proprietary or one of the many standard protocols [59]. Moreover, they usually cannot communicate directly with each other [61] but with a central station that allows communications among them through a “legitimate man in the middle”. Another issue is related to the use of different versions of the same technology. For example, in the case of Bluetooth Low Energy (BLE), backward compatibility with previous versions of the same protocol is not always guaranteed [22]. One adopted solution for this problem has traditionally been for the manufacturers to create their own IoT smart hub corresponding to the supported devices. Considering these aspects, the challenges in building a set of heterogeneous smart entities allowed to cooperate with each other grow harder.

Earlier, we mentioned a smart hub. In fact, in the case such a device is present, we can describe the correspondent IoT architecture as centralised; otherwise, we can define it as distributed.

In a centralised approach, the smart home hub is a gateway usually managing a group of mostly passive devices. The primary control belongs to the hub itself. The major threat related to this kind of architecture is that when the smart hub is compromised or stops working, the whole architecture will fail. As Singh [146] stated, many attacks can be performed against the smart home hub. A message modification attack or a replay attack are possible examples of attacks that can significantly impact a smart home environment. For example, using a replayed signal, the attacker can indefinitely send a command as continuously open and close a window. On the other hand, with a message modification attack, the attacker can modify a parameter set by the user or by the system. Thus, in the event of a fire, for example, the threshold level related to the smoke detection can be modified, and this can result in the alarm being switched on too late or remaining switched off. This is a safety risk, and it can lead to severe consequences for everybody living in a smart home or neighbours.

In a distributed approach, all the entities have determined rules [134]. Usually, when a condition is satisfied, the connected devices will execute an action locally and

independently without a smart hub command. Substantially, peer-to-peer communication is expected in this type of network [39]. According to Roman et al. [134], the major risk in a distributed architecture lies in the fact that the entities are not well protected as the central unit is in a centralised architecture. In fact, if an attacker knows how to target a particular node, it will be compromised, for example, leaking private information. Anyhow, there are possible different types of this architecture, like the one proposed by Parra [123] where some nodes are in the middle of the communication. It is a sort of mix between a centralised and a distributed architecture. In this case, the problem is raised in case one of these middle nodes fail. If this happens, the architecture will also be partially or entirely damaged.

Anyhow, these architectures have been considered in order to create frameworks used in the IoT [46, 149] and some of these structures can be applied to different IoT fields, such as smart-cities, smart-grids, or smart-homes [123].

About smart-grids, some of these architectures are well known in the industrial control systems [150] where the networks are divided into two or more parts, using firewalls to protect the more vulnerable networks from direct attacks exploited through the Internet. This approach enhances security, trust, and privacy. They are important characteristics that must be guaranteed in order to protect users and things from cyber-attacks and their consequences [127].

However, independently from the architecture, these objects have to communicate among them in order to interact. As we have shown earlier, communication can be difficult among different devices from different vendors because of many issues. Trust can help to address this need and to make the entities trust each other during their communication.

2.3.1 Trust in the IoT

In the state of the art, several authors have proposed how to consider trust in the IoT. However, due to the uncertainty, interoperability, and heterogeneity of the IoT environment, achieving trust is still a challenge. Leister et al. stated that “the Internet of Things will connect many different devices. In order to realise this, users

must be willing to trust the devices and communication that happens automatically” [86]. Moreover, because independent research communities have tackled these aspects separately, a holistic approach is desirable [46].

However, the field of trust related to IoT is still in its infancy as stated by Azzedin et al. [12]. Thus, with their work, they want to “raise the awareness and the need for behavior trust modeling” in information fusion and IoT areas. In fact, trust in the IoT is very important because, in order to communicate, the smart devices have to trust each others. Elkhodr et al. [42] focused on the fact that in the IoT is very important to know the origin of the source of data and understand if it is possible to trust them or not. Moreover, they stated that “this requires accurate, secure, and correct data collection processes and the provisioning of data provenance throughout the life-cycle of an IoT device and the data it produces”. In our thesis, we focus precisely on this point. Moreover, in the majority of the cases, the interacting smart entities have never communicated among them in the past. So, they do not know each other directly. For this reason, it is important to create a trust relationship to allow smart devices to communicate among them in a trusted way [161]. Furthermore, to be trusted is a prerequisite for being socially accepted for a software or an IoT entity [29]. In fact, if there is no trust, it will be challenging to sell a product and increase its market [101].

According to Wang et al. [157], “indicating trust or distrust of a node is a critical issue in the trust management of IoT”. However, because of the huge numbers of nodes in the IoT, “there are some challenges for the management with anonymous nodes”. A solution proposed by the authors considers a dynamic trust model to predict trust and distrust, computing their values.

Yan et al. stated that “trust management plays a vital role in IoT for reliable data fusion and mining, qualified services with context-awareness, and enhanced user privacy and information security. It helps people overcome perceptions of uncertainty and risk and engages in user acceptance and consumption of IoT services and applications” [161].

Furthermore, Gago et al. stated that “the Internet of Things (IoT) is a paradigm based on the interconnection of everyday objects. It is expected that the things

involved in the IoT paradigm will have to interact with each other, often in uncertain conditions. Therefore, it is of paramount importance for the success of IoT that there are mechanisms in place that help overcome the lack of certainty. Trust can help achieve this goal” [46].

As we wrote in the previous sections, trust is general, and to trust someone or something has different meanings depending on the context. Moreover, in an environment such as the IoT, trust can be related to different aspects. In such interaction, the context is critical, and it can be different for each of the things involved. Therefore, there is the possibility that in the same scenario, there can be different contexts with different trust relationships. In fact, IoT is dynamic, and this aspect affects the trust relationships because if a thing is trusted in a particular context, this should not be true for another context. In this case, if the context changes, the trust relationship can change too [46].

Reputation can be significant in an IoT environment, especially if two or more entities did not have any past interaction among them. Reputation can be used as a parameter to define the initial trust level. About this, Hussain et al. [73] stated that trust and reputation are always crucial in any interaction among IoT entities even this relationship is among Humans-to-Humans (H2H), Machines-to-Machines (M2M) or Human-Machine-Interactions (HMI). They proposed “a context-aware trust evaluation model to evaluate the trustworthiness of a user in a Fog based IoT (FIoT)” and they considered a “context-aware multi-source trust and reputation based evaluation system that helps evaluate the trustworthiness of a user effectively”.

Recently, Ursino et al. [153] stated that “if a thing can have a profile and a behavior like a human, it is not out of place to extend the concept of trust and reputation to things and to define ad hoc approaches for their computation”. They studied trust and reputation of a “thing” in multiple IoTs scenarios proposing a context-aware approach to evaluate them. However, they have modeled differently the way things and persons are considered. In fact, they have observed that “the number and the variety of available things is leading researchers to model the existing reality as a set of IoTs interacting with each other, instead of a unique IoT”. This is an interesting point to be taken into consideration during the development of a

smart IoT entity.

As for trust management, reputation management can be centralised or distributed [134]. In a centralised architecture, there is a node that contains all the reputation values of the nodes. In the distributed one, each node must store the reputation values of all the other nodes of the system separately. When an IoT device wants to establish a connection with another device, it needs a reputation value to instantiate its starting trust level. In a centralised architecture (i.e., with a central IoT hub), to obtain the other IoT devices' reputation value, the requesting IoT entity asks the central hub for the reputation value. Once the value is obtained, the requesting IoT device will decide if to proceed with the exchange of information. On the other hand, in a distributed architecture, every IoT device possesses some information about the other entities, and if a new connection is about to be created, the IoT entities exchange their information among them. In both architectures, trust is crucial in order to decide which node to trust and interact with or not.

In addition, Fortino et al. [54] proposed a survey on trust and reputation in the Internet of Things, analysing the state of the art and open research challenges. Their main contribution is to provide a comparative study of the existing architectures related to trust in the IoT. Summarizing, we can state that in a centralised approach, the amount of data that must be computed by the single IoT devices are less, but this creates a bottleneck in the communications. On the other hand, in the distributed approach, IoT devices need more computational power. Anyhow, there are researchers investigating how it is possible to reduce the amount of data to be computed in the IoT. Li et al. [90] focused on the fact that IoT allows the connection among many heterogeneous devices, and trust is fundamental in order to assess the quality of the different available services. Moreover, they consider context crucial because it is possible to trust one service for a particular purpose and not for another. They proposed a “new context-aware trust model for lightweight IoT devices” without storing information about the nodes' past behaviours because of their limited computational power. In fact, their model needs only a limited amount of stored information, and it can resist several attacks such as badmouthing and on-off. Another possibility is offered by Fortino et al. [53], where they proposed to

“use the capabilities of nearby devices having appropriate resources, given that they make their resources available for free or with a determined cost”. They proposed a solution “where each IoT device is associated with an agent that helps its device in choosing reliable partners for its tasks”. They use reputation as a “countermeasure against malicious IoT devices”.

Fortino et al. [54] have also analysed the up-to-date IoT architectures explaining how to integrate them with nodes belonging both to the fog and edge computing paradigms. Edge computing is intensely used in IoT. Sadique et al. [141] investigated the integration of distributed trust management in the IoT through edge computing technology, considering scalability and heterogeneity of the IoT devices. Moreover, Junejo et al. [79] proposed a “trust management system for fog-enabled cyber-physical systems”. They consider the trust values computed by their model in order to assess a credibility factor for each node of the system. This factor helps to avoid and isolate malicious fog nodes and preserve the others.

In addition, about Fog computing and trust, Alemneh et al. [5], proposed a two-way trust management system for fog computing. The authors aim that by guaranteeing trust, it is also possible to provide security and privacy. More specifically, they proposed a “logic-based trust management system that enables a service requester to verify whether a service provider can give reliable and secure services and lets the service provider check the trustworthiness of the service requester”.

Moreover, as we discussed earlier, security and privacy are strongly related to trust, and they must be taken into consideration. In fact, as Sicari stated, “the satisfaction of security and privacy requirements plays a fundamental role. Such requirements include data confidentiality and authentication, access control within the IoT network, privacy and trust among users and things, and the enforcement of security and privacy policies. Traditional security countermeasures cannot be directly applied to IoT technologies due to the different standards and communication stacks involved. Moreover, the high number of interconnected devices raises scalability issues; therefore, flexible infrastructure is needed able to deal with security threats in such a dynamic environment [143]”.

In order to solve IoT privacy issues, a necessary approach is Privacy-by-Design, as

stated in the US Federal Trade Commission report on consumer privacy [32]. More precisely, in the IoT, the challenges with respect to privacy are principally related to the users and their private data. How they are stored and in which architecture is fundamental. Furthermore, privacy issues are very different from each other [127] depending on the used application.

Concerning the security aspect, if a security architecture is not provided to the IoT entities, they can suffer from malfunctions or attacks. To mitigate these issues, IoT needs a holistic approach securing all the entities, from the physical to the application layer [133]. These security mechanisms are fundamental in order to preserve privacy and build a trusted environment.

Summarizing, in the state of the art, trust and IoT have been investigated by several authors. Some of them have proposed different frameworks to include trust in a system or software. We have proposed a collection of these works according to the topics discussed by the same authors. We summarize the search criteria that we have considered in Table 2.5 and the works that we discussed in this section.

Table 2.5: Search criteria about trust and IoT and related works analysed

Heterogeneity, Uncertainty and Dinamicity	[46, 86, 90, 143]
Communication	[12, 86, 90, 161]
SDLC, Context and Domains	[42, 46, 143]
Trust vs Distrust	[29, 101, 157]
Trust Management	[5, 79, 161]
Reputation	[53, 73, 153]
Architecture and Scalability	[5, 54, 134, 141, 143]
Holistic	[46]

In the next part, we will present frameworks developed to include trust in the IoT and some general frameworks (not specific for the IoT) that can be used (even if ineffectively) in the IoT. However, both of these categories have some issues. With this

thesis, we will present our framework in order to fill the gaps discovered, providing a holistic framework for trust in the IoT.

2.3.2 Frameworks for Trust and IoT

Starting from Pal et al. [121] work, they have proposed a trust management framework focusing on access control mechanisms improving decision-making processes under uncertainty. They provided attribute-based identity management. In their proposed trust model, the access control decision has been made considering three different types of trust: direct, recommended and derived.

Analysing the fact that mobile technologies are an enabler for the IoT, Bica et al. [19] proposed a “security framework with a multi-layer architecture that addresses the trust evaluation of sensing devices based on reputation scores calculated using a naive Bayes algorithm”.

Considering specifically the IoT, DeMeo et al. [35] proposed a reputation framework for the IoT embedded with a Reputation Agent (RA) acting inside an entity. This RA is separated from its belonging entity in order to estimate an “honest” reputation value. This is an interesting approach, but it is vulnerable to self-promotion attacks in the case that the device is manipulated. Indeed, a reputation score given by another trusted entity is more reliable. Furthermore, they state that it would be infeasible to effectively apply an approach based only on authentication deal with trust issues in a wide environment such as the IoT.

In addition to the framework proposed in Section 2.1.5, Ruan et al. [136] proposed a trust management framework explicitly intended for the IoT. This framework can be applied to various applications, and it is based on the measurement theory [66]. The authors considered only two metrics: trustworthiness and confidence. An interesting aspect is that they have modeled interactions between the IoT entities dividing them into four types of interactions: human/human, things/things, and human/things (in both directions) interactions. Moreover, they have considered reputation to calculate a trust level showing how trust can help recognize which nodes are malicious or trusted. However, they have analysed only two types of attackers,

so their framework is useful only against certain types of threats.

Sharma et al. [142], have presented a generic framework to manage trust in the IoT considering both qualitative and quantitative parameters. They have proposed a trust management solution considering all the requirements useful to perform trust management. This framework is interesting, but its main weakness is that there is only one feedback from the very last phase coming back to the first phase. This is a considerable limitation in the case some issue is encountered in the middle of the framework. In addition, another disadvantage is that the context has never been taken into consideration.

Gago et al. [46] moved forward and introduced a framework to help designers and developers consider trust in IoT. They state that privacy and identity requirements must be considered during trust and reputation management to enhance trust. However, there is no feedback between phases in this framework, and there is no connection among privacy, trust, and identity requirements. We have connected requirements to ensure traceability between them. Finally, they model only the first phases of the SDLC.

Then, Bahutair et al. [14] considered an adaptive trust model for IoT services. Users' utilization assesses the trustworthiness of these systems. In order to determine if a system can be trusted or not, an algorithm process several trust factors through four different stages. The first stage predicts trust factors. The second stage computes these parameters in order to predict the trustworthiness of the system. Then, in the third stage, a "usage-to-factor model" is built to detect how important is each factor for different scenarios. Finally, the last stage is composed of two models. Their aim is to compute a trust value according to the scenario chosen in the previous phase.

In IoT, we can also consider the cloud of things. About this, Abualese et al. [2] stated that cloud of things is a paradigm intensely used by e-government, and in this aspect, trust is of critical importance and also a challenge. Thus, they proposed a framework in order to enhance trust among IoT devices connected to the cloud. Their framework is composed of four layers. One of them is dedicated to trust in order to authenticate the IoT devices. They used several authentication methods "to

differentiate the access control for each device”.

In Table 2.6, we present the aforementioned frameworks highlighting the features that we believe are fundamental for a trust framework for the IoT. We can see that in the majority of the works there are only a few of these properties that are considered. However, Fernandez-Gago et al. started to consider all these properties together.

Our proposed framework [50] takes the previous frameworks into consideration and aims to move forward. We propose a framework in order to guarantee trust during the development of an IoT entity considering the whole SDLC. Moreover, this framework guarantees careful planning from the developer’s perspective. In addition, starting from the fact that trust is strongly related to other properties such as privacy and security, we consider the possibilities to connect them since the requirement phase. Another critical aspect is traceability, and it is provided among the different requirements and among the different phases of the framework. With this thesis, we enhance this work and we claim that in order to guarantee trust in the IoT, a framework must also guarantee the other connected properties. In addition, it is needed for the whole SDLC because it is crucial to take trust into consideration in every phase.

Table 2.6: Frameworks Properties

Authors	Models	Trust Attr.	IoT Arch.	SDLC	Domains	Activities
Pal et al. [121]	Access Control	Direct, Recom- mendation, Derived			Identity	Decision Making
Bica et al. [19]		Reputation, Multi Layer				
De Meo et al. [35]		Reputation, Reputation Agents				
Ruan et al. [136]		Reputation, Trustwor- thiness, Confidence	H2H, D2D, H2D, D2H			
Sharma et al. [142]		Quantitative and Qual- itative parameters		Require ments		
Fernandez- Gago et al. [46]	Decision, Evalua- tion	Reputation	According to the scenario	Require ments, Model, Develop- ment	Privacy, Identity	Context
Bahutair et al. [14]	Adaptive Trust		Four layers: prediction, compu- tation, usage, models			Context
Abualese et al. [2]	Access Control		Cloud of Things (CoT)			
Ferraris et al. [50]	Adaptive Trust Model [48]	Reputation, Direct, Indirect, Transitive, Dynamic, Local, Global, History dependent [49]	Segregated Trust Ar- chitecture [48]	Need, Require- ments, Model, Devel- opment, Verifi- cation, Vali- dation, Utiliza- tion	Usability, Security, Avail- ability, Privacy, Identity, Safety	Traceability, Context, Decision Mak- ing, Risk Analysis, Threat Modeling, Documen- tation, Gates

2.4 Conclusion

In this chapter, we have presented a set of definitions of trust proposed by several authors in state of the art over the years. Moreover, we have illustrated some essential characteristics related to trust actors and relationships, also defining strong connections to other domains such as privacy or security. We have then described how reputation is related to trust and how other authors have performed trust management in the past. We have then highlighted some important aspects related to the SDLC, especially requirements engineering and model-based analysis. Finally, we have illustrated what IoT is and how trust management has been considered in IoT in the state of the art.

CHAPTER 3

The K-Model: A Model for the Development of Trust in the IoT

As we presented earlier in the state of the art, we did not find any framework that can effectively consider trust and its related domains during the SDLC of an IoT entity. For this reason, in this thesis work, we develop a framework composed of the K-Model and transversal activities to achieve this missing goal. In this chapter, we will present the K-Model that is composed of seven phases, plus the context. The phases are the following: need, requirements, model, development, verification, validation, and utilization. The context will be considered in every phase. For all these phases, we will define their main concepts considering trust and its related domains to develop the desired trusted IoT entity following the SDLC flow.

3.1 General Overview of the K-Model

The K-Model, which is shown in Figure 3.1, has been developed in order to consider trust through the whole SDLC of an IoT entity, also considering its related domains. It is based on the V-Model that has been developed by Forsberg and Mooz [52]. We have chosen this model as a basis of our work because it considers connections among phases that are not only subsequent as the waterfall model does but also related to the same “level”. By level, we mean the same vertical position of the model. However, we enhance the connections among phases through traceability, as we will explain in the following sections.

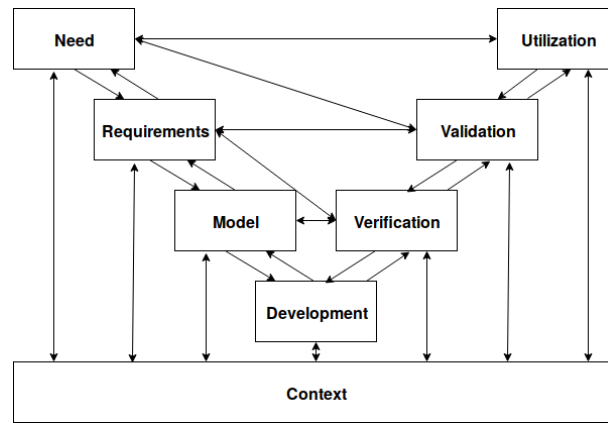


Figure 3.1: K-Model: with the *Transversal Activities* it compounds our framework

Moreover, the K-Model has been developed for the IoT and trust, but it can be adapted to other fields. We have chosen to call it K-Model because it can be viewed as a composition of a modified V-Model enhanced with the context layer. This layer is critical because it will be considered during every phase of an IoT entity’s development. In fact, due to the dynamicity and heterogeneity of an IoT environment, we always need to consider the context. The context can depend on the environment or the IoT architecture (i.e., centralised with a Smart Hub), on law regulations, or on the company’s rules developing the product. Moreover, it can be defined by the different services that an IoT entity can provide alone or together with other smart entities.

Furthermore, we have identified seven transversal activities: documentation, metrics, gates, traceability, threat analysis, risk management, and decision making. They do not belong to a particular phase, but they assist the whole process in order to enhance the model from the first phase (i.e., need) to the last phase (i.e., utilization). These activities, together with the K-Model, compound our framework.

We believe that the K-Model is appropriate for the IoT, especially for the consideration of the context that is always present. Moreover, the domains considered together are very useful for the IoT. However, it should be possible to implement the K-Model also for other systems, but we encourage the developers to consider it, especially for the IoT.

In the K-Model, each phase has connections. Firstly, we can identify the main flow starting from the first phase proceeding to the last phase. The backward connection enhances traceability to the previous phases. The forward connection guarantees specifications for the next phases. Moreover, there are other connections between the phases belonging to the K-Model's left and right sides. These direct connections are essential in order to guarantee that the specifications are fulfilled. For example, there is a direct connection from the requirements phase pointing to the verification and validation phase in order to control if the requirements have been fulfilled or not. Finally, each phase has a connection with the context layer because, as we stated before, the context is fundamental in every phase.

Starting from the need, we can state that this first phase is crucial to understand why an IoT entity is needed and what the stakeholders want from it. Then, after this phase, it is important to elicit the right requirements following the need specifications. In this phase, trust requirements are central, and they are related to other system requirements such as security and privacy requirements. The third phase is the model phase. In this phase, trust models [112], UML, and SysML [56] are essential, and we extend them in order to include trust and related properties into the development and to match the requirements and need specifications. After this phase, there is the development phase, where the developers will create the IoT trusted entity following the previous specifications. Furthermore, there are two phases: verification and validation. The verification phase will ensure that the models, the entity, and the

requirements are well implemented. The validation phase is important to test the entity in its real environment and to control if the requirements and the original need have been satisfied. Then, there is the utilization phase. This phase must take into consideration the environment and the other smart entities with which the developed entity will interact.

Finally, the context is critical to define boundaries and trust rules. These aspects need to be taken into consideration since the need phase for the whole SDLC in order to help the developers to build the proper IoT device.

In the next sections, we will present the seven phases of the K-Model. For each of them, we will define their input, their output, and their primary purpose. As we stated earlier, in each phase of the K-Model, we have to consider the context.

3.2 Need

Every product starts with a need, a problem, or an opportunity. Thus, the first phase of the K-Model is about capturing the stakeholders' needs (i.e., final users and vendors) representing the IoT entity under development. This phase is also related to the problem that the entity will solve. These needs are produced because of a problem, and the IoT entity to be developed can be considered its solution.

We can state that users and vendors are equally considered in this framework to help developers realize the desired IoT entity. Both of them will give the initial guidelines for developing the IoT entity to buy and sell it.

To satisfy these needs, we must consider several parameters: environment, context, and trust connected domains. These parameters are fundamentals for the needs documentation (see Section 4) that will be an output of this phase and an input for the following phases. Considering the environment, we can state that it is fundamental since this phase to know if the intended IoT entity will work in a centralised (i.e., with a Smart Hub) or a distributed IoT architecture in order to develop the right product. The context will be connected to the environment but also to the functionalities and final users of the developed entity. Hence, depending on the context and the

need, the entity can be developed in a very different way. For example, in an industrial system, it is necessary to separate the internal systems creating fixed boundaries between networks in order to improve the security of the whole system reducing the threats carried out by malicious external agents [119]. On the other hand, if we consider a smart home environment, hard boundaries are not defined, and the typical architectures are either centralised or distributed [123]. For this reason, a possible need can be to create boundaries even in a smart home environment. In this case, it is necessary to carefully plan both the new network system and the IoT devices that will populate it. Such boundaries would enable a precise network segmentation, and security controls must be injected (i.e., firewalls, SCADA-like systems) in order to protect home devices that are directly connected to the Internet, allowing a trusted environment. In fact, the threats can be severe in a smart-home environment where the manipulation of the Internet-connected devices can have dangerous consequences up to and including death (i.e., attack on health monitoring) [68].

Thus, one required practical consideration lies in the network segmentation between the smart entities, smart hubs, the networks, and the Internet in order to protect critical functions (such as banking) and devices (such as smart thermostats) managed by the users for critical functions.

These connections are important, but they represent a significant risk for the smart home and its inhabitants. For this reason, smart home protection is needed, and it can be achieved by dividing the internal network to protect the internal level [150] from external threats. Network sub-nets, intrusion detection systems, and firewalls, as other such security controls, are needed to protect and monitor the network allowing only particular ports to perform the actions needed.

Moreover, the basic security requirement necessary for home IoT systems is the ability of the devices when under attack to scale back operations to the bare minimum and trigger the failover systems if any are in place. This happens to effectively ensure continuity in the critical systems' operations and inform the appropriate systems of a failure. For this reason, the systemic architecture of a smart-home needs to be resilient, and this resiliency must be measure and design-dependent [76]. In short, the system needs to guarantee sufficient trust between nodes as this is vital to ensure

the privacy of various communications and actions carried out by the consumer in the network. The main research question here is how to design a resilient network that helps facilitate a measured response to incidents/alarms.

Resuming, we can state that in this first phase, the input is the needs of the stakeholders, the context, and the environment that will be considered for the new IoT entity. The purpose of this phase is to collect all these elements, considering threats and risks and deciding which need must be satisfied and which need to discard (if they create conflict among them). Finally, the documentation produced in this phase related to the needed IoT entity and its environment will be the input for the second phase of the K-Model. Moreover, it will be considered in the final phases of the K-Model in order to validate and check if the developed IoT entity reflects the originating needs.

3.3 Requirements - TrUStAPIS: A Method for IoT Requirements Elicitation

The second phase of the K-Model concerns the requirements. They are identified according to the output of the previous phase (i.e., the *need* phase). These requirements are written following the IEEE 830-1993 specification [33].

These requirements will translate the needs into a set of statements analyzed by the developers to model the IoT entity according to the chosen architecture developing its functionalities.

According to Hoffman [70] and Pavlidis [124], we consider trust requirements strongly dependent on other domains. For this reason, we have identified seven types of requirements: trust, privacy, identity, security, usability, safety, and availability. Trust is connected to each of them, and they cover all the aspects that can increase trust in an entity. Some of them can be in conflict (i.e., privacy and identity requirements), so decision-making is useful to solve these possible conflicts.

Moreover, the requirements elicitation must be performed according to its domain characteristics. For example, for trust, we need to consider its transitivity or

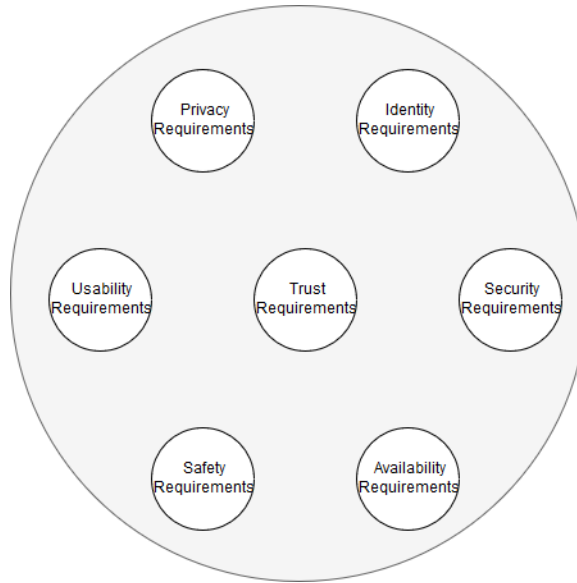


Figure 3.2: Requirements composition

asymmetry, and for privacy, we need to consider anonymity or confidentiality. It is also possible to consider a characteristic belonging to different domains (i.e., confidentiality belongs to privacy and security domains). Finally, a requirement can be specialized through one or more sub-requirements. This feature helps the developers in order to add more information to each of the sub-requirements specifying the original requirement.

The composition of the seven types of requirements is shown in Figure 3.2.

To transform the needs into requirements, we have developed a requirements elicitation method: TrUStAPIS. By implementing this method, the developers can consider trust and the other domains during the whole requirements elicitation process. The word “TrUStAPIS” is an acronym that originated from the use of the first letters of each of the seven domains taken into consideration: Trust (entirely written because it is the central one), Usability, Security, Availability, Privacy, Identity, and Safety. This method helps developers in eliciting the requirements specifically for their domain, connecting them through traceability. Moreover, this method allows the requirements elicitation considering dynamic aspects related to IoT.

TrUStAPIS considers several elements to design and then write the requirements. These elements are actors and roles, actions and measures, and goal and context.

- **Actor.** An *actor* can be a human or an IoT entity. It is the one that needs to fulfil a goal. This can be done alone or with the cooperation of another actor. Thus, an actor could have different *roles* (i.e., in a trust domain, the actors are trustor and trustee). Gago et al. [46] stated that humans could be considered IoT entities. Anyhow, during the requirements elicitation process, we consider them separately because, in order to elicit the proper requirements, we prefer to make this distinction.
- **Action.** An *action* is related to the task performed by the actor. An action can include measures or not. A *measure* helps stakeholders and developers to model a requirement that will be verified and validated in the later phases of the K-Model.
- **Goal.** A *goal* is the final purpose that the requirements represent. It can be achieved by actors through goal actions. This goal can be related to a particular capability of the IoT entity depending on the requirement's domain. Moreover, according to IoT's dynamicity, the same actor can be involved in different goals or perform different actions to fulfill them. When an action directly determines a goal, it is called *goalAction*.
- **Context.** The *context* is related to three elements. The first one is the requirements domain: trust, usability, security, availability, privacy, identity, and safety. Each of these domains has its proper characteristics. The second is related to the IoT environment. Finally, the third one is about the scope of the goal.

To summarize all these elements, we propose a conceptual model that is shown in Figure 3.3 to present the relationships among all the components of TrUStAPIS.

Regarding the *actor*, it can be a human or an IoT entity and it is the “subject” of the requirement. Each actor plays a *role* that is context-dependent. An *action* can be considered as the “verb” of the requirement and it is performed to fulfil or to request a *goal*. In the first case, the “object” of the requirement is directly the goal. Otherwise, the “object” of the requirement is another actor acting to fulfil the proposed goal.

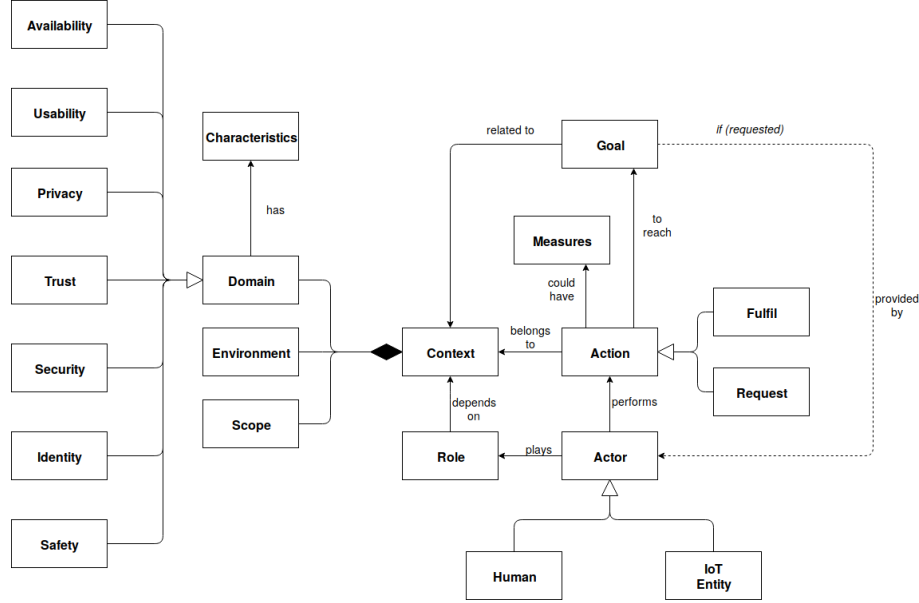


Figure 3.3: Conceptual Model for TrUStAPIS

A possibility is that an action can contain some *measures*. These measures are important in order to reach the goal and verify and validate the requirements in the following phases of the K-Model. Finally, the *context* comprises three components: a scope, an environment, and a domain. The *scope* is related to the purpose of the goal that an actor must fulfil. Then, the *environment* is related to the physical place where the action is performed. The *domain* refers to the seven types of requirements that we have identified. Finally, each domain has its *characteristics* that we will explain along with the requirements. The arrows represent the relationships among concepts. Each arrow is described by a text related to the dependence between concepts, and the direction of the arrow represents the order of the connection. Moreover, there is an optional dotted arrow representing the case that a goal is fulfilled by a secondary actor. The triangle is used to represent a specialization (i.e., an actor can be either a human or an IoT Entity). Finally, the rhombus represents a composition (i.e., the context is composed of a domain, an environment, and a scope).

Furthermore, to help developers in eliciting the requirements, we propose a JSON template. It is shown in Figure 3.4.

```

1 {
2   "IoT_requirement" : {
3     "Context" : {
4       "Domain" : [{
5         "Trust" : {
6           "Characteristic" : ["CharTrst_1", "...", "CharTrust_N"] },
7         "Usability" : {
8           "Characteristic" : ["CharUsab_1", "...", "CharUsab_N"] },
9         "Security" : {
10          "Characteristic" : ["CharSec_1", "...", "CharSec_N"] },
11        "Availability" : {
12          "Characteristic" : ["CharAvbt_1", "...", "CharAvbt_N"] },
13        "Privacy" : {
14          "Characteristic" : ["CharPriv_1", "...", "CharPriv_N"] },
15        "Identity" : {
16          "Characteristic" : ["CharIdnt_1", "...", "CharIdnt_N"] },
17        "Safety" : {
18          "Characteristic" : ["CharSft_1", "...", "CharSft_N"] } } ],
19     "Environment" : " ",
20     "Scope" : " " },
21   "Actor" : {
22     "Role" : " ",
23     "Type" : [ "Human", "Iot Entity" ] },
24   "Action" : {
25     "Type" : ["Fulfil", "Request"],
26     "Measure" : ["Meas_1", "...", "Meas_N"] },
27   "Goal" : " "
28 }
29 }

```

Figure 3.4: TrUStAPIS - JSON template to elicit requirements

We have chosen JSON because it is schematic and readable by humans and machines. Moreover, it is supported by many programming languages (i.e., Java¹). This aspect allows us to share a requirements code among stakeholders and developers through applications [6]. Furthermore, the JSON code is a useful tool to map the needs that have been identified in the previous phase of the K-Model, and it is useful in the following phases (i.e., development and verification), helping the developers to automatize the processes.

The JSON template contains all the TrUStAPIS main elements. Thus, the IoT requirement is always composed of a context, an actor, an action, and a goal:

Iot Requirement : (Context, Actor, Action, Goal)

As we explained earlier, the context is divided into three parameters: the domain, the environment, and the scope of the requirement. However, it is important to underscore that, for each requirement, there is only one domain. Each domain will have one or more characteristic that defines the requirement.

Context : (Domain, Environment, Scope)

Each actor plays a role and can be either a human or an IoT entity. All actors involved in the requirement must be set in the order of “appearance”.

¹<https://www.java.com>

Actor : (Role, Type[Human, IoT Entity])

An action is composed of types (fulfil or request) and optional measures.

Action : (Type[Fulfil, Request], Measure (optional))

Finally, the goal represents the connections beneath the requirements. In fact, some requirements may belong to distinct domains that can have connections if they have a similar or the same goal (i.e., to protect user's information can be related both to security and privacy).

Therefore, filling the JSON template can help developers in writing the proper requirements. A written requirement needs to be at least composed of:

1. one *actor*;
2. a keyword (“*shall*”);
3. a *goal* fulfilled by an *action*;

Then, a requirement could optionally have also:

- One or more secondary actors that perform an action in order to fulfil the goal.
- One or more actions needed to reach the final goalAction.
- One or more measures. The developers and the stakeholders need them in order to verify and validate the requirements.

This structure is resumed and formalized using the following *statement (1)*.

(1) *Actor shall predicate*

Thus, the Actor is also known as the main actor, and it is the subject of *statement (1)*.

The keyword shall has been chosen instead of the words *should* or *must*, because shall defines that the requirement is contractually binding and it shall be implemented and later on verified and validated.

The *predicate* can have different forms. The basic form is represented only by a *goalAction*. Otherwise, it is possible to have a more complex predicate. It could be

composed of one or more secondary actors, actions, or measures. This composition is strictly dependent on the context and defines the final written requirement.

In the following subsections, we present the seven requirements domain and their characteristics that must be considered when the developers write a requirement specification.

It is essential to underline that the same characteristic can be presented in more than one domain. Anyhow, in this case, their descriptions will be different.

3.3.1 Trust Requirements

Trust has many characteristics, which have been defined by several authors in the state of the art [124, 1, 18, 24, 27, 65, 98, 116, 160]. These characteristics must be considered while writing trust requirements. Moreover, we have highlighted the ones that are more connected to the IoT paradigm:

1. **Direct.** Trust can be direct because it is based on past experience between the trustor and the trustee. In this case, we can also say that trust is history-dependent. Moreover, this characteristic also means that trust can be subjective.
2. **Indirect.** This characteristic is fundamental whenever the trustor and the trustee have never had past interactions. In this case, trust is based on the opinion and recommendation of other entities, and it can be defined as objective. Moreover, reputation is usually considered as an indirect characteristic of trust.
3. **Transitive.** We can say that trust can be transferable from an entity to another. In fact, if an entity A trusts an entity B, but A does not know an entity C directly and B trusts C, it is possible that A can compute a starting trust value based on the fact that B trusts C. This characteristic is connected to the previous one.
4. **Dynamic.** Trust is not static over time. Even if it is not strictly time-dependent, it can change over time, increasing or decreasing its value.

5. **Local.** It depends on the couple trustor/trustee that we consider (i.e., Alice/Bob). Then, if we consider another couple (i.e., Alice/Charlie), it is possible that Alice does not trust Charlie, even if Bob trusts Charlie. It is connected to the transitive characteristic.
6. **Global.** This is a global reputation value used to compute an initial trust value. It can be related to the indirect characteristic.
7. **Specific.** The trustor (i.e., Alice) can trust the trustee (i.e., Bob) for a particular action or service, but not for other purposes. For example, we can say that Alice can trust Bob as a developer, but not as a cook.
8. **General.** If needed, it is possible to compute a general trust value by aggregating all the specific values related to point 7. To aggregate, the values will be used specific trust metrics.
9. **Asymmetric.** Trust can be asymmetric. This means that two entities tied to a relationship may trust each other in a different way. Thus, if Alice trusts Bob does not imply that Bob should trust Alice in the same way or at all.
10. **Measurable.** Trust needs to be measured. This is important for trust modeling and computation.
11. **Composite-property.** Trust can be composed of different parameters: attributes (i.e., dependability, honesty, reliability) and characteristics. Each of these parameters could have different weights and should be computed even if they belong to different metrics.

For the trust domain, the actors related to *statement (1)* are basically two and, as we mentioned earlier, they are called trustor and trustee. Moreover, it can be present with another actor, or even more, known as a trusted third party in order to fulfill or guarantee a trusted goal. Finally, it is possible to have some measures (i.e., trust thresholds), which help to decide whether to trust or distrust another actor.

3.3.2 Usability Requirements

Usability is an important property affecting the level of trust regarding an IoT entity, and it is differently considered whether the IoT entity is a person or an object. For this reason, the context is fundamental in order to choose which characteristic is more important while eliciting a usability requirement. According to Baharuddin et al. [13], we refine some usability characteristics according to their importance for the IoT:

1. **Effectiveness.** It determines the quality of the desired result or goal.
2. **Efficiency.** It affects how a result is reached, possibly saving resources and speeding up the communications among IoT entities.
3. **Simplicity.** It is important in order to reduce the complexity of the communication among the IoT entities. It is also important for users. In fact, if they have to interact with a device, a simple user interface improves its usability and trustworthiness.
4. **Understandability.** Many communication protocols such as ZigBee ² or Zwave ³ have been developed in order to let IoT entities to be able to interact among them. It is important to consider this aspect early in the SDLC in order to write the proper requirements. In fact, considering the different protocols, it is possible to choose which one to implement according to the IoT architecture (i.e., distributed or centralised) identified during the need phase. Moreover, from a user perspective, if the user interface of an IoT entity is understandable, its usability and trustworthiness will grow.
5. **Accessibility.** To guarantee accessibility via the Internet can increase an IoT entity's usability, but it can also raise issues. For example, to be able to access an IoT entity even outside its Smart Home can ensure its accessibility improving its usability, but it can also raise security threats.

²<http://www.zigbee.org/>

³<http://www.z-wave.com/>

6. **Flexibility.** Allowing an IoT entity to be connectable and reachable from different users and devices increases its usability, but it can raise security issues.
7. **Reliability.** This is an important characteristic that is strongly related to trust. In fact, the reliability in an IoT entity can improve both the usability and trustworthiness of the IoT entity.

Usability is connected to the interfaces (i.e., user interface) of the developed IoT entity. However, a device-to-human and a device-to-device interface need to be differently implemented. For this reason, it is fundamental to consider the right actor (human or IoT entity) in order to elicit the proper requirement.

3.3.3 Security Requirements

In the IoT and especially during the requirements elicitation process, it is imperative to take dynamicity into consideration, as well as heterogeneity and the context. These aspects raise the challenge of how to guarantee security in the IoT.

Nevertheless, according to [96, 114], we can take some security characteristics into consideration in order to enhance the security requirements elicitation process. The security characteristics are:

1. **Authentication.** This characteristic can improve both security and trust among the IoT entities.
2. **Authorization.** An entity must be authorized and trusted in order to be allowed to perform a particular action.
3. **Integrity.** Data integrity is a fundamental characteristic that must be guaranteed in order to preserve security and increase trust in the IoT entity.
4. **Confidentiality.** According to a particular context, communication among IoT entities must be confidential.
5. **Delegation.** In the IoT, more than non-delegation, it can be useful to delegate determined rights to other entities (human or devices). This delegation is strictly related to the context, and it could be dependent on time and users.

6. **Non-repudiation:** the performed action should be stored in order to be analyzed to understand which entities have been involved. It is crucial in the case of a security investigation after an error or an attack has occurred.

The aforementioned security characteristics must be considered according to the context of the IoT entity. Following the *statement (1)*, we can specify that the actor shall perform or request another actor a security action to fulfill a security goal. Examples of security measures are security levels (i.e., expressed in bits) or types of encryption.

3.3.4 Availability Requirements

Usually, availability is considered as a sub-property, or a characteristic of security [16, 70, 124]. Moreover, together with confidentiality and integrity, they compound the CIA triad [45]. We have decided to consider it in a self-domain because it can also be related to other domains (i.e., identity, usability) and also dependent on distinct aspects (i.e., the hardware). Availability is strongly connected to the context and the particularities of the IoT (i.e., heterogeneity, dynamicity). According to this, for example, the availability of an IoT entity functionality can be subordinate to a particular service or applicable for a determined period of time. Availability can affect positively or negatively the trustworthiness of an IoT entity if a particular service is guaranteed or not. An availability requirement can be either functional or non-functional [33].

For the availability requirements, in *statement (1)* the main actor is the IoT entity, and the *goalAction* can be related to guarantee a critical service even if a problem has occurred. A useful measure can be to have a particular threshold in order to raise the alarm or a warning and let the user knows that something wrong has occurred in order to perform corrective actions.

Considering the work proposed in [114], we identified four availability characteristics that are fundamental for trust and IoT:

1. **Resilience.** Resilience is the ability of a device to keep working and quickly recover even if something wrong has occurred (i.e., malfunctioning, attacks).

It guarantees availability because, without resilience, the device would simply stop working.

2. **Scalability.** A dynamic environment characterizes the IoT. Scalability is required to maintain the entity's functionalities even if the IoT ecosystem grows.
3. **Redundancy.** A replication of the data improves availability, but it can raise security threats. It is strictly connected to scalability.
4. **Integrity.** Availability is strictly connected to the integrity of data and devices. In fact, if the data are compromised, the entity will not work. It is strictly connected to resilience.

3.3.5 Privacy Requirements

Privacy could improve a customer's trust, but it can decrease the vendor's trust or vice versa. In fact, a vendor should decide to collect private information about the users for business purposes. On the contrary, a customer should like that its data remains private and available only for him/her without spreading personal information (even if the vendors only know them).

According to [96, 126], we consider the following privacy characteristics:

1. **Anonymity.** An entity should require to remain anonymous during particular actions.
2. **Pseudonymity.** This characteristic is important in the case that anonymity cannot be guaranteed. In this case, the privacy of the user/entity is preserved using a pseudonym.
3. **Undetectability.** This characteristic is useful for the entity to avoid its detection. It is a useful characteristic to avoid attacks.
4. **Unlinkability.** The user cannot be linked to particular data or action. Unlinkability and anonymity together guarantee unobservability.

5. **Confidentiality.** Encrypted communications among entities could guarantee confidentiality.
6. **Unobservability.** The user or the entity should prefer not to leave traces. Unlinkability and Anonymity together guarantee it.

These characteristics are compliant with the General Data Protection Regulation (GDPR) ⁴.

However, privacy requirements could be in conflict with identity requirements. In this case, decision-making processes will help developers decide which requirement to save and which one to release. Context and trust can be considered in order to perform this decision.

For the privacy domain, the actors represented in *statement (1)* shall perform actions in order to fulfill a privacy goal. So, the main actor should either guarantee privacy or avoid privacy issues. The measures related to privacy can be helpful in order to monitor privacy levels (i.e., considering differential privacy [91]).

3.3.6 Identity Requirements

Identity is significant for IoT entities. Knowing the interacting entities is fundamental in order to trust them. Moreover, identity is strongly in contraposition to privacy. In fact, the more knowledge there is about an entity, the less privacy can be guaranteed for it. For this reason, it is essential, since the early phases of the SDLC, to consider identity characteristics in order to apply them and, in the case of conflict with privacy, decide which requirement to save.

According to [95], we have identified the following identity characteristics:

1. **Authentication.** This is also a security characteristic. It is also important for identity management because if an actor is authenticated, it is also identified. Moreover, the system can perform a trust decision on him/her.

⁴<https://gdpr-info.eu/>

2. **Authorization.** It is related to security and authentication. In fact, only if an entity has been previously authenticated, then it could be authorized in order to perform an action.
3. **Attributes.** It is crucial in identity management to consider attributes because through them, it is possible to perform identification (often preserving privacy).
4. **Interoperability.** Identity management enhances interoperability among IoT entities. In fact, if an entity is identified, it could be trusted and interact with other IoT entities.
5. **Storable.** Identity information must be properly stored, accessible, and protected. It is strictly connected to the following characteristic.
6. **Manageable.** For us, manageable means that the identity information must be related to user data computation or on how they are stored and accessed. It is strictly connected to the previous characteristic.
7. **Scalability.** In determined contexts, scalability permits to have a resilient way to store and manage identity information.
8. **Accountability.** Through identity management, it is possible to recognize which actor has performed an action. For this reason, if it is possible to know which entity has performed an action, it is possible to guarantee accountability for the same entity.

As we specified earlier, identity and privacy can raise conflicts among their domain requirements. Thus, context, traceability, and specification of the identity characteristics during the requirements elicitation process can guarantee to solve these conflicts early in the SDLC.

Considering *statement (1)*, the main actor must fulfill an identification goal following one or more of the identity aspects highlighted before. Moreover, the optional secondary actor could be needed to verify the first actor's identity properties, authenticating the actor's identity attributes (i.e., code, id).

3.3.7 Safety Requirements

The Oxford dictionary ⁵ says that the definition of safety is: “the condition of being protected from or unlikely to cause danger, risk, or injury”.

Safety depends on people and machinery [87]. It is connected with the physical aspect of an IoT entity. Following the Oxford dictionary definition, we consider it in order to avoid harming the actors involved during the IoT entity utilization.

The characteristics of safety have a connection with the hardware level and physical functionalities. In IoT, safety is paramount because all the devices have embedded software, and its purpose is bound to the IoT entity.

We have identified the following safety characteristics in accordance with [64, 87, 145]:

1. **Feedback.** The users must be aware if something wrong has occurred. For this reason, the IoT entity shall provide them with information about its status. If implemented, this characteristic could avoid risks for both the user and the device.
2. **Protection.** Firstly, the users must be protected by the entity’s processes that could harm them. Secondly, the entity must be secured against physical damages.
3. **Resilience.** If a system continues to work properly, even if it is under attack or a malfunction occurs, it is called resilient. It is an important characteristic of the safety of devices and users.
4. **Integrity.** It is different from the security characteristic that is related to the data. In this case, we refer to the physical integrity of the device. In order to preserve integrity, the IoT entity shall be protected from external and internal damages that can compromise its working state.

Safety, if implemented, can improve the trustworthiness of an IoT entity. Indeed, like the other domains, it is dependent on the context (especially the environment).

⁵<https://en.oxforddictionaries.com/definition/safety>

The requirements goal shall guarantee safety for all the actors involved (i.e., to prevent the IoT entity from overheating).

3.3.8 Requirements Database Refinement

In the IEEE 830-1993 specification [33], there are two types of traceability: backward traceability (BT) and forward traceability (FT). BT is guaranteed if each requirement refers to its source. FT means that each requirement leads to a sub-requirement, a model specification in the following phase, or a particular feature. In addition to them, we define a new type of traceability: inner traceability (IT). By IT, we define the traceability among requirements belonging to different properties. For example, if we need to change a privacy requirement connected to trust and security requirements, the modification can affect all the requirements. Traceability helps to avoid domino effects. In fact, if the connection is not specified, we could have a problem in the case of changing or relaxing a requirement. In the K-Model, traceability is also a transversal activity that we will present in Section 4.

Each requirement has a unique ID to identify it and guarantee BT, FT, and IT among them. *ID* is the identifier of requirement. As we explained earlier, it must be unique and related to its type of requirement. For example, a trust requirement ID will be defined as “TRST-XX”, and a privacy requirement as “PRIV-XX” where XX is the number of the requirement. These connections will be mapped in the same requirement specification. A requirement specification sample is shown in Table 3.1.

Table 3.1: Requirement specification

ID	Requirement specification	BT	FT	IT
----	---------------------------	----	----	----

Requirement specification is the text of the requirement. It must be written following the IEEE 830-1993 standard [33]. It is written in natural language in order to be understandable by stakeholders and developers.

BT, *FT* and *IT* are the IDs of the related requirements, documents or model specification. Finally, the requirements must be saved in a requirements database.

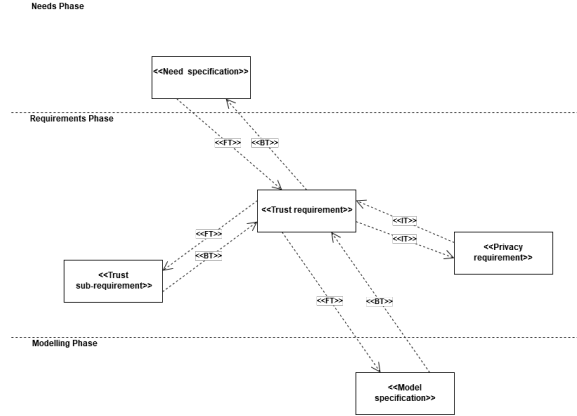


Figure 3.5: Requirements Relationships

Figure 3.5 shows the relationships between requirements of the same type, different types, and different phases of the framework. In the case of changes, it is possible to see which other requirements will be affected by this modification. At the center, there is a trust requirement. It has a parent/child relationship with a trust sub-requirement on the left, and it is connected to a need and a model. These traceability types are BT and FT. On the other hand, the trust requirement is also connected to a privacy requirement through an IT relationship. In the case the central requirement must be deleted or modified, this operation can also affect the connected elements. Thus, traceability is important to follow the impacts of this change and perform corrective actions or release the connections.

3.3.9 Step-by-Step Methodology

In this section, we present a step-by-step methodology of how TrUStAPIS should be applied. We can see the steps in Figure 3.6.

The output of each step is the input of the following step.

1. The first step starts with the output of the previous phase of the K-Model (the Need phase). This output is related to the documentation concerning the development of the IoT entity according to the stakeholders' needs.
2. During the second step, the developers fill the JSON template proposed in Figure 3.4 using the need documents as input and considering the conceptual

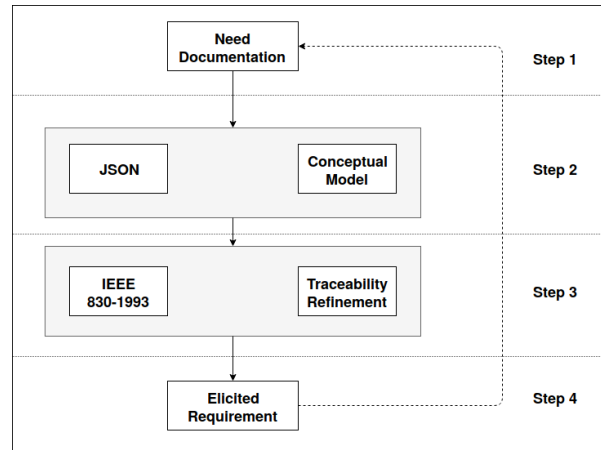


Figure 3.6: Requirements elicitation: step-by-step methodology

model that is shown in Figure 3.3. The conceptual model helps the developers in eliciting the requirements and in filling the JSON template. Both of them are used to write the requirements. Moreover, the JSON code will be the input for the later phases of the K-Model (i.e., Verification). Finally, by the JSON code, it can be possible to automatize the requirements elicitation process as proposed by [102].

3. The third step is related to the writing of the requirements. They must comply with the IEEE 830-1993 formalism [33]. According to this recommended practice, a requirement must be: correct, complete, unambiguous, consistent, ranked, verifiable, modifiable, and traceable. *Statement* (1) helps developers to write a complete, correct, precise, and consistent requirement. Then, using the JSON template, it is possible to verify a requirement in the following phases of the SDLC. Moreover, in the case of a modification, both the written requirement and the JSON must be changed. A modification can occur for several reasons, for example, for a conflict arisen with another requirement or because a stakeholder wants to change a specification. Then, traceability is guaranteed by the transversal activity as we have discussed earlier. However, we will also discuss about it in Chapter 4. Moreover, the JSON code helps the developers understand how traceability must be implemented and considered among requirements. In fact, if two or more requirements have a common goal or related

characteristics (i.e., the same characteristic or the same goal), they can have a traceability relation. Finally, it is possible to create sub-requirements that will specialize a general requirement avoiding the possibility that the general requirement will be verbose. This fact enhances that a requirement will be correct, complete, consistent, traceable, and unambiguous.

4. The last step of the methodology concerns the final elicited requirement composed as shown in Table 3.1. However, as we can see, there is a dotted line moving from Step 4 to Step 1. This arrow represents fundamental feedback that allows the developers to rewrite a requirement according to new or modified needs or solve a conflict among requirements. In this case, the developers will follow the steps presented earlier to elicit the new requirements.

In Section 5, we will provide a complete use case scenario in order to show how the TrUStAPIS method must be implemented along the K-Model.

3.3.10 TrUStAPIS methodology discussion

This method provides in-depth research focusing on the requirements elicitation process.

Firstly, it is essential to highlight that the more time is spent in this phase, the more money is saved, and the quality of the product is fulfilled [56]. The benefits of such a requirements elicitation process are shown in [85] considering security requirements. In addition to them, in this thesis, we consider other requirements connected to security and trust. This holistic approach is fundamental in providing a complete requirements elicitation process.

Secondly, a step-wise systematic methodology helps to reduce subjective issues during the requirements elicitation process. In fact, one well-known issue belonging to the requirements methodologies is that it is difficult to mitigate the subjectivity of the developers eliciting the requirements [60]. Our method minimizes this issue by providing the developers with a systematic step-by-step methodology. Furthermore, the JSON template can be used to automatize the requirements elicitation process, as proposed by Mavropoulos et al. [102].

3.4 Model Phase - Trust Model-driven Approach

After the requirements phase, there is the model implementation. During this phase, it is useful to consider modeling languages such as UML [137] and SysML [56]. Moreover, in order to include also trust management, we consider both trust decision and evaluation models identified by Moyano et al. [112]. Besides, because we design our framework for the IoT environment according to its heterogeneity and dynamics, there will be the possibility that each smart IoT entity must be modeled with different trust models. In fact, depending on the intended IoT architecture (i.e., centralised or distributed), these models will be implemented to define the proper architecture and the type of trusted communications among the IoT entities.

Thus, according to the work developed by Uddin [152], we have identified the possibility of considering trust in UML diagrams. We consider both UML and SysML diagrams in our work because we need the Class Diagram (only present in UML) and the Requirement Diagram (a SysML diagram).

Anyhow, we will extend basic diagrams related to UML and SysML and propose two new diagrams: the traceability and the context diagram.

The basic diagrams that we extend are the use case diagram, class diagram, activity diagram, sequence diagram, state machine diagram, and requirement diagram. We have chosen these diagrams because they permit developers to implement crucial aspects of an IoT entity. Indeed, the use case diagram provides developers with a tool useful to model universal interactions of the IoT entities; then, the requirement diagram allows developers to consider the requirements elicited in the second phase of the K-model to represent them with connections to the other diagrams adding extra features; thirdly, the class diagram will be fundamental to help developers in writing the software of the IoT entity. Finally, the activity, sequence, and state machine diagrams permit developers to specify the IoT entity functionalities and interactions under three different perspectives. Moreover, it is essential to add that even if these diagrams explore different aspects of the modeled IoT entity, they can be combined. For example, a sequence diagram can represent the flow of actions belonging to a use case diagram. Then, the same actions can be specified through an activity diagram

or a state machine diagram. Moreover, considering the new diagrams, the traceability diagram is fundamental to keep track of the connections among diagrams. This feature is enabled because each diagram has a unique ID that allows developers to refer to them uniquely. The traceability diagram can be considered a meta-diagram. On the other hand, the context diagram will be used to map the different contexts that will be considered for the IoT entity. It allows developers to consider all the contexts belonging to the IoT entity since the modeling phase in order to divide the functionalities according to the different contexts improving trust and security.

We describe the basic features and improvements with respect to the original UML and SysML diagrams for each diagram. A critical aspect of defining the diagrams is the consideration of trust and related domains. As we explained earlier, the domains related to trust are usability, security, availability, privacy, identity, and safety [49].

3.4.1 Use Case Diagram

The Use Case Diagram (UCD) is a UML and SysML diagram representing general user interactions with other IoT entities. In its basic form, this diagram is composed of actors involved in the interactions represented by ellipses. They represent actions that can be generic or specific. Other diagrams can represent these actions in order to show how an action can be fulfilled (i.e., an activity diagram can show the activities belonging to a particular action).

The actors have tags representing their roles. These tags are at the top of the actor representation (i.e., a stick man) and their format is «*tag*». Considering the trust domain, possible tags for the actors are the following: «*Trustor*» or «*Trustee*». On the other hand, at the bottom of the actor's representation, there is its name.

However, our UCD extension is enriched with this feature also in the use cases. In fact, this tag represents the specific domain considered for the use case. On the contrary, the name of the use case represents the action. Moreover, in our extended UCD, it is possible to model an actor action with different domains (i.e., security and trust) representing connected “parallel” use cases. In order to design this representation, we create a set of connections for the actions. They are described with

the following tags: «Dependence» and «Interdependence». The former is represented by a line with a direction, the latter by a not oriented line. For example, if action B depends on action A, this relationship is represented by a «Dependence» pointing from action A to action B. «Interdependence» means that both actions depend on each other.

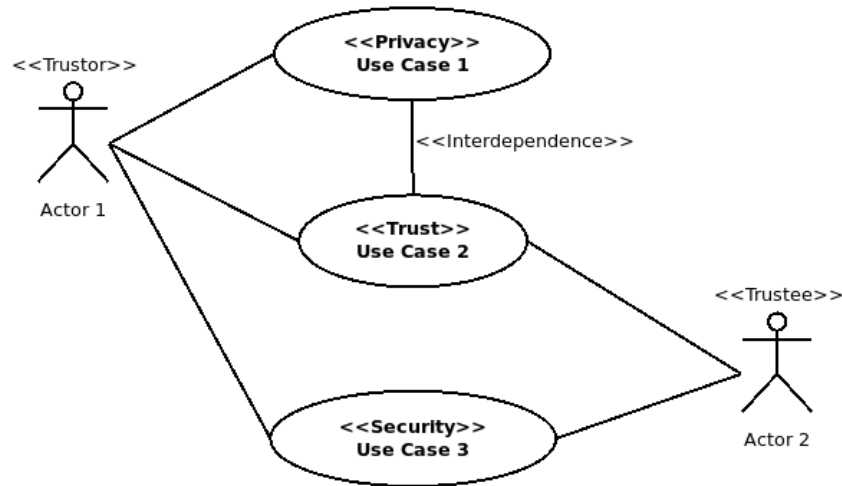


Figure 3.7: Use Case Diagram example

In Figure 3.7, we show a UCD involving two actors, a trustor and a trustee. They share two use cases: the first one is related to trust and the second one to security. Moreover, the trustor is also involved in a privacy use case, which is connected to the trust use case through an «Interdependence». We use general names as Actor 1 and Actor 2, but they can represent any actor existing in an IoT environment (i.e., a smart homeowner, a guest, or an IoT entity).

3.4.2 Class Diagram

IoT entities are both composed of hardware and software parts. For the latter, the Class Diagram (CD) is essential because, with its models, the developers can carefully organize the IoT entity's code to be developed, defining the actions that will be performed. In order to consider this diagram according to our framework, we propose an extension of the CD that is widely used both in Software Engineering

and UML.

In our CD version, the three canonical boxes originally developed in UML [137] represent the name of the class, the attributes, and the operations (or methods). Moreover, we add an extra box concerning the context.

In the case a class is used in every context, it will be empty. Otherwise, it will contain the name of the contexts belonging to that particular class of the IoT entity. In this box, we will represent the contexts using an array because they can be more than one.

Furthermore, we extend the box related to the name, also considering the class domains. In order to include all the domains connected with a specific class, we implement an array. Thus, the field related to the name of the class is represented as it follows:

`<< class_name[Domain(1), ..., Domain(n)] >>.`

A guide for the developer is to be consistent with the domains specified for the classes. Thus, for example, if the domain is related to trust, the attributes and methods need to be specified according to an evaluation or a decision model in order to implement their functionalities in the class. In fact, depending on which models will be considered, the methods implemented in the software will contain rules about decision or evaluation models.

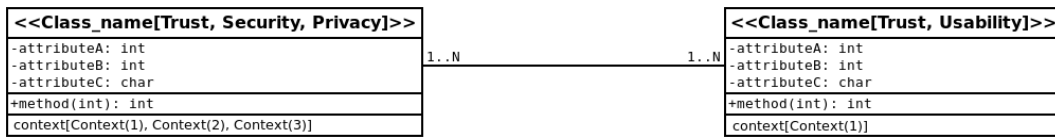


Figure 3.8: Class Diagram example

Figure 3.8 shows an example of a CD with two connected classes. The multiplicity of the connection shows how many instances of the classes could be represented. In this example, we have 1-to-N instances of class A related to 1-to-N instances of class B. Moreover, the domains represented by class A are trust, security, and privacy. On the other hand, class B belongs to trust and usability domains. Then, the attributes and operations are basically used to define the class variables and how they must be considered and computed. About the context field, the class on the left is considered

in three contexts. On the contrary, the other class contains only one context, which is also considered in the first class (for this reason, they are connected).

3.4.3 Activity Diagram

The original Activity Diagram (AD) is mostly an advanced flow chart. In fact, it represents the workflow among activities. An AD can model an IoT entity in order to specify the activities that need to be developed. Besides the activities, an AD contains one pre-condition and one or more post-conditions. Moreover, there can be other optional elements such as decision points, merge nodes, fork, join, and swimlanes. All these elements are defined as follows:

- **Activities.** The activities are the core of this diagram. They basically represent processes that the modeled entity must fulfill.
- **Pre-condition.** A pre-condition is the starting point of the AD.
- **Decision points.** The decision points are used to model alternatives. They use conditions to let the flow continues or ends if one or more conditions are not fulfilled.
- **Merge Nodes.** The alternatives modeled using decision points are reunited to the main flow by merge nodes.
- **Post-conditions.** The post-conditions are the ending points of the workflow. They could be reached because the workflow ends properly or after the decision points.
- **Fork.** Fork elements are used to split the flow into two or more parallel workflows.
- **Join.** Join elements are necessary to reunite the workflows into a single one.
- **Swimlane.** Swimlanes are useful to define the boundaries among the actors that are involved in a particular activity.

In our extended version, we enrich the swimlanes to model the domains rather than the boundaries among actors. So, for example, if a swimlane is related to privacy, this means that every activity belonging to that swimlane will be related to privacy concepts. We can then state that the connection between swimlanes belonging to different domains could represent the connection between UCD actions (such as the ones presented earlier).

An important new feature that we add with our extension is a new element called “*trust trigger*”. Its function is to allow the flow to pass from an activity to another according to a needed trust level. If this value is enough, then the workflow can continue. Otherwise, there can be three situations:

1. The workflow could end with a post-condition.
2. It could continue with an alternative path.
3. It can be possible to have an alternative flow that increases the trust level (i.e., giving more information). In this final case, the workflow can return to the trust trigger in order to check the new trust level.

The criteria to be taken into consideration in a trust trigger can be related to decision or evaluation model aspects. In the first case, it is basically an access control decision. In contrast, in the second case, it is possible to consider trust parameters like reputation in order to compose a single trust level.

There is the possibility to implement a decision point right after the trust trigger. This element can consider the computed level of trust to allow the workflow to proceed to a path or another. For example, it can be possible to let the flow continue in a secondary path (i.e., a less trusted path in the case the trust level is low).

A simple AD is shown in Figure 3.9, where there is a flow finishing with a single ending point. The activities represented are three, and the first one belongs in a privacy swimlane, whereas the other two activities are performed in a trust swimlane. Regarding the trust trigger, we will show an example later when presenting the state machine diagrams.

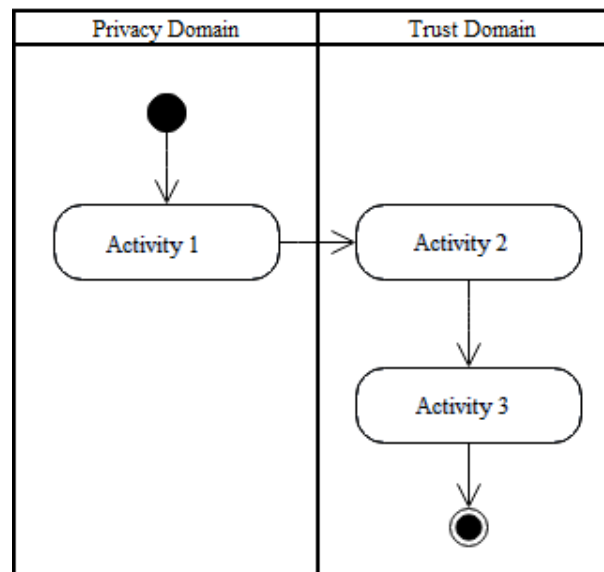


Figure 3.9: Activity Diagram example

3.4.4 Sequence Diagram

The Sequence Diagram (SD) represents the interactions among actors involved in a particular task or process. The principal purpose of an SD is to model the steps related to a particular interaction. Thus, the SD can represent and specify a particular UCD action or an AD activity.

For example, an AD activity can be the following: “Check the Smart Thermostat Temperature”. An SD can be designed to implement this activity (i.e., modeling all the proper steps in order to connect to the smart thermostat or the steps related to the temperature check). Thus, the SD represents all the messages exchanged by the actors or by the IoT entities involved in the modeled interactions.

The actors are represented by parallel vertical lines (known as lifelines). When these actors are active, their processes are represented by a bar. On the contrary, the inactive processes are represented by dotted lines. To represent the messages among the actors, we use arrows. They could be asynchronous or synchronous. An asynchronous message does not need to receive a reply in order to continue the

process. In contrast, a synchronous message waits for a reply before continuing a task. Other important elements for the modeling of an SD are the frames. They are represented by labeled squares. These labels specify the kind of actions performed inside the frame. Thus, the actions can be optional or repeated.

Our extension of the SD includes features about trust. In this domain, as we stated earlier, it is essential to have the possibility to distinguish between decision and evaluation models. For this reason, we provide frames with features related to these models. Thus, for example, if the frame must represent an evaluation model, the SD will be designed considering messages related to the computation of a trust value. On the contrary, if the frame wants to represent a decision model, the interactions must implement the process of giving credentials in order to trust the actors involved. Furthermore, we design frames also for the other domains.

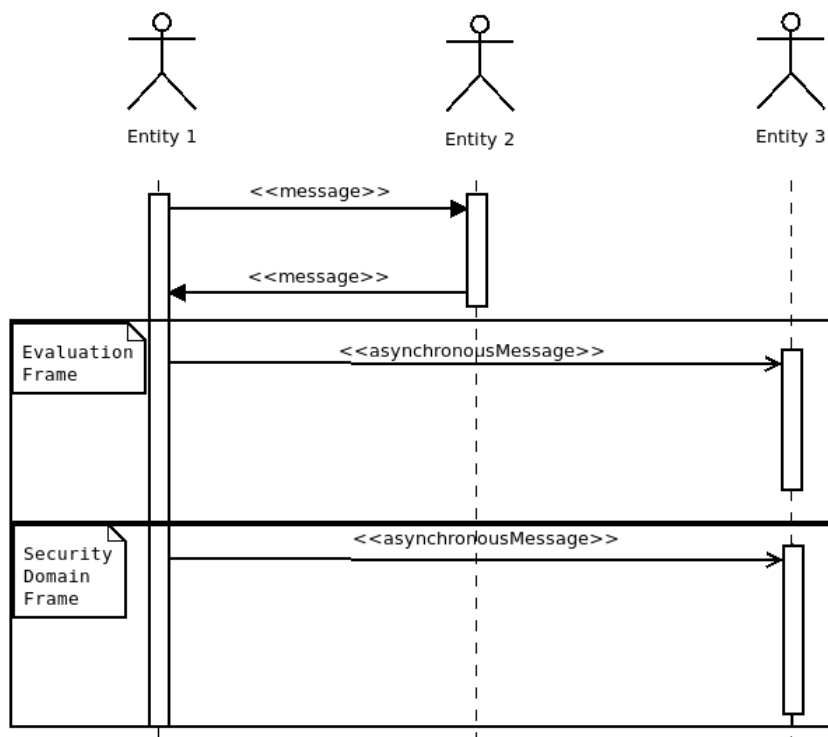


Figure 3.10: Sequence Diagram example

In Figure 3.10, we represent a simple SD showing the messages that can be exchanged by three entities (or functionalities) belonging to the same smart IoT entity

(i.e., a smart thermostat). The SD starts with synchronous messages exchanged by entities 1 and 2. Then, we have a frame related to trust and evaluation models where an asynchronous message is cast from entities 1 to 3. Then, we have a second frame that is related to the security domain.

3.4.5 State Machine Diagram

The State Machine Diagram (SMD) is a diagram useful to model all the states that can be reached by the IoT entity or its functionalities. In its original version, the SMD is composed at least of a starting and a final state. Then, it is possible to have transition states connecting these two states. The change of a state is regulated by transition events that can be modeled with triggers or decision points.

Summarizing, in an SMD, it is possible to have the following states:

1. **Starting state.** It is the starting point of the diagram.
2. **Transition state.** This state is reached during the iteration.
3. **Final state.** This is the last state. It is possible to have multiple final states related to alternative iterations. They can be reached whether the process fails or ends correctly.

It is important to underline that a state can be simple or composite. On the one hand, the transition to the next state is automatic after the functionalities represented by the state are executed. On the other hand, the composite state contains some functionalities or activities that must be performed inside the state. For example, it is also possible to have another SMD inside a single state (i.e., with a starting sub-state and ending sub-states). Here, the process continues until the internal SMD ends in order to proceed to the following state of the main SMD.

In our extended version, as for our AD, we have added the element known as *trigger* (i.e., a trust trigger). Thus, in the case of trust, if the trust level is not enough in order to proceed with the iteration, the procedure ends with a final state. As for the AD, it is possible to proceed after the trust trigger with alternative paths

according to the trust level. Moreover, for a trust trigger, we can consider decision or evaluation model rules.

In the first case, the flow is allowed to proceed only by providing the correct credentials. Anyhow, it can be possible to proceed with a subset of states useful to collect more information (i.e., if a password is needed, it is possible to create a new one or provide other information in order to be accepted).

In the second case, if evaluation model rules are implemented, the flow will proceed only if the trust level is higher than the trust trigger level. If not, we have two possibilities: the flow will end, or it will continue with a sub-state in order to increase the trust level.

About the other domains, it is possible to have other triggers (i.e., identity confirmation, privacy-preserving).

It is essential to highlight that with these triggers, we enrich the potentiality of the SMD with more control. In fact, without these triggers, the flow of the SMD can only end directly or proceed to the next state. Therefore, with these triggers, we increase the modeling possibilities of this diagram.

In Figure 3.11, we show an SMD example. We have a starting point and suddenly a composite state. Here, we have its starting point, then a sub-state followed by a trust trigger (represented by a circled **T**). If we consider an evaluation model, the trust trigger will assess a trust level. If this level is lower than the trigger level, the flow exits from the composite state, reaching a *Not Trusted State* and ending the SMD. Otherwise, if the trust level is higher than the trigger level, the flow reaches a final state belonging to the composite state. Then, it reaches a trusted state, and the SMD successfully ends.

3.4.6 Requirement Diagram

As we discussed earlier, the modeling phase follows the requirements elicitation phase. So, the Requirement Diagram (RD) helps to map all the requirements being elicited in the previous phase and, through the traceability diagram that we will present later, it is possible to map the connections among this and the other diagrams. This diagram

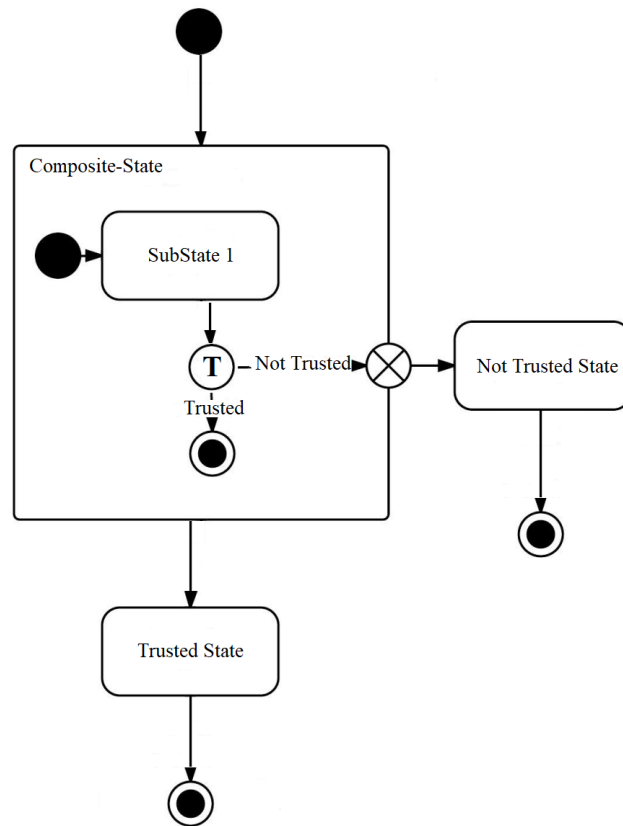


Figure 3.11: State Machine Diagram example

will also be fundamental in the following phases of the K-Model, especially during the development phase. The developers must build the functionalities according to the implemented requirements and models. The use of diagrams and traceability guarantees a holistic implementation of the IoT entity.

RD is a diagram originally implemented in SysML. It specifies capabilities represented by the requirements that shall be implemented by the entity under development. It is at least expressed by two elements: an ID in order to be uniquely identified and a text containing the description of the requirement. Moreover, optional elements can be useful to specify the relationships among requirements and other phases of the SDLC (i.e., the need phase). These operators are the following ones:

1. **Contain**: it shows the connection and decomposition among requirements and sub-requirements.
2. **Derive**: it shows the dependency of a requirement on another one.
3. **Satisfy**: it considers the connection of a requirement with a modeling element (i.e., a use case). The modeling element satisfies the elicited requirement.
4. **Refine**: this kind of connection is related to a model element that must describe the requirement with more details.

We extend the RD considering new stereotypes such as: «*Trust Requirement*», «*Usability Requirement*», «*Availability Requirement*», «*Privacy Requirement*», «*Identity Requirement*», and «*Safety Requirement*». Moreover, we consider also the «*Security Requirement*» stereotype presented in [99]. Each stereotype belongs to a set of requirements, and by this distinction, the developer can better recognize the domain of each requirement.

Moreover, we propose new elements in order to add optional information. The developers can implement these features if they need to specify a requirement further. The elements are the following:

- **Verify**. It is an important parameter connected to the verification phase. It contains specifications about which parameter must be verified.
- **ExpressedBy**. It is an element related to the stakeholder. It includes who expressed the need originating the requirement. The stakeholders are critical because they are the ones that have an interest in the IoT entity under development (i.e., they usually are vendors and customers).
- **ExpressedFor**. This element is specified for the IoT entity as a whole or a part of it (i.e., a subsystem) that the requirement must cover.
- **NeedSatisfied**. This element specifies which is the need originating the requirement. It can be connected to the element *ExpressedBy*.

- **ModelConnected.** This element represents the modeled diagrams to fulfill the requirement. It can be connected with the *Satisfy* operator.
- **RiskCovered.** It is possible that with a requirement, a particular risk is mitigated. This element represents that risk.
- **ThreatMitigation.** A requirement can solve or require threat mitigation. With this element, the requirement is connected to this information.

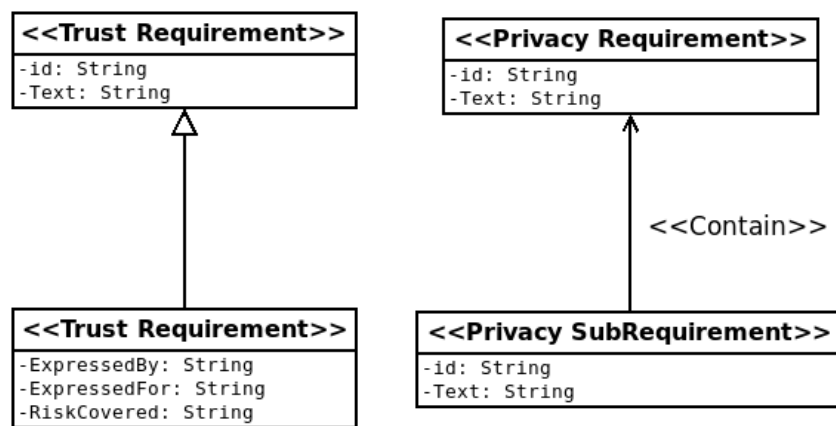


Figure 3.12: Requirement Diagram example

A requirement can have one or more sub-requirements. Usually, they specify additional information that must not be included in the main requirement. We can see an example of requirements and sub-requirements in Figure 3.12 that shows an example of an RD. There are two modeled requirements in this figure: one trust and one privacy requirement. The basic elements are written into the first box for both of them: the ID and the text related to the requirement (a sentence represents it). The Privacy Requirement includes a Contain extension connected to the privacy sub-requirement. In this case, privacy sub-requirement is a specialization of the privacy requirement. Concerning the trust requirement, we can see three optional extensions related to it: *ExpressedBy*, *ExpressedFor* and *RiskCovered*.

The optional information enables developers in order to specify a requirement better. As we can see, the optional elements are a sort of extension of the main requirement. We have chosen this approach to have the central part of the requirement more straightforward than it could be with the additional and optional features.

Moreover, we can state that there is no connection between trust and privacy requirements.

A graphical view can be complicated in the case a huge number of requirements must be represented. Thus, we implement the database notation proposed in [49] that guarantees traceability among requirements.

3.4.7 Context Diagram

In our model-driven approach, we create two new diagrams. One of them is called Context Diagram (XD), and it is essential in order to map and model the context related to the developed IoT entity. Using this diagram, the developers map the dynamic behavior of the IoT entities considering all the contexts connected to them.

Moreover, the XD helps developers in considering the possible problems due to the connection of the different contexts within the same IoT entity. Thus, for example, if an IoT entity is used to perform different tasks belonging to distinct contexts (i.e., calendar, weather, and bank), it is essential to keep them separated in order to avoid sharing important information among different contexts and users. In fact, it is possible that a user can be trusted and involved in a particular context (i.e., calendar), but that is not trusted in another (i.e., bank).

According to our framework, each context can be connected to a particular domain (i.e., security, trust). For each of them, the developer must keep separated the users involved and which kind of information they are able to access and manipulate. Moreover, it is possible that more domains are involved in the same context. In this case, the developers must model the context considering the characteristics related to each domain. These characteristics have been presented in 3.3, and they are useful to specify the contexts and the functionalities of the IoT entity.

An XD can represent a set or a sub-set of contexts belonging to an IoT entity. Figure 3.13 shows how an XD can be represented. In this example, an IoT entity will participate in four different contexts. For each of them, different domains belong to it. We can see that there are contexts belonging to a single domain (i.e., context 1) and contexts belonging to multiple domains (i.e., context 3). This diagram helps

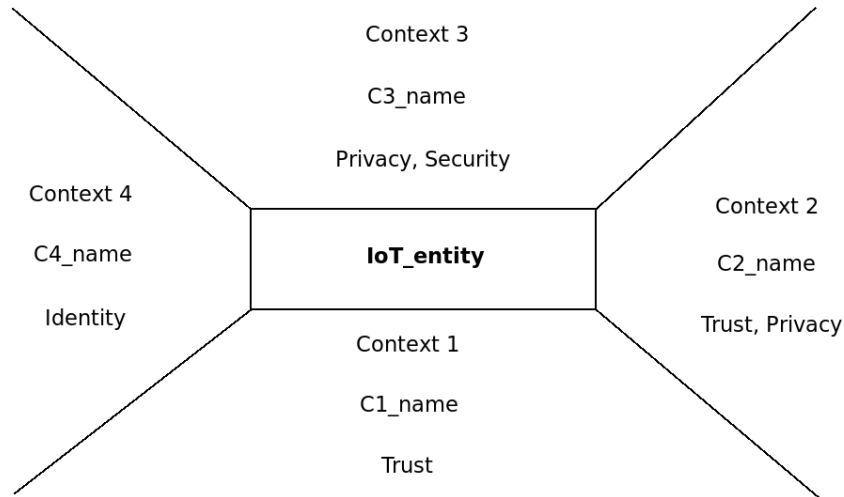


Figure 3.13: Context Diagram example

developers to define the functionalities of the IoT entities according to the considered domains.

Moreover, it is also possible to represent the contexts through a database notation. The table of the database is composed of three columns. The first contains the context ID that is unique and considered as the primary key of the table. Then, the second column includes the context name. Finally, the third one specifies the domains necessary for the related context. The database template is shown in Table 3.2.

Table 3.2: Context Diagram - Database template

Context_ID	Context_Name	(Domain_1, Domain_2, ..., Domain_N)
------------	--------------	-------------------------------------

Considering the example shown in Figure 3.13, the related database is presented in Table 3.3.

Table 3.3: Context Diagram Example - Database view

Context_1	C1_name	[Trust]
Context_2	C2_name	[Trust, Privacy]
Context_3	C3_name	[Privacy, Security]
Context_4	C4_name	[Identity]

3.4.8 Traceability Diagram

The second new diagram is the Traceability Diagram (TD). It is important to record the connections among all the diagrams. Thus, the traceability diagram can be considered a meta-diagram. Moreover, when two or more diagrams are connected among them, they create a cluster.

Using the TD, developers can have a holistic view of the models because we can map all the connections among the modeled diagrams. Moreover, using TD, it is possible to avoid or mitigate domino effects due to the deletion or modification of a diagram. In addition, considering that each diagram is mapped with a unique identifier, it is possible to store these data in a proper database. This feature enables the possibility of showing this diagram graphically (showing the connections among diagrams) or being stored in a traceability database.

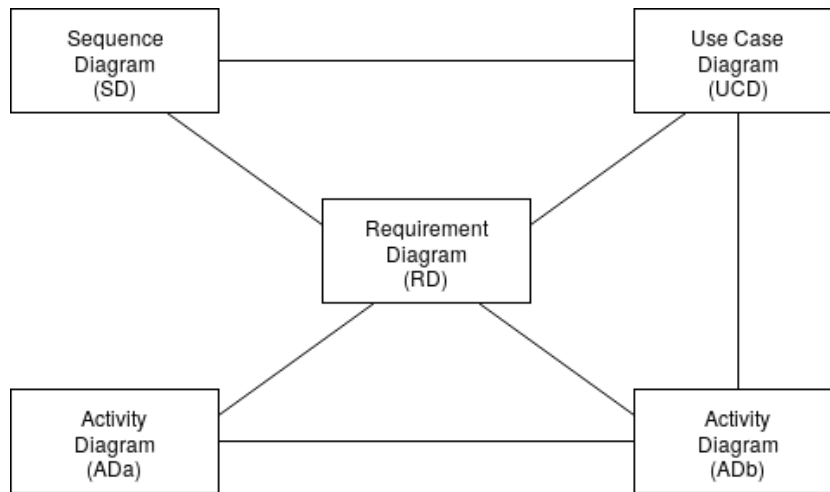


Figure 3.14: Traceability Diagram example

In Figure 3.14, we show an example of a graphical TD. The diagrams are represented by boxes containing the type of the diagram and their IDs. Lines represent the connections among the diagrams. As we can see, there is a requirement diagram (RD) (at the center) that is connected with the other diagrams. This connection means that the requirements represented in RD1 are modeled through the other diagrams. These diagrams are a Sequence Diagram (SD), a Use Case Diagram (UCD), and two Activity Diagrams (ADa and ADb). Regarding the connections, we can see

that SD is connected to RD1 and UCD, but not to ADa and ADb. This means that the diagram represented in UCD contains the requirements considered in RD1 and represented by the sequence diagram in SD1. About the other connections, we see that ADb is connected to ADa, RD, and UCD. Finally, Ab is connected only to RD and ADa.

In the case the diagram grows, it can be challenging to show the TD graphically. Thus, we can represent it by a database notation.

TD database is composed of two columns. The first one is related to the diagram ID, and a unique string represents it. The second one is related to the diagrams connected to the one represented in the first column, and it is represented by an array of strings. The general composition of the database is shown in Table 3.4.

Table 3.4: Traceability Diagram - Database view

Diagram_ID	(Diagram1_ID, Diagram2_ID, ..., DiagramN_ID)
------------	--

Considering the example of Figure 3.14, the related database is presented in Table 3.5.

Table 3.5: Traceability Diagram Example - Database view

RD1	(SD1, UCD1, AD1, AD2)
SD1	(RD1, UCD1)
UCD1	(SD1, RD1, AD2)
AD1	(RD1, AD2)
AD2	(RD1, AD1, UCD1)

3.4.9 Methodology

In order to show how the model-driven approach must be implemented, we show a step-by-step methodology. It is shown in Figure 3.15.

1. The first step includes the data collected in the previous phases of the K-Model (i.e., Need and Requirements). This information is necessary to produce the

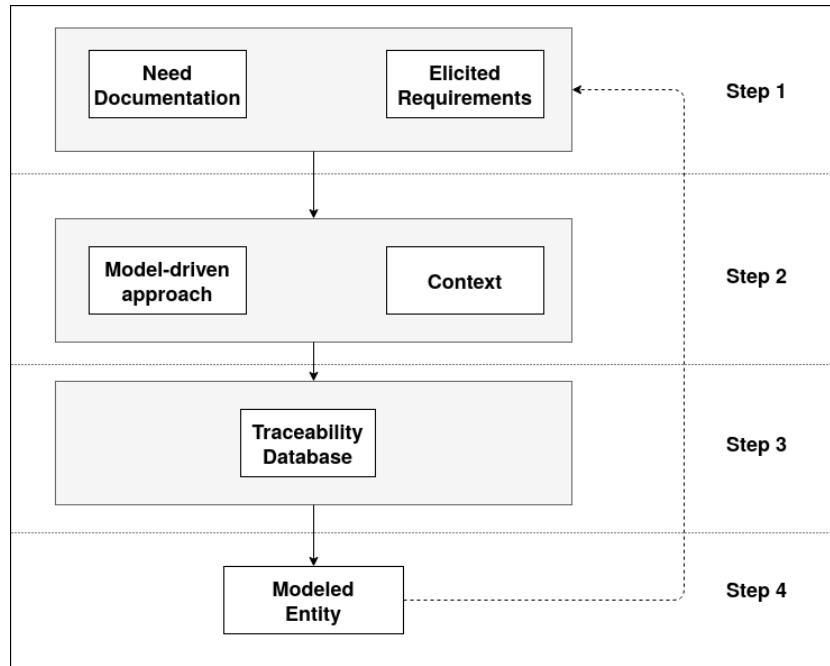


Figure 3.15: Model-driven approach: step-by-step methodology

proper diagrams and design the smart IoT device according to requirements and needs.

2. Secondly, the developers can draw the diagrams following the model-driven approach and analyzing the IoT entity's context. We have stated that the context is heavily conditioned by the environment and the IoT entity's scope. Considering the diagrams that we have shown earlier, they can be drawn in any order, but the preferred one is to consider the requirement diagram firstly according to the elicited requirements of the previous step. Then, it is better to draw the use case diagrams collecting all the general actions that the actors and the IoT entity will perform. The context must be considered for the whole step. Therefore, the context diagram can be drawn at the start and terminated at the end of this step, considering the modeling process dynamically. Then, the developers will use the class, activity, sequence, and state machine diagrams to specify the general actions modeled by the use case diagram. The traceability diagram can also be drawn during the development of the other diagrams to map their relationships from the origin, but traceability will be considered in

the next step.

3. The third step is entirely related to traceability. Thus, considering the traceability diagram created in the previous phase and all the important data related to the other diagrams, it is reasonable to create a traceability database. This information will avoid domino effects in the case a diagram must be modified or deleted. This step will be explained in Section 5.3.8.
4. The fourth is the last step of the methodology, where the modeled entity is delivered, and it will be developed in the subsequent phase of the K-Model. However, if some modifications are needed (for example, because the developers did not address a requirement or a risk was partially or not covered), there is the opportunity to come back to step 1 to implement these modifications. Indeed, developers need to consider traceability in order to make these modifications.

This systematic methodology assists the developers to follow a guideline to draw the models appropriately.

In the following section, we will analyze the central phase of the K-Model.

3.5 Development

The development is the core phase of the K-Model. This phase transforms the needs, requirements, and models in the product that will be verified and validated in order to be distributed to the customers. In the IoT, the challenges are numerous because of the dynamicity and heterogeneity of this technology.

Thus, we believe that in order to holistically consider all the information collected in the previous phases of the K-Model, the developers must organize their work in a schematic way. Our proposed approaches help them fulfilling this goal. Therefore, the top-down approach that we will present in Section 3.5.1 allows developers to consider functionalities firstly from a generic perspective and then in a specific way. On the contrary, the bottom-up approach that we will discuss in Section 3.5.2 considers contexts from a specific point of view to a more generic one. The utilization of both

approaches helps developers better to consider all the fundamental aspects of the IoT entity. Finally, both the approaches are considered during the trusted finite-state development that we will propose in Section 3.5.3.

In order to show the steps of the development phase, we will present in Section 3.5.4 the order of utilization of our proposed approaches.

3.5.1 Top-Down approach

During the development of an IoT entity, a useful methodology is to analyze the IoT entity under development following a top-down approach.

Several methodologies implement this approach such as the Work Breakdown Structure (WBS) or the Functional Breakdown Structure (FBS) [36]. Moreover, in the previous phases of the K-Model, we have focused on how important it is to consider trust domains. For this reason, we think that it is useful to mix the FBS, also considering the related domains of each functionality.

We named this top-down approach as FDBS (Functional Domain Breakdown Structure). In this approach, it is fundamental to highlight each functionality according to its domain. However, it is possible that a particular functionality could belong to more than one domain.

According to the original FBS structure, we create a descending tree where the root is related to the IoT entity, and the children define its functionalities. The final leaves of the FDBS tree will contain basic functionalities. For the FDBS, we create another parameter denoting which domain is considered for the proper functionality or sub-functionality (i.e., trust and privacy). As it is possible that two or more requirements could be the same and belong to different domains [48]; thus, it is possible that a single functionality belongs to different domains. However, in this case, we will not have two or more functionalities, but we will have a single functionality belonging to more than one domain.

Figure 3.16 shows an example of FDBS. There are three levels. The IoT entity is set at the top level. Then its main functionalities are set in the medium level, splitting them into sub-functionalities at the bottom level. As we can see, the domains are

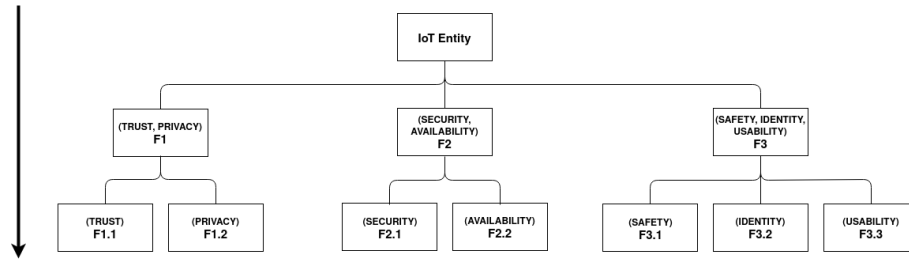


Figure 3.16: Functional Domain Breakdown Structure (FDBS)

represented at the top of the box containing the functionalities. However, in order to show how to implement our top-down approach in a use case scenario, we will present a specific example in Chapter 5.

3.5.2 Bottom-Up approach

For the bottom-up approach, the essential elements of a system are firstly analyzed and implemented. Then, the developer analyzes composed elements in order to proceed from a specific to a general view of the entity.

As we explained in the previous sections, it is crucial to take context into consideration during the SDLC of an IoT entity. Therefore, we will use this approach to model all the contexts belonging to the IoT entity under development. In fact, the IoT entities can participate in different contexts, and some of them shall be separated from the others. Strongly related to the contexts, we also consider the domains (i.e., trust and security). However, there is the possibility that some contexts share functionalities, and they can be considered together under a super-context. This super-context will include the single domains belonging to the contexts in the lower layer.

The bottom-up approach is presented in Figure 3.17. It starts from the single contexts considering the ones at the bottom level and their domains. Then, the contexts having commonalities are considered together under a super-context. In this general representation, we have three levels, and the top-level belongs to the IoT entity as its whole.

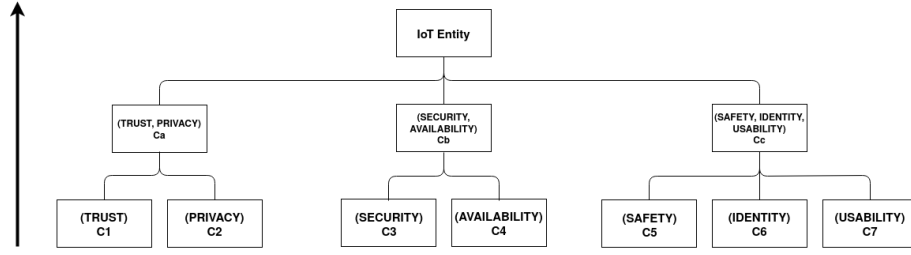
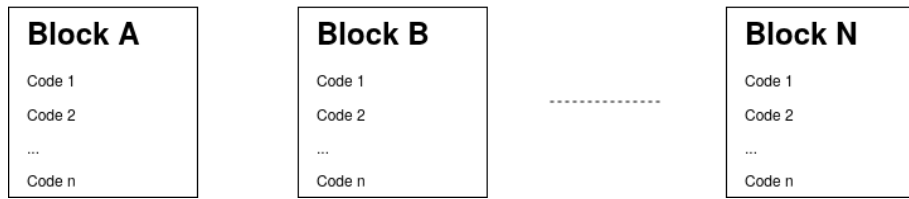


Figure 3.17: Bottom-Up Approach (Context)

Figure 3.18: Block Development (BD): each block is a *trust island*

3.5.3 Trusted Block Development

In order to consider and develop software that is fundamental for any IoT entities, we propose a trusted finite-state development.

This approach helps developers to delimit the software according to the contexts and functionalities highlighted in the bottom-up and top-down approaches. This development style is fundamental to keep separated the different codes.

Figure 3.18 shows that the blocks are separated, and they can contain several sub-blocks of code. They can be sequential or not, but it is essential that they are separated. This is fundamental in order to preserve the separation among functionalities and contexts. This programming technique allows developers to create boundaries related to trust, creating “*trust islands*”. In fact, if we consider variables existing only in a particular block, we can create trust operations according to the users or functionalities belonging only to that particular block. In fact, this design is helpful in keeping the roles of a user separated according to a particular context. For example, a user can be considered and trusted in block A, but it cannot be considered or not trusted in block B. We will show an example of this part in Section 5.4.3.

3.5.4 Implementation Approaches

These approaches must follow a step-by-step methodology in order to be effectively implemented within the K-Model. This methodology is presented in Figure 3.19.

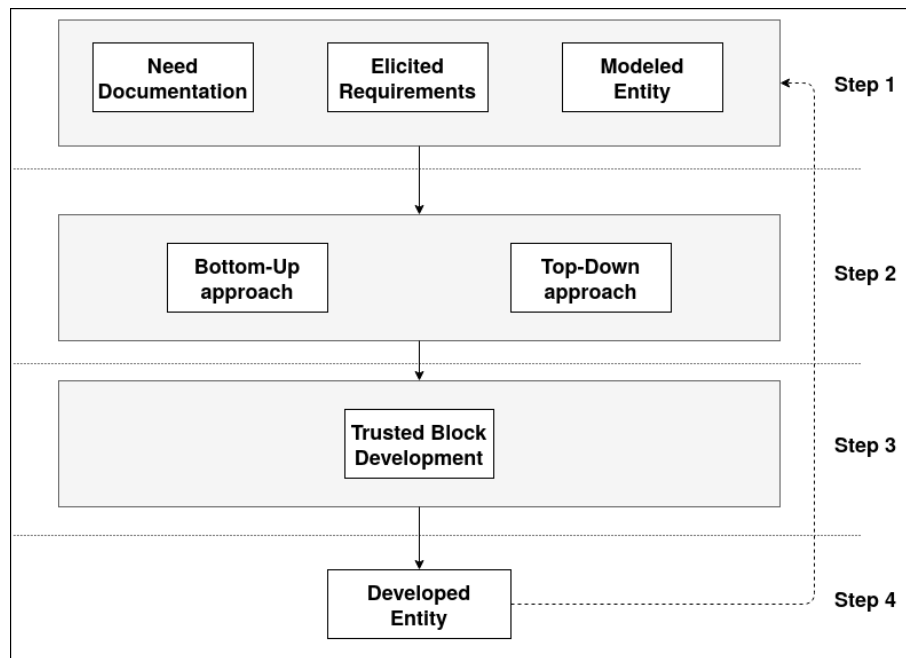


Figure 3.19: Step-by-step methodology

The steps are the following:

1. The first step corresponds to the output of the previous phases of the K-Model (i.e., Need, Requirements, and Model). In fact, all the tasks performed up to this step must be considered to develop the IoT entity. The needs specify the intended IoT entity. Thus, the requirements have been elicited according to them, and the implemented models create useful guidelines to develop the IoT device in this main phase of the K-Model.
2. Then, in Sections 3.5.1 and 3.5.2, the proposed approaches are implemented in the second step. A context diagram will be handy in the bottom-up approach. On the other hand, the other diagrams will be fundamentals during the implementation of the top-down approach.

3. Thirdly, as we specified in Section 3.5.3, according to both the approaches, we need to implement the trusted finite-state development.
4. The fourth step of the methodology corresponds to the final developed entity that will be verified and validated in the following phases of the K-Model. Anyhow, it is possible to come back to step number one in the case some modifications are needed.

In the next section, we provide a use case scenario that realizes the implementation approaches presented in Section 3.5.

3.6 Verification and Validation

Verification and, later in the SDLC, Validation are two critical phases that are useful to prove that the IoT entity under development has been adequately developed. If validation must be performed only for the final product, verification can be implemented in any phase in order to find errors or modify the original plan. However, it is essential to consider verification as a separate phase after the development phase because every verification test must be made before proceeding to the validation phase.

3.6.1 Verification

A definition of verification appears in the Project Management Body of Knowledge (PMBOK) [41] where verification is considered as “the evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation”.

This phase is fundamental in order to verify that *the entity has been built in the right way*. This means that all the specifications have been followed in the correct way. In the verification phase, the functionalities of the entity are tested as well as its correct implementation. Requirements related to the functionalities and not strictly

dependent on the environment are checked in this phase, as well as the implemented models.

Moreover, this phase is intended to be performed in order to check that the developed IoT entity meets a set of design specifications.

After the development phase, the verification procedures are performed through tests that model or simulate a part of a determined functionality of the IoT entity. This procedure's success acknowledges the fact that the smart object is working correctly (according to the tested functionality). It is also possible that the same tests are performed continuously to assess that the developed IoT device meets the requirements, specifications, and regulations as time progresses.

Verification is performed as a whole in this phase, but it can be performed during the previous phases of the K-model (if necessary) to test the functionalities of the device partially. It can be useful to perform these tests in advance in order to make the proper modifications early in the SDLC. Anyhow, in the verification phase, all the functionalities must be tested, so it is possible to assess that the IoT entity has been adequately developed.

As stated by Arthur et al. [10], about verification and validation, a powerful tool that can be used is called *Independent Verification and Validation (IVV)*. This is a mechanism that can be exploited in order to mitigate the growing complexity related to the expansion of modeling and simulation problems. The authors organize a literature review about verification, validation, and IVV concluding that: “(a) validation is the primary focus of most modeling and simulation efforts, (b) verification plays only a secondary role, and (c) independent V&V is, for all practical purposes, being ignored”. The authors aimed “to raise the awareness of the benefits and applicability of IVV within the modeling and simulation community”. They described a step-by-step methodology on how to apply IVV to a particular SDLC model.

SysML can be helpful because, as we presented earlier, it includes several diagrams, but one of them is fundamental in the verification phase: the requirements diagram. During the requirements elicitation process and the modeling phase enforcing the requirements diagram, the developers implement the *verify* process giving hints on how to reproduce the verification process later on during the SDLC. It is

represented by an arrow stereotype.

Moreover, testing is helpful in this phase to check if the functionalities work as intended. Anyhow, test and verification represent different goals. Linhares et al. [92] stated that “test is realized on the implemented system”. For this reason, testing is useful both in the verification and validation phases. It involves a set of possible inputs testing the system’s functionalities under consideration to check if something wrong occurs. Usually, testing in SysML allows ensuring that a specific set of test scenarios are accepted. On the other hand, the authors stated that formal verification “allows us to be sure that the standard requirements are fulfilled by the system specification”.

Another mechanism that is implemented during verification processes is the inspections. Fagan et al. [44] described for the first time the inspection processes which were previously considered by Ackerman et al. [3] that described the inspection processes as fundamental for the verification phase. In fact, the latter declares that “the use of software inspections improves product quality, as well as the productivity and manageability of the development process”.

For our purposes, we state that the verification phase of the K-Model is keen to analyze the requirements according to the models and test the functionalities to prove that the requirements and the functionalities are well-formed and the “system has been built right”.

The JSON template presented in 3.3 helps developers to analyze them and test the functionalities.

At the end of this phase, we can state that “the trusted IoT entity has been built right”.

Figure 3.20 shows that the verification tests are performed according to the requirements and models designed in the previous phases of the K-Model. These verification tests are presented in the following sub-section.

3. 6. 1. 1 Methodology

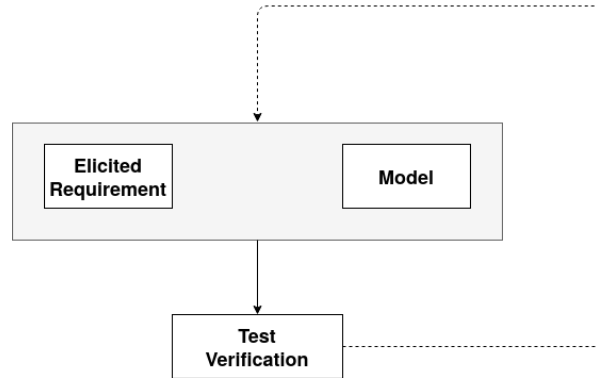


Figure 3.20: Verification

In this section, we present a step-by-step methodology on how to perform verification. The steps are shown in Figure 3.21.

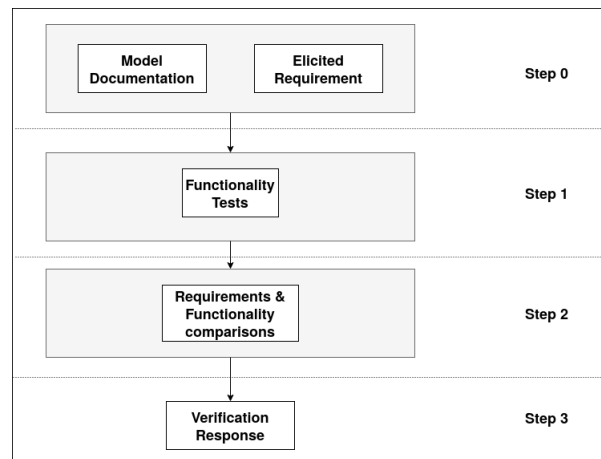


Figure 3.21: Test Verification: step-by-step methodology

The output of each step is used during the following steps.

1. Step zero is related to the comparison of the originating model documentation (collected during the third phase of the K-Model) and the elicited requirements. In fact, we need to compare the requirements and models to verify the functionalities of the developed IoT entity as the developers have modeled and planned. It is also represented in Figure 3.20.
2. The first step is performed to execute tests related to the functionalities of the developed IoT entity. It is fundamental in order to check that there are no

malfunctions and the functionalities perform as intended. In the case of trust functionalities, it is checked that only trusted users can perform determined actions (i.e., to access the IoT device).

3. In order to check that the requirements have been respected, the functionalities tested in the previous step must be connected to the originating requirements, so it is possible to certify that they have been designed in the correct way.
4. The response of the comparison performed in step 2 is the input for the final step. If the functionality follows the requirements, then the verification test is successful. Otherwise, the functionality must be modified according to the elicited requirements. This feedback is represented in the K-Model, and it is necessary in order to develop the correct IoT entity.

This systematic methodology helps the developers to follow a guideline that helps them to test all the functionalities of the developed IoT entity.

3.6.2 Validation

Validation is defined by the PMBOK [41] as “the assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification”.

By validation we mean that *the right entity has been built* [67]. This means that the desired entity has been developed as the users and vendors wanted and that meets the operational needs of the users.

This phase checks that the needs have been met and the IoT entity works appropriately in a real system’s environment. All the requirements are checked, assuring that the entity fulfils its intended purpose.

The developed entity may have passed the verification phase, but it fails when validated. This situation can happen when the entity has been built for the specifications, but the specifications themselves fail to address the needs of the users.

In the State of the Art, there can be different categories of validation.

Process validation has been analyzed in [82], where the authors discussed guidance for the process validation. This guidance promotes a “lifecycle” approach built in three steps. The first step regards a process design based on the know-how gained during the development of the product, and it is “the beginning of a structured and interconnected chain of validation evidence”. Then, the second step is about the process qualification, where the process design is “evaluated and assessed to determine if the process is capable of reproducible commercial manufacturing”. Finally, the third step is related to continued process verification. It is important to underline the difference between “continued” (i.e., ongoing) and “continuous” (i.e., without interruption). In addition to these steps, in the guidance are provided recommendations on creating proper documentation and using analytical methods during process validation.

Then, retrospective validation is executed for a product that is already distributed or sold to the customers. It is performed following the written specifications based on the collected documentation produced during the SDLC. If any critical data is missing or discovered after the utilization phase, then the product has been completed partially [51] and a restyling is necessary. This kind of validation is necessary if process validation is missing or incomplete, in the case there are changes of standards and regulations affecting the distributed product, and in the case, a disposal item is about to be revived.

Next, partial validation is often used for research purposes or prototypal studies if the time before the utilization phase is constrained. In this case, the tests are performed only on the most significant parts or functionalities. The developers can perform the ranking of the functionalities through a decision-making process.

Another type of validation is called re-validation or periodical validation. It is performed on items that are dismissed, repaired, relocated, or after a fixed expected time. Moreover, re-validation can be performed in the case a modification of the product or item is needed.

Finally, concurrent validation is conducted in parallel with a routine process of services, manufacturing, or engineering.

Independently from the kind of validation, all of these techniques have determined

tests to be performed. We can see them in Table 3.6.

Table 3.6: Validation tests

Tests
Selectivity/specificity
Accuracy and precision
Repeatability
Reproducibility
Limit of detection
Limit of quantification
Curve fitting and its range
System suitability

These tests are based on the fact that the product, the equipment, and the functionalities to be analyzed constitute an integral system, and it must be evaluated as it has been developed in its whole.

For our purposes, we can state that the validation process is based on the fact that “the right system has been built”. In order to clarify this, we need to check that the IoT entity and its functionalities match the stakeholders’ needs.

Each need is represented by a statement, and it is represented by one or more stakeholders. As we mentioned earlier, stakeholders are the “persons” having an interest in the system. Usually, it can be for monetary gain or to create a new technology to improve human life.

$$Need\#X = (Stakeholder, Statement)$$

The IoT entity can be represented by a number of needs and stakeholders. As we explained earlier, developers must collect these needs in the first phase of the K-Model and check if some conflicts arise among them. Then, in the requirements phase, the needs are analyzed and transformed into requirements.

Here, in the validation phase, these needs and requirements are checked together to analyze if they are respected and the functionality of the IoT entity represents the original needs.

At the end of this phase, we can state that “the right trusted IoT entity had been built”.

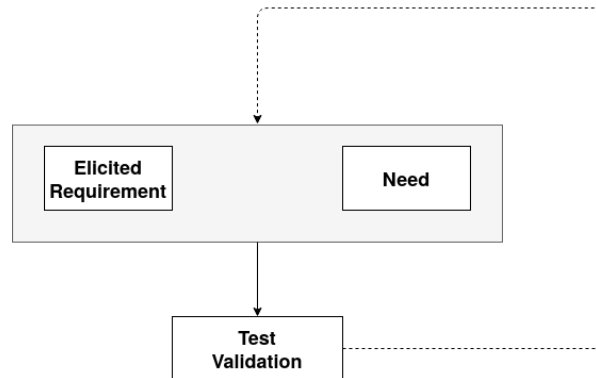


Figure 3.22: Validation

Figure 3.22 shows this representation of which items are needed to perform validation tests (i.e., the elicited requirements and the needs). Then, if a validation test fails, it is possible to follow the feedback to the requirements and needs to solve the issue.

The validation tests are presented in the following sub-section.

3. 6. 2. 1 Methodology

In this section, we present a step-by-step methodology on how to perform validation. The steps are shown in Figure 3.23.

The output of each step is used during the following steps.

1. Step zero is related to the comparison of the originating needs (collected in proper documents) and the elicited requirements. This step is basically similar to the one performed in the second phase of the K-Model, where the developers elicited the requirements from the needs. In this phase, the needs and the connected requirements are compared. This step is helpful to discover if there are some missing or wrong requirements.
2. The first step is performed to execute tests related to the functionalities developed following the elicited requirements in order to satisfy the needs.

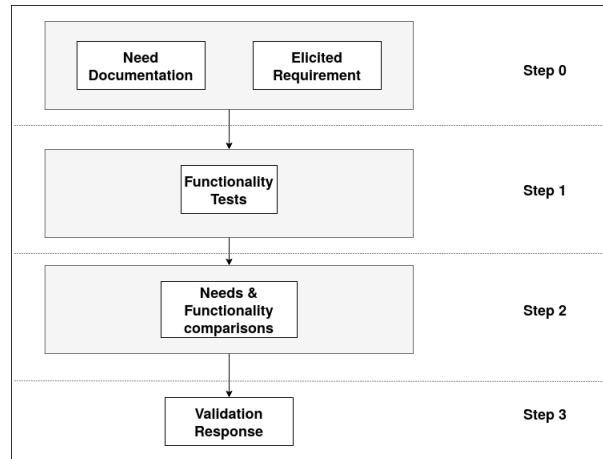


Figure 3.23: Test Validation: step-by-step methodology

3. The comparisons between needs and functionalities are performed in the second step, where the output of the test is compared to the originating need.
4. The response of the comparison performed in the previous step is considered in the last step of the methodology. If the functionality satisfies the need, the validation test is successful. Otherwise, the functionality must be modified. This means that the developers must go back in the K-Model cycle to the previous phases in order to perform the correct modifications. However, if performed correctly, all the previous phases minimize the risk that this possibility should happen.

This methodology provides developers with a guideline to test the developed IoT entity in its intended environment.

3.7 Utilization

The last phase of the K-Model is related to the utilization of the developed IoT entities. In fact, after the entity has been verified and validated, it is finally possible to use it in its intended environment. As we represented in the K-Model shown in Figure 3.1, there are connections among this phase and the needs and requirements

phases. These connections are useful to control that the entity works as intended to satisfy the needs and the elicited requirements.

We can state that, during the utilization phase, an IoT entity can belong only to three states about being in an IoT network:

- Join: when an IoT entity enters or is connected to an existing network.
- Stay: after the join state, an entity can be rejected or accepted. If accepted, the entity will stay in the network until something occurs (i.e., leaving the network, ban).
- Leave: an entity can leave a network because it is changed or expelled. When an object leaves a system, it can join another one or be dismissed.

In an IoT environment, these three states (join, stay, and leave) will often be present according to the dynamicity of the field and the context. This requires that trust should be strongly considered in order to legitimate the relationships among the IoT entities. For this reason, it is necessary to develop a smart entity capable of computing trust values in real-time. Moyano [113] proposed the idea of *trust@run.time*, which is an aspect that we took into consideration for this thesis. Moreover, trust decision-making, trust modeling, and trust metrics are needed when an IoT entity joins a new network. So, it is possible to collect trust information about other objects in deciding how to proceed. Moreover, the trust parameters are important in choosing which IoT system can be trusted in order to interact with it. Staying in a network signifies dealing with the dynamicity of the network. This means dealing with other joining, staying, and leaving devices. For this reason, trust can be helpful in organizing the relationships among all these entities. Finally, when an entity leaves a network, there are three possibilities:

- The entity is dismissed.
- It will never come back.
- It will join the network again.

The historical trust value of an entity that has left the network can be useful only in the case it returns to the network. Anyhow, according to the dynamicity of trust and IoT, these values should be outdated. In this case, they can be valid only for a short time period. Trust modeling and metrics can help consider all these different situations.

We have proposed an adaptive trust model that can be used to monitor and control these states.

3.7.1 Adaptive Trust Model

The adaptive trust model comprehends different situations. In our work [50], we have modeled the three possibilities that we stated earlier.

The trust level computed by the adaptive trust model will be fundamental to decide if the new entity is allowed to join the network or stay in it.

For smart homes, we consider the possibility that there is a Smart Hub monitoring the other devices. This device will compute the trust values, and it will have the right to access several Databases (DB) that we illustrate later. Moreover, it could store data related to the actions performed by the IoT entities for forensics purposes, but this is out of this Tesis scope. We assume that this Smart Hub cannot be compromised because it has a Root of Trust ⁶.

According to the need phase, where we identified a segregated architecture, the smart hub will consider the new entities approaching the network through the adaptive trust model. Thus, when a new entity wants to join a network and it is accepted, it should be allowed into the internal or the external network. However, in order to decide if to allow or not an entity to access the network, a trust estimation is needed according to the following parameters: threats DB, reputation DB, context, and risk calculation.

- **Threats DB.** The known vulnerabilities of the smart devices are collected in this DB. In case no known attacks related to the device are present, its trust

⁶<https://www.synopsys.com/designware-ip/technical-bulletin/secure-iot-system.html>

value is high. In the case of known attacks, the greater the danger, the lower the trust value.

- **Reputation DB.** The reputation DB is used to store the devices old reputation values. For example, in the case a new device tries to join the network again, but it has been banned in the past, the smart hub will deny its access. We assume that a ban is performed after a serious security issue. For this reason, a banned device cannot join the network again. Furthermore, avoiding a second opportunity, we prevent Whitewashing Attacks (see Chapter 5.6.3). We assume that both the DBs are secured and encrypted. In addition, they are stored in the internal network where we assume that a malicious entity cannot access, because of the implementation of the join, stay and leave phases (as we will show later).
- **Context.** The context depends on the environment, purpose, and services that the device provides alone or with the other smart devices. The more critical a device is, the higher the necessary level of trust.
- **Risk Calculation.** We consider three parameters to calculate the risk. The first parameter is the likelihood (L) of an event; this is the probability that a situation that harms the system can occur (either an attack or a malfunction). The second parameter is the severity (S) of the effect that a malfunction or an attack can have on the system. The more critical the component involved is, the more critical the threat is for the whole system. Finally, there is a parameter that is usually not taken into consideration but one which we think is crucial to calculate the risk: the detectability (D). The detectability is the possibility of a malfunction or an infected device can be detected. If an attack occurs and we cannot detect it, the system will fail or be manipulated. We have considered either detectability or likelihood separately because the likelihood is related only to the probability that an event occurs, even if we detect it or not. These values will be computed together, calculating a final risk level. The result can be high, medium, or low. In the case that the calculated risk is high, the device cannot be added to the network, or it must be banned. In the case

the risk value is low or medium, the device can join or stay in the network depending on other criteria. We will discuss this more deeply about risks in Section 4.7.

3.7.2 Trust Estimation - Join, Stay and Leave

In our model, trust estimation considers different parameters. It is computed to establish whether a new entity can join the network or not, as shown in Figure 3.24.

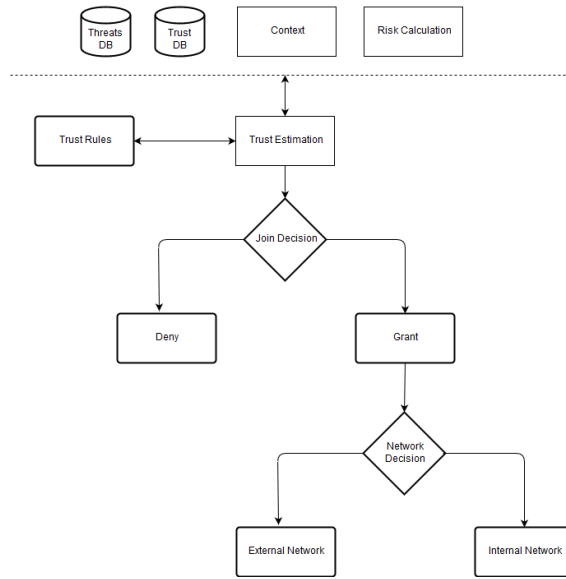


Figure 3.24: Adaptive Trust Model: Join Decision

3. 7. 2. 1 Join

When the smart homeowner connects a new entity to the home network, the centralised monitor (i.e., a smart hub) is contacted by the new entity that asks permission to join the home network. Thus, the smart hub performs a trust estimation about the entity (i.e., who is the owner or if there are known threats concerning the device). In the case of positive trust estimation, the smart hub notifies the joining entity about how to interact with the other entities giving it a key for exchanging messages. The rule for joining the other entities depends on the trust estimation parameters. Moreover, which network to join is related to the fact that the new entity

belongs or not to the BYOD paradigm; if the case is affirmative, the new entity can only join the external network.

The join procedure is shown in Figure 3.25, and it is similar to the SDP technique⁷. When the new device joins the network, it sends a broadcast message to communicate with the smart hub in order to be allowed to join the network (action 1). Thus, the smart hub checks the permissions of the new device (i.e., password, owner key, rights), performing the trust estimation. If the access is denied, the smart hub informs the new device that it is not allowed to join the network (action 2). Otherwise, if the access is granted, the smart hub instructs the new device that it can join the network and which other devices are allowed to interact with it (action 2). Then, the smart hub informs the devices belonging to the network that they can exchange information with the new device (action 3). In both actions 2 and 3, the smart hub gives a symmetric key to the *new* device and to the *old* devices enabling them to communicate with each other. The devices must perform an acknowledgment to the smart hub (action 4) and from that moment, the devices can interact among them (action 5).

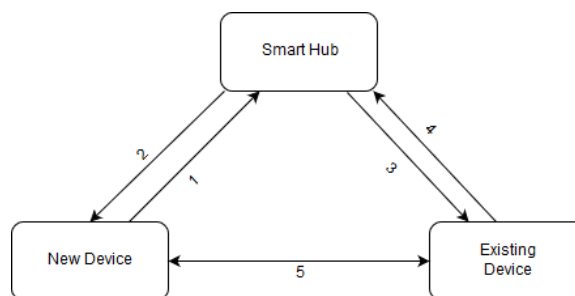


Figure 3.25: Joining a network

3. 7. 2. 2 Stay

When an entity stays in a network, it must be monitored, according to external and internal factors, as shown in Figure 3.26.

During the monitoring, the smart hub checks whether the entities are behaving

⁷<https://cloudsecurityalliance.org/download/software-defined-perimeter/>

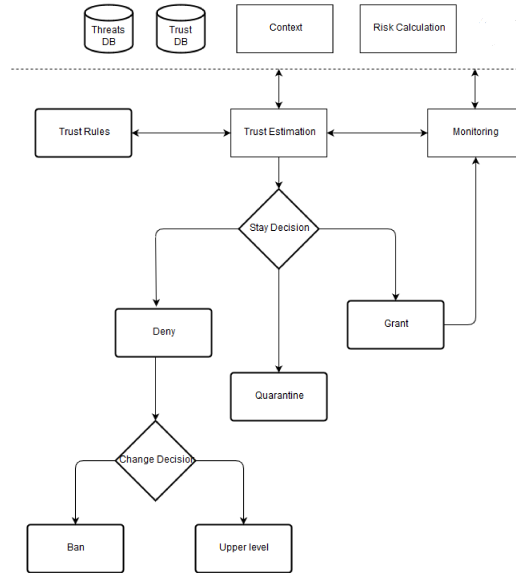


Figure 3.26: Adaptive Trust Model: Stay Decision

normally. If something not expected occurs (according to the context, risk calculation, and entity involved), a trust estimation is needed to decide whether the entity is behaving maliciously or not. During trust estimation, the context and the risk calculation of the action are all taken into consideration, together with the reputation DB data where the history of the entities is stored and a threat DB updated with the latest known attacks. The smart hub can allow the entity to stay, or it can decide to ban or put it in quarantine. When an entity is put in quarantine, it remains in the network without being able to communicate with the other entities. The entity can only receive communications from the smart hub. The quarantine will continue until new information are available (i.e., known attacks or vulnerabilities related to the entity). In the case an entity is banned or put in quarantine, the smart hub must communicate the decision to the entities having a connection with the banned one. The model for stay decision is similar to the work of Atlam et al. [11]. They have proposed a risk-based access control model for IoT to calculate the risk associated with the access request to a particular resource. We extend this model using risk calculation as a parameter for trust estimation.

3. 7. 2. 3 Leave

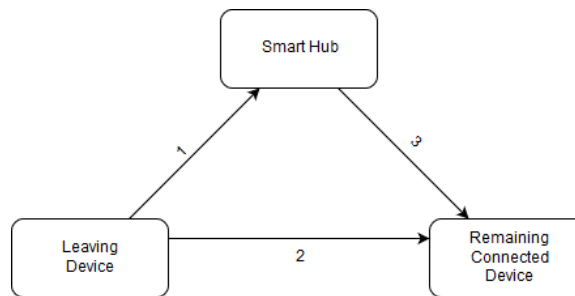


Figure 3.27: Leaving a network

When an entity leaves the internal or external network, it must announce its intention to leave both to the smart hub and to the related entities (actions 1 and 2 in Figure 3.27). Then, in order to enhance security, the smart hub must communicate the change to its related entities. (action 3).

3.8 Conclusion

In this chapter, we have presented the K-Model that compounds our trust-by-design framework together with the transversal activities. Firstly, we have discussed the need phase, which is the starting point of the framework, where all the stakeholders having an interest in the new IoT entity under development illustrate their ideas. Then, the requirements elicitation is performed in the second phase through the TrUStAPIS methodology. Thirdly, we have proposed a trust model-driven approach proposing UML and SysML diagrams enriched by trust and related domains for the model phase. Then, the development phase has been proposed considering several approaches to analyze the functionalities and the contexts of the IoT entity along with a trusted finite-state development. After this phase, verification and validation methods have been presented in order to guarantee that the IoT entity has been developed as intended. Finally, the utilization phase covers the aspects related to the interaction of the developed IoT entity with other entities.

CHAPTER 4

Transversal Activities of the K-Model

In this chapter, we present the transversal activities, which, together with the K-Model, compound our trust-by-design framework. These activities are seven: traceability, documentation, metrics, gates, threat analysis, risk management, and decision-making. For each of them, we will describe its importance for the framework and where they must be considered along the K-Model. In fact, there are activities more important for a particular phase than others. However, some activities (such as gates or traceability) are considered in all the phases.

4.1 General Overview

As we stated earlier, the K-Model is only a part of our trust-by-design framework. In fact, along with its seven phases, we implement several transversal activities fundamental for its correct utilization.

The first activity is *traceability*. It is considered among every phase as shown in Figure 3.1 (i.e., represented by the arrows) and in some phases also within them (i.e., Requirements, Model).

The second activity is *documentation*. This task is important in many phases, especially in the need, requirements, and development phases. In fact, during these three phases, the developers must redact documentation that is fundamental for the following phases of the SDLC. During the need phase, the documentation helps describe the idea and define *what* the entity shall do and *why* the entity is needed. Then, these documents are the starting point for the following phase to elicit the proper requirements. During the development phase, the documentation is important to explain the code and the implemented methodologies to be later verified and validated.

Thirdly, *metrics* are useful in several phases of the SDLC. In the requirement phase, they are defined in order to specify the requirements. During verification and validation, metrics are used to check if the entity has been developed correctly. Then, in the utilization phase, they are fundamental in order to take decisions. For example, trust metrics will be used to measure the trust level of the other IoT devices in order to perform trust decisions and decide which entity to trust.

The *gates* are important activities related to the organization of the work between two following phases. The gates have basically a double function. In the first instance, they define a backup where it is possible to come back if some problems occur in the following phases. Moreover, it is used to check if every task has been finished in the previous phase in order to proceed to the following one.

The fifth activity is related to the *threat* analysis. Indeed, a difficult challenge in the IoT is to protect the smart entities and customers from different known and unknown attacks. Hu et al. [71] focused on IoT environments. They stated that

it is very important that any IoT architecture considers these attacks proposing a solution.

Risk management is another critical activity that must be considered during the SDLC. Every technology can raise the risk for IoT entities and users. We analyze three parameters: likelihood, severity, and detectability of the risk events.

Finally, decision-making is an activity that helps developers in deciding among different alternatives, and it helps the IoT entity in order to perform trust decisions (i.e., choose the most trusted candidate among different ones).

4.2 Traceability

Traceability enhances the connection among phases through the whole framework. For this reason, we have deeply discussed traceability along with this thesis, especially in Section 3.3.8 and 3.4.8. However, in this section, we summarize the importance of traceability for all the phases of the K-Model. Thus, traceability connects needs to the other phase in order to acknowledge why a decision has been taken. Moreover, traceability permits safely modifying requirements or specifications, avoiding unintended consequences or domino effects due to the deletion of requirements connected to others. In the model phase, traceability is essential to map all the diagrams connections. Then, during the development phase, traceability guarantees that the IoT entity follows the specifications written in the previous phases. For the following phases, it is crucial to implement traceability in order to verify and validate the IoT entity according to needs, requirements, and models developed. Finally, in the last phase of the K-Model, it is essential to guarantee that the developed IoT entity reflects its original needs.

4.3 Documentation

Documentation is fundamental in order to provide developers and stakeholders with crucial information during the whole SDLC. Moreover, the documentation must be

redacted by all the actors involved in each phase of the K-Model. Thus, we can say that a document should contain important information for each phase.

In state of the art, documentation is widely used in many fields. Especially in agile software development, where there are basically two primary reasons in order to perform the documentation tasks: to communicate and to understand ¹.

In [28], the author presented the idea that a document-driven methodology is necessary for software development to create a guide on how to proceed and summarize what the developer has done. For the author, documentation is “not an all-encompassing ‘other’ category, but a natural deliverable of each stage in the process. In supporting the thinking processes and providing checkpoints for review and reference, the value of the documentation is meant to become self-evident”.

In [148], the authors evidenced that in software development, there is also the possibility that documentation can be considered as a “side-task” and usually, the output is only provided at the end of the project. The motivation is due to the fact that software projects are strictly dependent on budget and time. Moreover, the members of a team are usually needed for new projects. These problems lead to having less time to document all the needed knowledge about software or project. This is a common problem, but it is demonstrated that the more time is spent at the start of a project to document it deeply, the less money will be spent later in order to solve or avoid unexpected issues [52].

Thus, we can understand that the documentation can be considered either as an input or an output of the phases. This activity is even more critical for the SDLC, where there are different phases, and each of them needs inputs and provides outputs for the following phases. Nevertheless, our K-Model is not an exception and considering the phases presented in this thesis, the documentation process shall be performed in all of them. In the first phase, it is used as an input for the need identification, and it is provided as an output containing all the collected needs and their relationships with the stakeholders. Then, for the second phase, the input will be the documents generated during the previous phase, and they will be fundamentals in order to elicit the proper requirements. At the end of the second phase, the produced documents

¹<http://www.agilemodeling.com/essays/agileDocumentation.htm>

will contain information about the elicited requirements. During the third phase of the K-Model, the requirements documentation will be a fundamental input. Moreover, during this phase, documents related to the models will be produced, and the developers will need them during the following phase related to the development of the final IoT entity. Then, during the verification and validation of the product, the documentation related to all the previous phases will be considered. In fact, in order to verify the requirements, documentation related to the second phase will be needed. Moreover, the modeling verification and the development documentation will be important to verify the functionalities of the IoT entity. Then, the validation will be managed considering the need documentation. In fact, the final product will be validated only if it fulfils the originating needs. Finally, the validation documentation will be produced for the final phase, originating the guide for the final user. It will define how the product works. Moreover, in this last phase, the need for documentation will be considered by the developers to find issues related to the product and understand if the original needs are met.

To conclude, it is important to highlight that documentation enables traceability among phases in order to keep track of the decisions and why they have been made. Moreover, the documents can be considered a guide for the next phases and as feedback for the previous phases. This feedback allows developers to go back to a previous phase in the case it is necessary to solve any problem that might have arisen.

4.4 Metrics

In our framework, metrics are essential to measure the performance and efficiency of the IoT entity. Moreover, metrics are fundamental during the interactions among IoT entities in order to define the rules of the interactions themselves. For example, in a trust interaction, we can state that trust metrics are fundamentals to decide whether to trust or not another IoT entity. Furthermore, trust metrics are considered in every phase of the SDLC in order to guarantee that the actions are performed in a trusted way. For this purpose, the characteristics of trust explained in Section 3.3 can be used to compute the proper metric in order to help the entities in deciding how to

proceed in a particular action or with which entity to cooperate. The most crucial phase where they will be used is the last one, where the IoT entities must decide how to behave on the basis of the defined metrics.

Essentially, metrics can be qualitative or quantitative. Hermann [69] made the following distinctions. The simplest ones are the qualitative metrics, and they can be divided into nominal and ordinal. A nominal scale represents all the categories we want to measure. They are mutually exclusive and useful for sorting items. On the other hand, ordinal scales are useful for organizing items in sequences. We can use them to identify relationships in order to understand which items are lesser or greater. However, these types of metrics have no numeric values. For example, on the one hand, we can use nominal scales to aggregate trusted nodes in a network dividing them into untrusted, unknown, and trusted. On the other hand, we can use ordinal scales to analyze the relationships of the nodes (i.e., we can have more trusted than untrusted nodes).

Considering quantitative metrics, they can be divided into interval and ratio scales. The interval scales are basically nominal scales with an exact value. According to the previous example, using interval scales, we can enumerate the network nodes (i.e., 57 trusted, 24 untrusted, and 12 unknown nodes). On the other side, a ratio scale is like an interval, but in addition, it has a clear definition of zero value. Thus, we can give positive and negative values to the previous nominal scales using ratio values. For example, 0 can represent unknown nodes, +1 the trusted nodes, and -1 the distrusted nodes.

NIST [26] defined a guideline for security measures. They are useful for decision-making processes. Moreover, it can be implemented to measure performance and enable accountability. About the performance, it is used to improve functionality or apply corrective actions.

In our framework, the measures depend on which phase of the SDLC they are applied. For example, effectiveness/efficiency measures are useful during the validation phase in order to check that the IoT entity and its functionalities have been implemented as intended. Moreover, these types of measures assess two fundamental aspects: effectiveness ensures the robustness of the result, and efficiency reflects the

time and resources spent in order to achieve the result.

Table 4.1 shows the different measures according to the SDLC phases.

Table 4.1: Metrics table

SDLC Phase	Relevant Metrics	Purpose
Needs	Nominal, Ordinal	To define qualitative metrics for the IoT entity
Requirements	Trust, Security, Interval, Ratio	To write requirements that can be measurable and verifiable
Model	Trust, Security, Interval, Ratio	To design models according to the written requirements
Development	Trust, Security, Interval, Ratio	To develop the IoT entity following needs, requirements and models
Verification	Interval, Ratio	To verify that requirements and models are reflected in the developed functionalities
Validation	Efficiency/Effectiveness	To check that the IoT entity and its functionalities have been implemented as intended
Utilization	All of the above	The IoT entity must decide how to behave in different contexts and situations according to the metrics defined during the whole SDLC

4.5 Gates

In our K-Model, we have six gates. They are placed in the middle of the transition between the phases (i.e., between requirements and model phases) and determine if it is possible to proceed to the following phase or not.

Basically, they can be considered as decision points, and it is not possible to proceed to the next phase if the previous phase is not entirely fulfilled. Anyhow, during this activity, all the documents belonging to the previous phase are collected, creating a backup. This aspect is fundamental if something goes wrong in the process, and it is necessary to come back to the previous phase (i.e., a need cannot be satisfied or a conflict among requirements arises during a later phase). For example, this can

happen if the process has passed the gates prematurely (i.e., needs not precisely defined).

The gates are considered both in system engineering [52] and project management². Usually, in order to proceed to the following phase, during gate activity, reviews are needed. These reviews are called gate reviews (GR). As we have stated earlier, GR are fundamentals in order to let the process move through the different phases.

Moreover, GR are useful to mitigate the risks throughout the whole project, and they are useful to monitor changes in the goal or the scope of the product/project. In this case, it is important to backtrack the process in order to change the originating needs. GR must be performed by developers and stakeholders, and when they all agree, it is possible to proceed to the following phase. Through GR, it is also possible to carefully check the progress of the IoT entity under development according to the previous gates. During GR, it is also possible to decide if the development should be delayed, modified, or deleted.

In the case of a successful GR, it means that the IoT entity development is going as planned.

Because GR occurs when there is a transition to the next phase, it is not periodic. However, it should be planned according to the project's schedule, but it can be postponed (i.e., in the case unexpected issues arise).

Moreover, a GR should accomplish the following sub-activities:

- Assessment of the feasibility of the project in order to proceed to the following phase.
- A re-planning of the purpose of the project (if necessary).
- A recapitulation of project history.
- Creating a backup of the work carried out so far (if it is possible to proceed to the next phase).
- A re-commitment of the available resources (if necessary).

²<https://www.pmi.org/learning/library/contemporary-gate-philosophy-implemented-outcome-7786>

- A project review.

The success of this transversal activity depends on a predefined GR process. As we stated earlier, there are six gates in the SDLC related to the K-Model, and all of them are defined according to their specific place in the project flow. Thus, we can state that each gate is unique, but there can be commonalities among them. For example, common elements can be the GR participants (i.e., stakeholders and developers), meeting duration, or decision-making rules (i.e., majority, veto, unanimity). On the other hand, unique elements will be the ones related to their position in the SDLC.

Nevertheless, each gate results in three possible outcomes:

1. It is possible to proceed to the next phase.
2. The phase is not correctly concluded, so the process must go back to the previous phase. Another GR will be performed where the arisen issues will be solved. In this case, it is also possible to go back further in the project flow.
3. The project is cancelled. This may occur if huge risks have arisen or for business reasons.

In Table 4.2, we can see the differences among the gates related to our framework. For each of them, the activities are strongly related to the previous phase plus other phases considered through traceability (i.e., in gate five are considered both the verification phase than the requirements and model phases).

On the other hand, in the first gate review, only the need documentation and the stakeholders are considered because the IoT project and product flows are at their start. Thus, we can state that the more advanced the flow is, the more elements we need to consider during the GR.

4.6 Threat Analysis

During the SDLC of a new IoT entity, it is essential to consider the possible threats that can affect the IoT device, its environment, and its architecture. For this reason,

Table 4.2: Gates table

Gate number (position)	Gate Topics
Gate 1 (From Needs to Requirements)	Collected Needs, Stakeholders
Gate 2 (From Requirements to Model)	Elicited Requirements, Traceability to Needs
Gate 3 (From Model to Development)	Models, Traceability to Requirements
Gate 4 (From Development to Verification)	Functionalities verification process, requirements verification document
Gate 5 (From Verification to Validation)	Entity verification, requirements and model documentations
Gate 6 (From Validation to Utilization)	Entity validation completed and connected to the originating needs

in order to protect the IoT entity under development and enhance trust, it is important to be aware of the potential threats (i.e., known attacks). Thus, in Table 4.3, we can see the threat consideration performed in each phase.

Anyhow, because it is challenging to be protected against zero-day attacks (i.e., attacks not known), the *modus operandi* is to collect all the information about known attacks and guarantee that the new IoT entity will be protected against them.

This part is essential to be investigated, especially during the development phase. However, since the requirements phase, it is advantageous to elicit the proper requirements in order to specify what is needed to enable the protection. Then, during the model phase, it is essential to design the models according to the possibility of being attacked. Then, as we stated earlier, the development phase is crucial because, following the previous phases and the known attacks, the developers can produce the proper code. Moreover, during the verification and validation process, these attacks can be performed in order to find a vulnerability. If this occurs, the IoT entity must be further protected going back to the previous phases, but planning a defense since the earliest phases of the K-Model helps minimize or avoid this possibility.

Table 4.3: Threats table

Phases	Threats Analysis
Needs	No Countermeasures here because it is most important to decide what it is needed
Requirements	According to the needs, design the requirements considering specific threats
Model	Model the defense against threats following the elicited requirements
Development	Design the IoT entity according to requirements and models
Verification	Verify that the IoT entity resist to known attacks
Validation	Validate the IoT entity that is protected and reflects the originating needs
Utilization	Support and modification in the case a zero-day attack has been performed

4.7 Risk Management

In the state of the art, there are many techniques related to risk estimation [17], and most of them are related to trust. Yan [160] stated that risk is correlated to trust. In fact, “trust involves uncertainty and risk. There is no perfect guarantee to ensure that the trustee will live up to the trustor’s expectation”. Marsh [98] declared that in all the cases where trust is involved, the trustor risk something in trusting the trustee. Ruohomaa [138] stated that “A trust decision is binary and based on the balance between trust and risk, and it has some sort of effect on the trustee. Usually, it is made with a class of applicable situations in mind, such as concerning a particular trustee in performing a certain action only.” Furthermore, Cvrcek [34] stated an essential semantic distinction between trust and risk: “trust is established with respect to entities/principals (a user, a process, implementation of a service), while the risk is a property of particular processes (i.e., service invocations, protocol instances). Both are subject to context.”

In our framework, risk management is strongly connected to threat analysis. In fact, a risk for the IoT entity can be related to an attack. Nevertheless, risk can also

be provoked by possible internal malfunctions or lousy implementation. However, we can minimize this risk considering it since the early phases of the SDLC.

During the need phase, the risk is connected to the possibility that different needs belonging to diverse stakeholders can raise conflicts among them. If a solution is not found, there can be the possibility that the project will not start. Another risk is that the outcome of the IoT entity will not repay the investors, and it will become a loss. In the requirements phase, risks can be related to conflict among requirements or to the time spent in this phase. In the model and development phase, risks can be related to the difficulty of representing a requirement or a need. Verification and Validation risks are related to the possibility that it is not possible to verify or validate a functionality of the IoT entity in a proper way. This situation can lead to two different outputs: returning to previous phases or moving forward aware of the risk. A risk analysis helps in this decision. During the utilization phase, risk can be caused by a malfunction (related to the not verifiability of a function) or a zero-day attack performed against the IoT entity.

We perform risk management considering three parameters: likelihood, severity, and detectability. Firstly, likelihood (L) represents the probability that a dangerous situation will occur. It can be related to a threat or a defect. Secondly, severity (S) is connected to the impact that the defect or attack represented by L can have on the system. The more critical it is, the higher value is set. Usually, in many risk analysis techniques [17], these two parameters are enough in order to calculate the overall risk. However, in our opinion, we need to consider a third parameter: detectability (D). It describes the possibility of detecting the hazard represented by S and L. In fact, for example, if an attack is occurring and we cannot detect it, the system will fail or will be manipulated without letting the user be aware of it.

We compute these three parameters together with a multiplication. This is a common approach performed in many risk methodologies [17].

Thus we will have:

$$Risk = L * S * D$$

The three-dimensional risk graph is shown in Figure 4.1. The lowest risk value

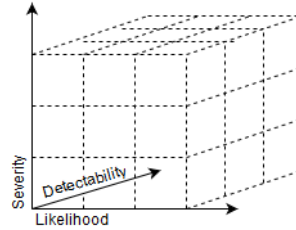


Figure 4.1: Requirements Relationships

is presented in the first box. The maximum risk value is the box where L, S, and D are the highest. In the middle, there are all the other possibilities. However, we can define that there is a high risk when two of L, S, or D are the highest and one is the lowest or medium, or where one is the highest and the other two are the medium. Then, we have a medium risk in four cases: firstly, when either one of L, S, or D is the highest and the other two are the lowest; secondly, if one is the highest, another one is the lowest, and the last one is the medium; thirdly, when L, S, and D are medium; finally, if two parameters are medium and one is low. Finally, we have a low risk when one parameter only is medium and the remaining two are low.

The values which we take into consideration for L, S and D are:

$$Low = 1; Medium = 3; High = 9$$

To keep the computation simple, we have considered only three values for each parameter. Thus, according to Figure 4.1, if the result is lower than 9, the risk is considered as low. Then, if the result is between 9 and 27, we have a medium risk. Finally, if the value is higher than 27, we have a high risk.

Summarizing, the overall risk values have been chosen according to the following criteria:

1. It is the same level of all the parameters if they belong to the same level (i.e., low if all of L, S, and D are low).

2. Low, if there is a medium parameter only and the other two parameters are low.
3. High, if there are two or more parameters set to high or two parameters set to medium and one set to high.
4. Medium, in all the other cases.

In Tables 4.4, 4.5 and 4.6, we can see that the risk values have different definitions according to their category. They are expressed directly in the table according to their value and meaning.

Table 4.4: Likelihood

Value	Meaning
Low (1)	The event is unlikely to happen
Medium (3)	The event can quite probably happen
High (9)	The event is almost certain to happen

Table 4.5: Severity

Value	Meaning
Low (1)	The network is not damaged
Medium (3)	The network can be partially damaged
High (9)	The network can become completely useless

Table 4.6: Detectability

Value	Meaning
Low (1)	The problem is easily detectable
Medium (3)	The problem cannot be entirely detected
High (9)	It is not possible to detect the problem

Table 4.7: Values of Risks

Risk level	L * S * D	Total
Maximum	9 * 9 * 9	729
High	9 * 9 * 3	243
	9 * 3 * 9	243
	3 * 9 * 9	243
	9 * 9 * 1	81
	1 * 9 * 9	81
	9 * 1 * 9	81
	3 * 3 * 9	81
	9 * 3 * 3	81
	3 * 9 * 3	81
Medium	9 * 3 * 1	27
	1 * 9 * 3	27
	3 * 1 * 9	27
	9 * 1 * 3	27
	1 * 3 * 9	27
	3 * 9 * 1	27
	3 * 3 * 3	27
	9 * 1 * 1	9
	1 * 9 * 1	9
	1 * 1 * 9	9
	3 * 3 * 1	9
	1 * 3 * 3	9
	3 * 1 * 3	9
Low	3 * 1 * 1	3
	1 * 3 * 1	3
	1 * 1 * 3	3
Minimum	1 * 1 * 1	1

Then, as we can see in Table 4.7, we have different levels of risk. The maximum level value is 729, and it occurs only when there is the highest value for each of the variables. Moreover, from 81 to 243, we also have a high risk. The medium risk is between values 9 and 27. The low risk is when we have 3 as value. Finally, the minimum risk is when all the variables are at their lowest value.

4.8 Decision-Making

The decision-making process can be performed in each phase of the framework.

In the need phase, decision-making can help to choose among different needs in the case they create conflicts among them. The same situation can be stated for the requirements phase, where it is possible to find conflicts among the elicited requirements. In order to proceed to the following phase, it is mandatory to solve these issues. Then, in the model phase, the Decision Maker (DM) can help in choosing different diagrams to model a particular functionality. Another type of DM, related to the decision that the IoT entity must take in the final phase is modeled in the development phase according to the elicited requirements. Finally, during the utilization phase, decision-making is related to the interactions among entities. For example, it is needed to decide which entity can be trusted in order to perform a particular action.

However, trust has a crucial role in order to perform decisions in every phase of our K-Model. For this reason, we have developed a Pairwise Ordination Method (POM) to assist developers in the decision-making activity. We will apply it in the second phase of the K-Model (i.e., requirements phase), but it can also be applied to the others whenever it is mandatory to choose among different alternatives. Anyhow, this method is based on the Analytic Hierarchy Process (AHP). Thus, we present it in order to introduce POM properly.

4.8.1 Analytic Hierarchy Process (AHP)

AHP has been developed by Saaty [140], and it is a structured technique for organising and analysing complex decisions. It is one of the most widely used methodologies in Multi Criteria Decision Analysis (MCDA) [151].

Since its creation, AHP has been commonly applied in complex decision-making problems thanks to its ability to synthesise both tangible and intangible characteristics to accommodate both shared and individual values [40].

In state of the art, only a few works consider AHP for trust management. One

of them has been developed by Pang et al. [122]. In this work, the authors have proposed AHP as a solution for the limited computational power of IoT nodes. They have developed a model to compute the trust level of the IoT entities, also considering reputation parameters. However, they do not consider related properties as privacy or security.

Concerning cloud security, Taha et al. [151] have introduced an AHP-based technique allowing a comparative analysis of it. They consider a methodology helping users to understand better and identify their security needs. However, in our framework, we have considered other types of requirements in addition to security (i.e., availability, privacy) that in their work are missing. Furthermore, Kassab et al. [81] explore AHP in order to assist in the prioritization of quality requirements. This approach can be limited due to the increase of complexity if it is used for all the requirements. In our approach, even if we focus only on the conflictual requirements, we avoid the issues related to the consideration of a considerable number of alternatives.

Another work has been proposed by Kim [84]. This is an AHP method based on network interfaces and a channel selection algorithm for multichannel MAC protocols in IoT ecosystems that considers several decision-making factors such as expected channel condition, latency, and frame reception ratio. The proposed scheme considers an IoT-based healthcare system. Finally, due to the fact that AHP can be utilized in numerous fields, it has been applied in individual and group decision-making processes [7].

As we stated earlier, a remarkable peculiarity of this methodology is that it allows developers to compare both tangible and intangible characteristics belonging to different fields, prioritising and normalising them. This is a critical aspect because, due to the multidisciplinary and heterogeneity of trust and IoT, it is advantageous to be able to compare alternatives from different areas in order to prioritize them. These alternatives have to be compared with respect to determined criteria. In the case a higher level of detail is needed, a criterion can be divided into two or more sub-criteria. We can see all these elements (i.e., alternatives, criteria, and goal) in the general AHP model shown in Figure 4.2.

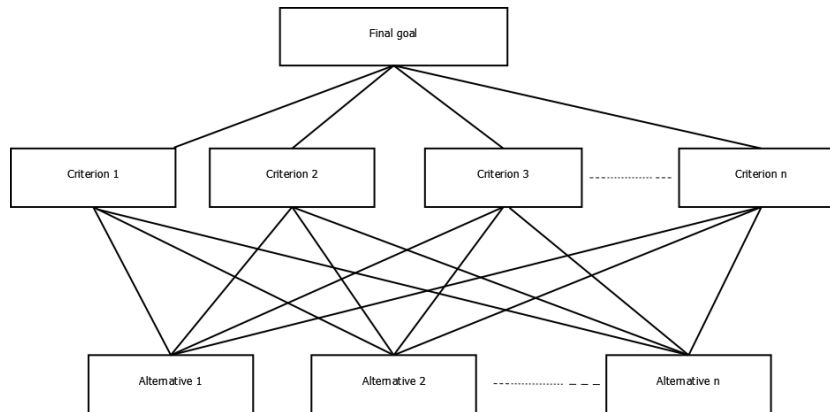


Figure 4.2: General AHP model

The AHP works in the following way. The nodes belonging to a lower layer are pairwise compared with respect to their contribution to the nodes above them. The results fill a matrix, which is then mathematically processed to derive the priorities. During the pairwise comparisons, it is decided if the first term (A) is more important than the second one (B) or vice-versa. Saaty has proposed a fundamental scale to compare the elements shown in Table 4.8.

Table 4.8: The fundamental scale for pairwise comparisons [140]

Intensity	Meaning
1	A and B are equally important
3	A is relatively more important than B
5	A is more important than B
7	A is much more important than B
9	A is absolutely more important than B

We can state that the more important A is for B, reciprocally, the less important B will be for A. Thus, for example, if we compare A and B in accordance with a criterion C and we think that “A is more important than B”, we will give A the value of 5 with respect to B, so we have to reciprocally give to B the value of $1/5$ with respect to A.

The order of comparisons usually begins with comparing the criteria with respect to the goal. Then, if they exist, there is a comparison of the sub-criteria according to the originating criterion. Finally, the alternatives are compared with respect to

each (sub)criterion. All these values will fill the proper matrix.

For every matrix, a Consistency Index (CI) is calculated. This value is needed in order to check if the performed comparisons are consistent or not. If CI is lower than 0.10 [140], the matrix is consistent. If not, the choices must be reconsidered in order to solve the issue. The more elements have to be compared, the higher is the probability of facing inconsistency. For this reason, AHP utilization is discouraged for more than ten alternatives or criteria [140]. This is an issue that in an environment such as the IoT can limit the effectiveness of this approach. We mitigate this issue with our methodology, as we show in the next section.

4.8.2 Methodology: Pairwise Ordination Method (POM)

Our approach is based on AHP and similarly to it, POM is composed of a goal, alternatives, and criteria, but the operations among them are performed differently. In our case, the goal is to rank the requirements that guarantee the maximum level of trust for the developing IoT entity according to the criteria and the conflicting alternatives.

Similarly to AHP, the criteria can be divided into sub-criteria to improve the level of detail.

Finally, there are the alternatives, which in our case are the conflicting requirements.

In this approach, trust is considered as a way to improve the quality of the system in choosing which requirement to keep and which one to release or modify.

A peculiarity of our methodology is that it is possible to compare aspects related to different fields, prioritising them with a normalised value. This aspect is critical because, in accordance with the multidisciplinary aspect of trust and IoT, it is useful to be able to compare aspects belonging to different areas.

The goal we have to achieve is to choose the requirement maximising the trust level of the developed system.

The alternatives we have to take into consideration in this method are requirements arising conflicts among them. Furthermore, we assume that the elicitation

process and the identification of these conflicts have already been performed. Thus, depending on the alternatives, it is important to define the criteria. Hence, we have identified three groups of criteria that are mandatory to be used in our methodology (considering the requirements phase):

1. **Context criteria.** As we stated earlier, the context can be considered as a composition of functionalities or depending on the environment. Moreover, context is always present and needed to be taken into consideration in an IoT environment. Furthermore, according to the general aspects related to trust, we have to identify *general criteria* that can affect the system's trust level.
2. **Traceability criteria.** Another important activity that we have presented in Section 4.2 is traceability. It is crucial during the development of an IoT entity because we cannot solve a conflict requirements issue without considering which other requirements, needs, models, or documents are connected with them. Moreover, validation and verification aspects must be taken into consideration in this group criteria. In fact, when a requirement is elicited, it should be decided how to verify and validate it to speed up this process. For this reason, if we modify such requirement, we have to modify as well the way we can verify and validate it.
3. **TrUStAPIS criteria.** As we stated earlier, trust is strongly related to other properties: privacy, identity, security, usability, safety, and availability. The domains of these criteria are the same as the conflicting requirements. It is useful to perform this comparison because it is important to rank them accordingly to the other connected requirements in order to see which criterion is more useful compared to all the requirements. It can be only one (in the case all the conflicting requirements belong to the same domain), or there can be seven criteria (all the domains identified in TrUStAPIS).

4.8.3 Procedure

In this methodology, we have to perform comparisons among the elements belonging to contiguous layers. We suggest starting by comparing criteria according to the goal. Then, we compare sub-criteria (if they exist) according to the originating criterion, and finally, we compare the alternatives with respect to the criteria or sub-criteria. These comparisons are fundamental to order alternatives and criteria. Thus, at the end of the procedure, it is possible to rank the requirements from the most to the least important. The importance is given by which one of the requirements mostly improves the level of trust of the system.

During each comparison, an ordered branch-tree is created. For each round, we compare an element with the following not yet ordered element and the DM decides which one is more important following the criterion or the goal. The procedure is iterative, and each time an element *wins* the comparison, it has to be compared with the following element. The round ends after each element has been compared at least once. When the first round finishes, we obtain a partially ordered tree. We will compare the not yet ordered elements in the following rounds of comparisons until we have a wholly ordered branch-tree. Anyhow, this branch may have some levels populated by more elements.

For example, let us assume that we have six alternatives (A, B, C, D, E, and F), and we have to compare them according to criterion X.

In the first round, we have to compare all the elements among them in order to decide which one is the most important. Hence, we firstly compare A with B in order to decide which alternative is more important. Imagine that the DM decides that B is the most important. It means that B will be then compared with C. Let us assume that B wins all the comparisons until F. Thus, we find out that B is the most important alternative among the others according to the criterion X, but we do not know how to rank the other alternatives. Thus, we have to estimate also the order of the other alternatives, and we can start the second round of comparisons.

Firstly, we compare A and C, and we find out that C is the most important. Secondly, C is compared with D, and the DM decides that D is more important than

C. Then, we have to compare D with E and D wins. Finally, we compare D with F, and D is the most important.

Now, we know that B is the most important element, followed by D. But we also have to order the remaining alternatives.

In the third round, we compare A and C again, but we previously found out that C was more important than A, so it is possible to skip this comparison. Then, we compare C and E deciding that C is the most important. Finally, C is compared with F, and C wins again.

Now, our branch-tree is composed of B, C, and D ranked in this order. The remaining alternatives to be ordered are A, E, and F.

Thus, we start the fourth round by comparing A and E, and we find out that they are equally important according to X. Finally, we compare A with F, and the DM decides that A is the most important.

In the last round of comparisons, we find out that E is more important than F because A and E have the same importance.

Finally, the algorithm ends, and we have an ordered branch-tree shown in Figure 4.3.

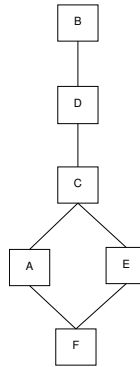


Figure 4.3: Example: ordered branch-tree

Algorithm 1 represents our case.

Now that the branch-tree is ordered, we need to give each element a weight to normalize them. In our approach, the weights are based on the number of elements and on their importance. The maximum weight value is equal to the number of elements, and the minimum value is one. In order to assign the weights, we proceed

Algorithm 1 Trust model algorithm for home devices

```

1: procedure POM
2:   while Each_element_is_not_ordered do
3:     branch["A", "B", "C", "D", "E", "F"]
4:     orderedBranch["", "", "", "", "", ""]
5:     for all  $i = 1$  to branch.size() - 1 do
6:       if  $i = 0$  then
7:         Compare the first two elements of branch array
8:         if One of the two elements is better then
9:           BestElementWins
10:        else They are equals
11:          BothElementWins
12:        else
13:          if One of the two elements is better then
14:            BestElementWins
15:          else
16:            BothElementWin
17:          Save_Winner(s)_into_orderedBranch
18:          Remove_Winner(s)_from_branch
19:          orderedBranch["B", "D", "C", "A", "E", "F"]
20:          branch["", "", "", "", "", ""]

```

following a bottom-up approach. If the bottom layer has only one element, the element belonging to the layer above will have a value equal to the lower value plus one. In the case a layer has more than one element of the same importance, such elements will have the same value. However, in this case, the element above them will have a value equal to their value, plus the number of equal elements. We use this “jump” to highlight the difference between the upper element respect to the lower elements.

We can summarize these cases in the following general formula:

$$Element_weight = Lower_element_weight + Number_of_lower_elements$$

Considering the previous example, we had six elements (A, B, C, D, E, and F). Hence, we have 6 as the maximum value and 1 as the minimum value. Hence, we give F weight of 1 because it is the lowest. Secondly, we have two elements of the same importance (A and E), and we give both of them a weight of 2. Above them,

there is C that weights 2 (the value of the element beneath it) plus 2 (the number of elements of the same value in the lower level). So, the weight of C is 4. Then, we have D that is equal to 5. Finally, we have the most important element that has a weight of 6, which is B. Our weighted and ordered branch-tree is shown in Figure 4.4.

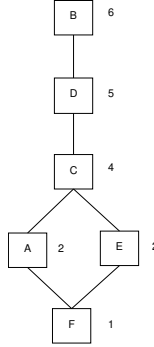


Figure 4.4: Example: ordered and weighted branch-tree

Considering that in a real scenario, there will be other criteria, we have to normalize the values of the alternatives in order to compare them with the results related to the other criteria. This operation will also be performed for the criteria according to the final goal.

Following the previous example, to normalize the elements, we have to divide each of them by the sum of the weights that is 20. Thus, the normalized values for each alternative will be:

$$B = 6/20 \Rightarrow 3/10$$

$$D = 5/20 \Rightarrow 1/4$$

$$C = 4/20 \Rightarrow 1/5$$

$$A = 2/20 \Rightarrow 1/10$$

$$E = 2/20 \Rightarrow 1/10$$

$$F = 1/20$$

The sum of all the normalized weights is 1. In order to reach the final goal and choose which alternative is the most important, each alternative will be multiplied for the normalized weight of the (sub)criteria among them and added to the values of the same alternative compared to the other sub-criteria and criteria. The final sum

of all these values will be 1, and the higher alternative will be the most important. We will show this procedure with a complete example in Section 5.2.6.

4.8.4 POM vs AHP

The complexity of POM is the following: the maximum number of comparisons depends on the number of alternatives, and the minimum number of comparisons is the number of the alternatives minus one, as we show in Table 4.9. The latter is the best case. In fact, imagine that with each comparison, the following element always wins. It means that the previous elements have already been ordered, and we need only one round of comparisons to order the branch-tree.

Table 4.9: Maximum comparison related to the number of alternatives. Legenda: i = number of alternatives, \max = maximum number of comparisons, \min = minimum number of comparisons, n.a. = not available.

i	1	2	3	4	5	6	7	n
max	n.a.	1	3	6	10	15	21	$n(n-1)/2$
min	n.a.	1	2	3	4	5	6	$n-1$

In AHP, we have a fixed number of comparisons that is equal to the maximum number of comparisons of our method:

$$AHP - comparisons = n(n-1)/2$$

A possible problem using AHP is the possibility of having an inconsistent matrix when the number of elements grows. With POM, the inconsistency is avoided because the comparison tree is ordered at every step of the algorithm. Another difference between AHP and our methodology is related to the weights. If in AHP, the weights are shown in Table 4.8, in our approach, the weights depend on the number of elements. Thus, in our methodology, the weights are fixed. However, even if in AHP, the weights can better represent the differences among the elements, this possibility can increase errors due to subjective decisions. Our approach mitigates this error.

4.9 Conclusion

In this chapter, we have presented the seven transversal activities that are part of our framework. Firstly, we have presented traceability that represents the connection between the phases of the K-Model. In addition, it is fundamental in the requirement and the model phases connecting their elements among them. Secondly, documentation is an activity that allows creating records of why the decisions have been made along the whole SDLC. Without this task, it is challenging to have accountability for the decisions made or verify or validate the IoT entity. Thirdly, metrics guarantee that trust decisions must be measurable or to verify a requirement considering a designed parameter. To continue, we have the gates between each phase to allow the flow to continue or not (in the case the previous phase is not completed). Then, threat analysis and risk management are performed in several phases in order to develop the proper countermeasures in the case some attacks could be performed, or design choices can raise some risk. Finally, decision-making processes help developers of the IoT entity to decide in the case different choices can be made considering the most trusted one.

CHAPTER 5

Implementation of the Framework in an IoT Scenario

In this chapter, we will present an IoT scenario in order to show how to implement our framework. In each subsection, we will present the phase and the related transversal activities. Thus, in the first phase, we will present the need related to the IoT entity under development. In the second phase, we will elicit the requirements according to the needs. Thirdly, we will design the models. In the central phase of the K-Model, we will present how to perform the development of the IoT entity. Then, we will show how to verify and validate it. Finally, we will present how the intended entity will be used in the final phase. For each of them, we will present several sub-sections considering the transversal activities that will be important for each phase.

5.1 Need

In this phase, it is important to define which entity is needed and what its development should satisfy. So, let us assume an IoT scenario where a Smart Cake Machine (SCM) is needed to be developed. The stakeholders having an interest in it are vendors and customers only. This IoT entity will provide a list of which ingredients are needed in order to bake cakes. It is important that the temperature will not overcome 250°C. Moreover, the recipes will be downloadable from a website, or users can insert them. In order to interact with the SCM, the users must be trusted and authenticated. In order to recognize users, they must be registered providing their data. The users would like to remain anonymous. On the other hand, these data should be accessed by the vendors for market surveys. Considering IoT interactions, the SCM must cooperate with other IoT entities such as the Smart Fridge (SF) or Smart Supermarkets (SM). Thus, if an ingredient is needed, the SCM can check the SF if it is available. If not, the SCM can send a request to the nearest trusted supermarket to order the missing ingredients.

However, from the previous description, it is possible to elicit and write the needs in the following format (it should also be included in the documentation in order to be considered in the following phases of the K-Model). We elicit the following ones in order to show how they must be described.

1. **Need 1:** The temperature of the SCM must be checked and it could not overcome 250°C.
2. **Need 2:** The recipes must be downloadable from the vendor website or inserted manually by authenticated users. Authentication must be done by code.
3. **Need 3:** The SCM could interact with a Smart Fridge (SF) to check whether a particular ingredient is in the fridge or not. If not, the SCM could interact with a trusted Smart Supermarket (SM) through the home Smart Hub (SH) and order the missing ingredient.

4. **Need 4:** The communication among the smart home entities must be guaranteed and encrypted.

In addition to the needs description it is important to have clear in mind since this phase the architecture in which the IoT entity to be developed will be used. In this case, we propose a smart home centralised architecture with a Smart Hub (SH). These relationships are shown as a possible example in Figure 5.1.

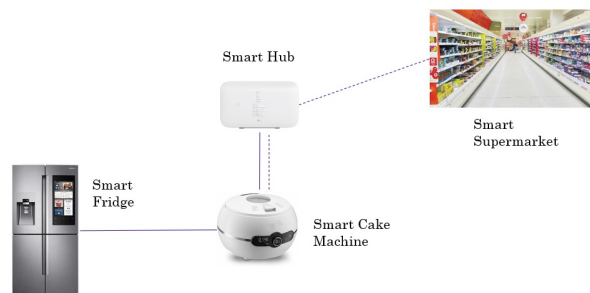


Figure 5.1: Smart Cake Machine and its relationships with the other IoT entities

The dotted lines between the SCM and the SH and between the SH and the SM are related to the fact that there is an indirect connection between the SCM and the SM. In fact, the SCM must delegate the SH to interact with the SM.

In this phase, context is strictly connected to the architecture in which the IoT entity under development will be used. Thus, in order to create a trust environment for the SCM, we need an architecture that provides a segregated trusted environment. This architecture answers to a specific need proposing a new way to solve problems of security, trust, and privacy belonging to classic IoT architectures [144]. This need can be satisfied by protecting IoT entities with a segregated trust architecture. With the locution *segregated trust*, we consider that the IoT entities inside an internal network can trust only the entities which are allowed to interact with. This segregation is guaranteed by the internal architecture, which is designed to prevent external and internal threats.

The architecture that shall be developed is similar to Obregon's work [119]. However, moving further from this work, we propose the model shown in Figure 5.2, which is divided into six levels plus a Demilitarized Zone (DMZ). The levels are divided

into two main zones: the blue zone is related to the internal network, and the green zone is related to the external network.

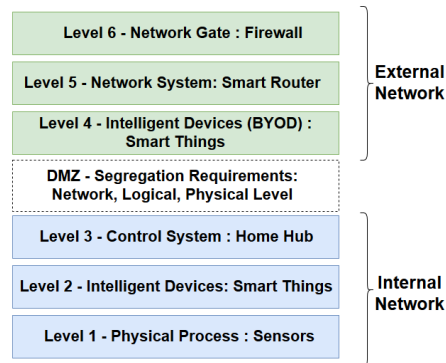


Figure 5.2: Hierarchy Levels of the Segregated Architecture

Starting from the bottom, we have the level concerning physical sensors. They collect raw data from the field, sending them to the level above belonging to the smart entities. These devices have to process and analyse the raw data originated in the level below and act when it is necessary. For example, a smart smoke detector can notice that there is smoke in the smart home, and if the smoke level is higher than a threshold, the sensor triggers an alarm. The third level of the architecture belongs to the control system, where there is a central unit (i.e., a home hub). This smart hub must monitor the other smart objects, and it is the connection point between them the external network and the Internet. The segregation takes place at this level. This level is the highest of the internal network. The home hub is connected to the Internet through a DMZ. This zone prevents the lower level from being compromised by external threats, and a firewall monitors the inbound and outbound traffic. It is possible to use two firewalls in order to create a DMZ, and this is considered the most secure approach [75].

In this case, the first firewall configuration must allow traffic only to the DMZ; then, the second firewall allows the traffic from the DMZ to the internal network. This configuration increases the internal network's protection from external threats, for example, preserving privacy protecting the data inside the protected zone.

Beyond the inner zone, there is the fourth level. In this level, we have all the entities classified under the Bring Your Own Device (BYOD) paradigm [108]. These

devices can be, for example, smartphones or laptops that the owner also uses in other networks. For this reason, they have to be separated from the internal network. Anyway, they can communicate with their smart devices through the DMZ.

Then, we have the fifth level. It is related to the Network System, which exchanges communications with the Internet through the sixth level. At this level, we have the smart router that is able to block dangerous communication or forward the harmless ones to the layer below in the case coming from the Internet or vice versa. Finally, there is the sixth level where we can find the last firewall, protecting the external layer from Internet threats. Both the firewall and the smart router of layer five can be implemented to block or not the communication in both directions. This implementation is strongly dependent on the context and the environment. In our case, the SCM will belong to level number 2, but during its development it is important to consider also the devices belonging to the other levels in order to properly build the connections among them and the SCM.

About the transversal activities, we can state that documentation is its core activity. In fact, all the functionalities and services needed must be collected in a proper way. This task allows developers to keep track of the motivations behind a particular choice. Thus, also traceability has a significant role here in order to preserve these connections. Then, through documentation and traceability, it is possible to connect the following requirements to their originating needs. If a decision-making process is needed, there will be added the information related to this task. It is essential to keep all the documentation related to the needs, even if they will be modified in order to keep track of all the modifications. Moreover, it is crucial since this phase to collect information about possible risks and threats. In our case, it will be created documentation according to the functionalities expected by each stakeholder. About the gates, it is possible to proceed to the following phase (i.e., Requirements) when stakeholders and developers agree that all the needs have been documented and have been connected to the proper stakeholders through traceability.

5.2 Requirements

In this section, we exemplify the methodology presented in Section 3.3.9 continuing the use case presented in the previous section (i.e., the SCM development).

Thus, considering needs and context, we will elicit the requirements in a four-step methodology. As we explained earlier, the first step will be related to the analysis of the need documentation containing the needed functionalities. Thus, in the second step, the JSON template will be modified according to each requirement. For this process, the conceptual model will help to focus on which elements of the JSON code to fill. The third step will then consider the IEEE 830-1993 specification in writing the text of the requirement. Moreover, it will be considered traceability among the elicited requirements. Finally, the fourth step will include the final elicited requirement. In the case a modification is needed, it is possible to have feedback from the last step to the first one.

5.2.1 Step 1

Analyzing the need documentation, it is possible to understand that the customers necessitate an SCM that tells them which ingredients are necessary to bake a cake. The SCM temperature must be checked, and it could not overcome 250°C for safety reasons. The recipes must be downloadable from the vendor website or inserted by authenticated users. The authentication methodology must be performed by user name and password. The SCM must interact with a trusted Smart Fridge (SF) to check whether a particular ingredient is in the fridge or not. If not, the SCM could interact with a trusted Smart Supermarket (SM) through the home Smart Hub (SH) and order the missing ingredient. The communication among the smart home entities must be guaranteed and protected.

Thus, considering these needs and the context, it is possible to elicit the proper requirements following the TrUStAPIS methodology. This phase is of fundamental importance to continue the development of the SCM. According to the K-Model, the output of this phase will be the input for the modeling phase.

5.2.2 Step 2

After analyzing the need documentation produced in the previous step, it is possible to proceed to step 2.

Following the JSON template shown in Figure 3.4, it is now necessary to set up the JSON code related to our use case and requirements.

Thus, by the JSON template and the conceptual model, it is possible to properly fill the JSON code according to the needs identified before. The JSON code presented in Figures from 5.3 to 5.10 contains the parameters related to each requirement and collected from the previous needs.

Firstly, we model the relationship between the SCM and the SM. To assign a trust value, we need to consider some trust characteristics (i.e., direct, measurable) and the type of action (i.e., fulfill). This trust value will be used by the SCM to decide whether to trust or not to trust the SM.

```

1- {
2-   "IoT_requirement_TRST01" : {
3-     "Context" : {
4-       "Domain" : [{
5-         "Trust" : {
6-           "Characteristic" : ["Direct","Indirect","Global","General","Measurable"] }
7-       },
8-       "Environment" : "Smart City",
9-       "Scope" : "Establish trust between SCM and SM" },
10-    "Actor" : {
11-      "Role" : ["Trustor, Trustee"],
12-      "Type" : ["SCM", "SM"] },
13-    "Action" : {
14-      "Type" : ["Fulfill"],
15-      "Measure" : ["Trust level"],
16-      "Goal" : "To fix a trust value"
17-    },
18-    "IoT_requirement_USAB01" : {
19-      "Context" : {
20-        "Domain" : [{
21-          "Usability" : {
22-            "Characteristic" : ["Simplicity, Understandability"] } },
23-        "Environment" : "Smart Home",
24-        "Scope" : "User Interface" },
25-      "Actor" : {
26-        "Role" : "User",
27-        "Type" : ["Human User"] },
28-      "Action" : {
29-        "Type" : ["Fulfill"],
30-        "Measure" : [" " ] },
31-      "Goal" : "Let the user insert new recipes"
32-    }
33-  },
34- }

```

Figure 5.3: JSON code part 1

We can now analyze the JSON code in Figure 5.3 (from line 2 to 16). The specific identifier related to the first requirement is shown in row 2.

row2 – “IoT_requirement_TRST01”

As we mentioned earlier, the trust requirement has some characteristics (row 6) needed for the computation of the trust value. These characteristics must highlight particular aspects that we want to represent with the elicited requirements.

row6 – “Characteristic” : [“Direct”, “...”, “Measurable”]

Here, the characteristics that we consider are: direct, indirect, global, general, and measurable. *Direct*, *indirect* and *global* are defined for the computation of the trust level. For the SCM, a subjective value depends on past interactions with the specified SM (i.e., direct trust). However, it is needed an objective value too. We can consider two objective values: an indirect value and a global value. On the one hand, the indirect value is computed from known and trusted entities. On the other hand, the global value is computed by a centralised authority considering the trusted entities of the overall system. In this scenario, an indirect value is the one computed by the SF (in the case it has a past relationship with the SM), and the global value could be computed by the vendor website (collecting the data of all the SCM interacting with an SM). *General* depends on the fact that it is possible to have a single trust value for each entity. In this scenario, the SM can be considered with a single computed value about its trust level even if different parameters or metrics can be considered (i.e., time, distance, price, quality). In this phase, we do not discuss how these values will be computed, but it is important to underline that in order to compute a trust value, trust metrics are fundamentals. For this reason, the last trust characteristic taken into consideration is *measurable*.

Then, in row 10, we can see the roles of the actors. In this case, they are both trustor and trustee.

row10 – “Role” : [“Trustor”, “Trustee”]

In fact, in our scenario, the SCM can be considered as the trustor because it is the entity ordering the ingredients, and the trustee is the SM, id est the entity providing the missing ingredients.

With this elicited requirement, we want to fix a trust value (goal: row 15) between the SCM and an SM (row 11). The action is considered a goalAction because the type is *fulfill* (row 13), and it has a measure (row 14) computed from the characteristics defined earlier.

row11 – “Type” : [“SCM”, “SM”]

$row13 - \text{“Type”} : [\text{“Fulfil”}]$
 $row14 - \text{“Measure”} : [\text{“Trust level”}]$
 $row15 - \text{“Goal”} : \text{To fix a trust value}$

The second requirement that we model from the needs is a usability requirement.

$row17 - \text{“IoT_requirement_USAB01”}$

It is represented in Figure 5.3 from lines 17 to 31.

This usability requirement is related to the need 2 identified in the previous section and explained by the sentence “The recipes must be *downloadable* from the vendor website or inserted *manually* by authenticated users.” Thus, because a requirement must be complete, we must create (at least) two different requirements, one for the “downloadable” part, the other one for the “manual” part. In this case, we focus only on the second part, so the requirement USAB01 will be elicited according to this need. Then, we will create another requirement, USAB02, in order to cover also the download part. For a user, the procedure of inserting new recipes must be simple, and the user interface must be understandable. For this case, we do not decide how the recipes must be inserted (i.e., by a smartphone, a website or the user interface of the SCM). We only model that the user shall be able to insert new recipes.

Another important aspect is that in order to specialise requirements, sub-requirements are needed.

$row21 - \text{“Characteristic”} : [\text{“Simplicity”, “Understandability”}]$
 $row30 - \text{“Goal”} : [\text{“Let the user insert new recipes”}]$

In Figure 5.4, we collect the second Usability requirement, and we can see that it is similar to USAB01, but there are differences related to the actors (rows 40 and 41) and the goal (row 45).

$row40 - \text{“Role”} : [\text{“User”, “IoT Entity”}]$
 $row41 - \text{“Type”} : [\text{“Human User”, “SCM”}]$
 $row45 - \text{“Goal”} : [\text{“Let the user download new recipes”}]$

```

32 ▾ "IoT_requirement_USAB02" : {
33 ▾   "Context" : {
34 ▾     "Domain" : [{
35 ▾       "Usability" : {
36 ▾         "Characteristic" : ["Accessibility"] } }],
37     "Environment" : "Smart Home",
38     "Scope" : "User Interface" } ,
39 ▾   "Actor" : {
40 ▾     "Role" : ["User", "IoT Entity"],
41 ▾     "Type" : ["Human User", "SCM"] },
42 ▾   "Action" : {
43 ▾     "Type" : ["Fulfill"] ,
44 ▾     "Measure" : [ " " ]},
45   "Goal" : "Let the user download new recipes"
46 ▾ },

```

Figure 5.4: JSON code part 2

In this requirement, we consider both the User and the SCM because the action that will be implemented also requires an active part from the SCM (i.e., connect to the vendors' server to download the recipes). On the other hand, with the requirement USAB01, we focus only on the user perspective.

```

47 ▾ "IoT_requirement_SEC01" : {
48 ▾   "Context" : {
49 ▾     "Domain" : [{
50 ▾       "Security" : {
51 ▾         "Characteristic" : ["Authentication"] } }],
52     "Environment" : "Smart Home",
53     "Scope" : "User Authentication" } ,
54 ▾   "Actor" : {
55 ▾     "Role" : ["User", "IoT Entity"],
56 ▾     "Type" : ["Human User", "SCM"] },
57 ▾   "Action" : {
58 ▾     "Type" : ["Fulfill"] ,
59 ▾     "Measure" : [ " " ]},
60   "Goal" : "To authenticate the user"
61 ▾ },
62 ▾ "IoT_requirement_SEC02" : {
63 ▾   "Context" : {
64 ▾     "Domain" : [{
65 ▾       "Security" : {
66 ▾         "Characteristic" : ["Delegation"] } }],
67     "Environment" : "Smart Home",
68     "Scope" : "Rights and delegation" } ,
69 ▾   "Actor" : {
70 ▾     "Role" : ["Delegator", "Delegatee"],
71 ▾     "Type" : ["SCM", "SH"] },
72 ▾   "Action" : {
73 ▾     "Type" : ["Request", "Fulfill"] ,
74 ▾     "Measure" : [ " " ]},
75   "Goal" : "To delegate another IoT entity because
76 ▾   of rights limitations"

```

Figure 5.5: JSON code part 3

Then, we can see in Figure 5.5 two security requirements. One is related to the authentication characteristic (row 51), and the other is related to delegation (row 66). According to the identified needs, we know that in order to interact with the

SCM, a user must be trusted and authenticated. The second requirement is related to the delegation. In fact, the needs documentation specifies that the SCM must communicate with the SM only through the SH, so in this case, the SCM delegates the SH to order the ingredients.

row51 – “Characteristic” : [“Authentication”]

row66 – “Characteristic” : [“Delegation”]

Considering the actors, in SEC01, they are the human user and the SCM (row 56). Whereas, in SEC02, the actors are the SCM and the Smart Hub (row 71).

row56 – “Type” : [“Human User”, “SCM”]

row71 – “Type” : [“SCM”, “Smart Hub”]

```

77  "IoT_requirement_AVBT01" : {
78    "Context" : {
79      "Domain" : [{
80        "Availability" : {
81          "Characteristic" : ["Resilience"] } }],
82      "Environment" : "Smart Home",
83      "Scope" : "Connectivity between entities" } ,
84    "Actor" : {
85      "Role" : ["Connector, Connectee"],
86      "Type" : ["SCM", "SH"] },
87    "Action" : {
88      "Type" : ["Fulfill"] ,
89      "Measure" : [" " ]},
90    "Goal" : "To provide connection between the IoT
          entities"
91  },
92  "IoT_requirement_AVBT02" : {
93    "Context" : {
94      "Domain" : [{
95        "Availability" : {
96          "Characteristic" : ["Integrity"] } }],
97      "Environment" : "Smart Home",
98      "Scope" : "User Data" } ,
99    "Actor" : {
100     "Role" : ["Vendor"],
101     "Type" : ["Human User"] },
102    "Action" : {
103     "Type" : ["Fulfill"] ,
104     "Measure" : [" " ]},
105    "Goal" : "Availability of User Data for Vendors"
106  },

```

Figure 5.6: JSON code part 4

In Figure 5.6, we can see the JSON code for the availability requirements AVBT01 and AVBT02. Considering the needs, we can state that for the first availability requirement, the characteristic that we take into consideration is *Resilience* (row 81).

In fact, in order to assure the availability of the communication between the entities, resilience guarantees to be available also in the case of malfunctions or attacks. In this case, we require only that the communication is available. How resilience is guaranteed will be implemented in the following phases of the SDLC or by other sub-requirements. The “scope” is related to the connectivity between IoT entities (row 83) and the “goal” is about to provide connection between the IoT entities (row 90).

row81 – **“Characteristic” : [“Resilience”]**

row83 – **“Scope” : “Connectivity between entities”**

row90 – **“Goal” : “To provide connection between the IoT entities”**

In the second part of Figure 5.6, there is the second availability requirement, related to the “goal”: availability of user data for vendors (row 105). In this case, the “characteristic” taken into consideration is integrity. In fact, the integrity of the data assures that they can be available for whoever needs them.

row96 – **“Characteristic” : [“Integrity”]**

row105 – **“Goal” : “Availability of User Data for Vendors”**

In the first part of Figure 5.7, we can see the first privacy requirement. It is elicited by the need to express that the communications must be encrypted.

We model the requirement considering the confidentiality characteristic (row 111) and both the IoT entities involved (row 116) have the roles of “Encryptor” and “Decryptor” (row 115).

row111 – **“Characteristic” : [“Confidentiality”]**

row115 – **“Role” : “Encryptor, Decryptor”**

row116 – **“Type” : [“SCM, SF”]**

Then, in the second part of Figure 5.7, there is the second privacy requirement that is needed by the users in order to anonymize their data (row 135). The characteristics are three: anonymity, unlinkability, and confidentiality (row 126).

```

107 ~ "IoT_requirement_PRIV01" : {
108 ~   "Context" : {
109 ~     "Domain" : [{
110 ~       "Privacy" : {
111 ~         "Characteristic" : ["Confidentiality"] } }],
112 ~     "Environment" : "Smart Home",
113 ~     "Scope" : "Information excahnge between
114 ~       entities" } ,
114 ~   "Actor" : {
115 ~     "Role" : ["Encryptor, Decryptor"],
116 ~     "Type" : ["SCM", "SF"] },
117 ~   "Action" : {
118 ~     "Type" : ["Fulfill"] ,
119 ~     "Measure" : [ " " ]},
120 ~   "Goal" : "To guarantee an encrypted communication
121 ~     between the IoT entities"
121 ~ },
122 ~ "IoT_requirement_PRIV02" : {
123 ~   "Context" : {
124 ~     "Domain" : [{
125 ~       "Privacy" : {
126 ~         "Characteristic" : ["Anonymity,
127 ~           Unlinkability,Confidentiality"] } }],
127 ~     "Environment" : "Smart Home",
128 ~     "Scope" : "Data anonymization" } ,
129 ~   "Actor" : {
130 ~     "Role" : ["User","IoT Entity"],
131 ~     "Type" : ["Human User","SCM"] },
132 ~   "Action" : {
133 ~     "Type" : ["Fulfill"] ,
134 ~     "Measure" : [ " " ]},
135 ~   "Goal" : "To anonymize user data"
136 ~ },

```

Figure 5.7: JSON code part 5

row126 – “Characteristic” : [“Anonymity, Unlinkability, Confidentiality”]

row135 – “Goal” : “To anonymize user data”

The authentication need can also be expressed by identity requirements. In Fig 5.8, we can see two identity requirements. The second one is a sub-requirement of the first one (rows 137 and 152).

row137 – “IoT_requirement_IDNT01”

row152 – “IoT_requirement_IDNT01.1”

Because the needs require authentication, we decide to model the sub-requirement specifying that the authentication must be performed by username and password. An important aspect is that the goal of the first identity requirement is the same as the first security requirement shown in Figure 5.5. We have this situation because the authentication characteristic belongs to both the domains. These connections are taken into consideration concerning the traceability.

```

137 - "IoT_requirement_IDNT01" : {
138 -   "Context" : {
139 -     "Domain" : [{
140 -       "Identity" : {
141 -         "Characteristic" : ["Authentication"] } }],
142 -     "Environment" : "Smart Home",
143 -     "Scope" : "User Authentication" } ,
144 -   "Actor" : {
145 -     "Role" : "User",
146 -     "Type" : ["Human User"] },
147 -   "Action" : {
148 -     "Type" : ["Fulfill"] ,
149 -     "Measure" : [ " " ]},
150 -   "Goal" : "To authenticate the user"
151 - },
152 - "IoT_requirement_IDNT01.1" : {
153 -   "Context" : {
154 -     "Domain" : [{
155 -       "Identity" : {
156 -         "Characteristic" : ["Authentication"] } }],
157 -     "Environment" : "Smart Home",
158 -     "Scope" : "Username and Password Authentication"
159 -   } ,
160 -   "Actor" : {
161 -     "Role" : "User",
162 -     "Type" : ["Human User"] },
163 -   "Action" : {
164 -     "Type" : ["Fulfill"] ,
165 -     "Measure" : [ " " ]},
166 -   "Goal" : "To authenticate the user by username and
    password"
  },

```

Figure 5.8: JSON code part 6

Then, in Figure 5.9, we have the identity requirement related to the manipulation of user data (row 180). In fact, in order to be authenticated and recognized, the data must be stored. The characteristics are four (row 171): Attributes, Storable, Manageable, and Accountability. In fact, data can contain attributes (i.e., useful also in the case of pseudonymity). They must be storable and manageable. Besides, they provide accountability for users.

row171 –

“Characteristic” : [“Attributes, Storable, Manageable, Accountability”]

row180 – **“Goal” : “To manipulate user data”**

Finally, in Figure 5.10, we can see the JSON code related to safety requirements where the second one is the specification of the first one. The need expressed here is related to the temperature and the integrity of the device. In fact, according to the needs, the SCM must check the temperature level, and we decide that it cannot overcome 250°C.

row182 – **“IoT_requirement_SFT01”**

```

167 ~ "IoT_requirement_IDNT02" : {
168 ~   "Context" : {
169 ~     "Domain" : [{
170 ~       "Identity" : {
171 ~         "Characteristic" : ["Attributes, Storable,
172 ~           Manageable, Accountability" ] } ]],
173 ~     "Environment" : "Smart Home",
174 ~     "Scope" : "User Data" } ,
175 ~   "Actor" : {
176 ~     "Role" : "User",
177 ~     "Type" : ["Human User" ] },
178 ~   "Action" : {
179 ~     "Type" : ["Fulfill" ] ,
180 ~     "Measure" : [ " " ] },
181 ~   "Goal" : "To manipulate user data."
182 ~ },

```

Figure 5.9: JSON code part 7

```

row186 – “Characteristic” : [“Integrity”]
row197 – “IoT_requirement_SFT01.1”
row201 – “Characteristic” : [“Integrity”]

```

5.2.3 Step 3

After the second step is completed eliciting all the requirements connected to the needs, we write down the text of the requirements starting from the JSON codes shown from Figures 5.3 to 5.10.

As explained in 3.3, these requirements are written following IEEE 830-1993 and *statement (1)* formalism. Traceability is enhanced considering common goals, characteristics, and sub-requirements. Finally, the output of this third step will be the final elicited requirement. The requirements are shown in Tables 5.1, 5.2 and 5.3.

In order to show how the process must be performed compactly, we consider only these few requirements, but in a project, it is possible to have hundreds or thousands of requirements.

Thus, analysing the requirements presented in Table 5.1, 5.2 and 5.3, we can confirm that there is always at least one actor, one action and one goal as explained in section 3.4.6.

Then, as for TRST01, it is possible to have a secondary actor and a measure. On the other hand, it is possible to have requirements such as SFT01.1, including only

```

182 ▾ "IoT_requirement_SFT01" : {
183 ▾   "Context" : {
184 ▾     "Domain" : [{
185 ▾       "Safety" : {
186 ▾         "Characteristic" : ["Integrity"] } }],
187     "Environment" : "Smart Home",
188     "Scope" : "Temperature of the device" } ,
189 ▾   "Actor" : {
190 ▾     "Role" : "Checker",
191     "Type" : ["SCM"] },
192 ▾   "Action" : {
193 ▾     "Type" : ["Fulfill"] ,
194     "Measure" : ["Temperature Level"]},
195     "Goal" : "To check the level of the temperature"
196   },
197 ▾ "IoT_requirement_SFT01.1" : {
198 ▾   "Context" : {
199 ▾     "Domain" : [{
200 ▾       "Safety" : {
201 ▾         "Characteristic" : ["Integrity"] } }],
202     "Environment" : "Smart Home",
203     "Scope" : "Temperature of the device" } ,
204 ▾   "Actor" : {
205 ▾     "Role" : "Checker",
206     "Type" : ["SCM"] },
207 ▾   "Action" : {
208 ▾     "Type" : ["Fulfill"] ,
209     "Measure" : ["Temperature Level"]},
210     "Goal" : "To fix the maximum value of the
211       temperature in Celsius degree"
212 }

```

Figure 5.10: JSON code part 8

metrics. Each of these requirements is stored in a requirement database for each domain table as we have presented in 3.3.8.

Finally, as stated before, we have traceability between SEC01 and IDNT01. In fact, this type of connection is always present if the goal is the same, even if the domain of the requirements is different or if a requirement is a specialization of another one (i.e., in our case, IDNT01.1 is a specialization of IDNT01).

5.2.4 Step 4

In order to show an example about the requirement composition presented in Table 3.1 and discussed in Section 3.3.8, we obtain Tables 5.4, 5.5 and 5.6 related to the final elicited requirements SEC01, IDNT01 and IDNT01.1.

As we can see, Inner Traceability (IT) is kept, and there is a connection between the two requirements. Moreover, we can notice that for requirement IDNT01, Forward Traceability (FT) points to requirement IDNT01.1. Symmetrically, Backward Traceability (BT) points from IDNT01.1 to IDNT01. With the term *na*, we specify

Table 5.1: Requirements elicited using TrUsTAPIS - part 1

Trust Req.	TRST01 - The SCM shall trust a Smart Supermarket with a trust level above 0.5
Trust Req.	TRST02 - The User shall be trusted in order to perform actions
Usability Req.	USAB01 - The user shall be able to insert new recipes
Usability Req.	USAB02 - The user shall be able to download new recipes
Security Req.	SEC01 - The user shall be authenticated
Security Req.	SEC02 - The SCM shall delegate the Smart Hub to order the missing ingredients

that there is no connection of that type.

These relationships are represented in Figure 5.11.

As we can see, traceability is maintained between the identity and the security requirement. This relationship is bidirectional because if we have traceability between the ID of IDNT01 and the IT field of SEC01, thus, we will also have traceability between the ID of SEC01 and the IT field of IDNT01. The same reasoning is valid considering IDNT01 and IDNT01.1 (saved in the DB table “Identity sub-requirements” shown in Figure 5.11).

Table 5.2: Requirements elicited using TrUsTAPIS - part 2

Availability Req.	AVBT01 - The SCM shall be able to connect to the Smart Hub
Availability Req.	AVBT02 - The vendor shall access users data
Privacy Req.	PRIV01 - The SCM shall perform an encrypted communication with the Smart Fridge
Privacy Req.	PRIV02 - The user shall remain anonymous

Table 5.3: Requirements elicited using TrUsTAPIS - part 3

Identity Req.	IDNT01 - The user shall be authenticated
Identity Req.	IDNT01.1 - The user shall be authenticated by user name and password
Identity Req.	IDNT02 - The user shall provide his/her data in order to be registered
Safety Req.	SFT01 - The SCM shall be able to check its temperature level.
Safety Req.	SFT01.1 - The SCM temperature level shall be lower than 250°C

Table 5.4: Security requirement: SEC01

SEC01	The user shall be authenticated	na	na	IDNT01
-------	---------------------------------	----	----	--------

5.2.5 From Step 4 to Step 1

Now, we assume that the stakeholders do not like our choice about the authentication process that we formalized with requirement IDNT01.1. Thus, we need to shift from a user and password authentication to a code authentication. Therefore, because the needs are changed, so the requirements must be modified. In order to perform this action, the developers must delete the old requirement (IDNT01.1) and add the new one (IDNT01.2). It is better to add a new requirement instead of modifying the old one because each requirement ID must be related only to a single requirement for traceability purposes.

Through this procedure, we implement traceability in the requirements database

Table 5.5: Identity requirement: IDNT01

IDNT01	The user shall be authenticated	na	IDNT01.1	SEC01
--------	---------------------------------	----	----------	-------

Table 5.6: Identity sub-requirement: IDNT01.1

IDNT01.1	The user shall be authenticated by username and password	IDNT01	na	na
----------	--	--------	----	----

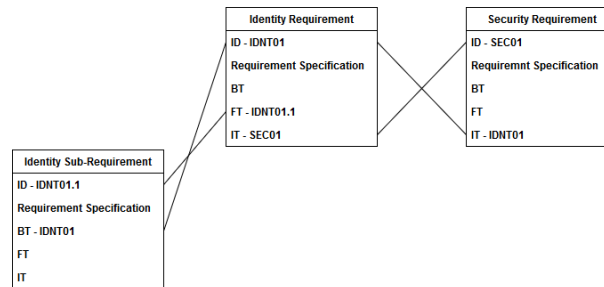


Figure 5.11: Traceability between requirements

connecting requirements among them.

In this case, as shown in Figure 5.11, we have connections between the identity requirement and the identity sub-requirement and between the identity requirement and the security requirement. However, if we try to add the new requirement and delete the old one without releasing the connections, an error is raised. This feature prevents developers to accidentally delete a requirement due to the connection between IDNT01 and IDNT01.1. Hence, this traceability feature permits developers to check the connection and, only after releasing it, it is possible to proceed with the requirement deletion. This feature is fundamental in order to avoid domino effects that can affect the other connected requirements.

In Figure 5.12, it is shown the JSON code related to IDNT01.2. We can state that it is similar to the one related to IDNT01.2, but the scope (row 8) and the goal (row 15) are related to the code authentication instead of user and password authentication.

row8 – “Scope” : “Code Authentication”

row15 – “Goal” : “To authenticate the user by code”

Finally, we can see the new final elicited requirement IDNT01.2 in Table 5.7 and the modified connection regarding IDNT01 in Table 5.8.

```

1 {
2   "IoT_requirement_IDNT01.2" : {
3     "Context" : {
4       "Domain" : [{
5         "Identity" : {
6           "Characteristic" : ["Authentication"] } } ] ,
7       "Environment" : "Smart Home" ,
8       "Scope" : "Code Authentication" } ,
9     " Actor " : {
10      "Role" : "User",
11      "Type" : [ "Human User" ] } ,
12    "Action" : {
13      "Type" : ["Fulfil"] ,
14      "Measure" : [ " " ]},
15    "Goal" : "To authenticate the user by code"
16  }
17 }

```

Figure 5.12: JSON code for IDNT01.2

Table 5.7: Identity sub-requirement: IDNT01.2

IDNT01.2	The user shall be authenticated by code	IDNT01	na	na
----------	---	--------	----	----

According to the transversal activities, we have already discussed traceability because it strictly binds to the requirements elicitation process. GR is successfully performed only if stakeholders and developers are satisfied with the elicited requirements, and there are no conflicts among them. The documentation is related to the collection of the requirements, their connection, and the originating needs. Then, threats and risks are considered in the requirements elicitation process, especially about trust and security. Finally, metrics are considered in requirements such as TRST01 and SFT01.1.

However, in this phase, a fundamental transversal activity is related to the decision making process that we illustrate in the following section.

Table 5.8: Identity requirement: IDNT01

IDNT01	The user shall be authenticated	na	IDNT01.2	SEC01
--------	---------------------------------	----	----------	-------

5.2.6 Decision Making

During the requirements elicitation process, among the others, we have elicited the following conflicting requirements:

- **IDNT02:** The user shall provide his/her data in order to be registered.
- **PRIV02:** The user shall remain anonymous.
- **AVBT02:** The vendor shall access users data.

They have been elicited following the stakeholders' needs, but different stakeholders, according to their diverse perspectives, may have various needs.

In fact, vendors would like to know more data is possible about their customers (i.e., for market surveys purposes). On the other hand, it is necessary for the customers to remain anonymous, or at least they need to know that only trusted users can access their data. Thus, if it is not possible to correct them in the first phase of the K-Model, it is possible to perform a decision-making process following POM presented in Section 4.8.2.

Therefore, without solving this conflict issue, the GR will fail, and it will not be possible to proceed to the following phase of the K-Model.

In order to implement POM we have to discuss about *goal*, *criteria* and *alternatives*.

The alternatives are three, and they are conflicting requirements. The criteria belong to the ones identified in Section 4.8.2, and they are defined according to our scenario. Finally, the goal is to decide which requirement to keep to maximize the level of trust of the IoT entity perceived by the “conflicting” stakeholders. Once this step is achieved, it will be possible to modify or delete it considering traceability aspects (i.e., release the connections among requirements).

5. 2. 6. 1 Criteria

According to our scenario, we have considered the following important aspects as decision criteria:

- **Context criteria**

1. **Stakeholders Importance.** When a product is developed, different stakeholders can make decisions, and they are significant for the project. In fact, a vendor could stop the project, or the customers could not buy the product. For example, if a vendor decides to stop the project, the customers will never have it. On the other hand, if the customers will not buy the product, it will be a failure. However, in order to be more specific, this criterion needs to be divided into two sub-criteria.

(a) *Vendors.* For the vendors, it is important to have as much information as possible about their customers to provide better customer service or market surveys.

(b) *Customers.* The customers would like to keep their data private. At least, they can accept to share the minimum information as possible only with trusted users.

2. **Faster.** It is possible that a requirement could be easier than another in order to be implemented. Thus, in the case of a strict deadline, this can be the most important parameter. This criterion can be objective.

3. **Cheaper.** This criterion is about the cost of the implementation. In the case of a low budget, it can be the most important parameter to be taken into consideration. It is an objective parameter.

- **Traceability criteria.** These criteria are objective. In fact, according to them, the more connections with other elements are developed, the most important the requirement is.

1. **Connected Requirements.** This information is provided by the traceability database 3.3.8. Thus, if a requirement is released, then the connected requirements can be affected by this operation.
 2. **Connected Needs.** Requirements are derived from needs, so this is an important element to be taken into consideration. This knowledge is guaranteed by the documentation activity 4.3. Thus, if a requirement is released or must be changed, it is possible to go back to the originating need and modify it following the stakeholders.
- **TrUStAPIS criteria.** These criteria depend on the type of requirements chosen as alternatives. It is important in order to highlight how much a requirement belonging to a particular domain (i.e., privacy) is connected to another domain (i.e., trust). In this use case scenario, the domains are the following: identity, privacy, and availability.

5. 2. 6. 2 *Alternatives*

As we presented earlier, the alternatives are the following:

1. **Privacy Requirement.** PRIV02 - The user shall remain anonymous.
2. **Availability Requirement.** AVBT02 - The vendor shall access users data.
3. **Identity Requirement.** IDNT02 - The user shall provide his/her data in order to be registered.

Our methodology will help to choose the most important requirement to assure the highest possible trust value for the IoT entity. However, in some cases, changing the requirements means that it is needed to change also the originating need.

As we explained earlier, the alternatives must fulfil the goal. Moreover, it is essential to rank the requirements in order to release or change the less important requirement and keep others solving the conflict issue.

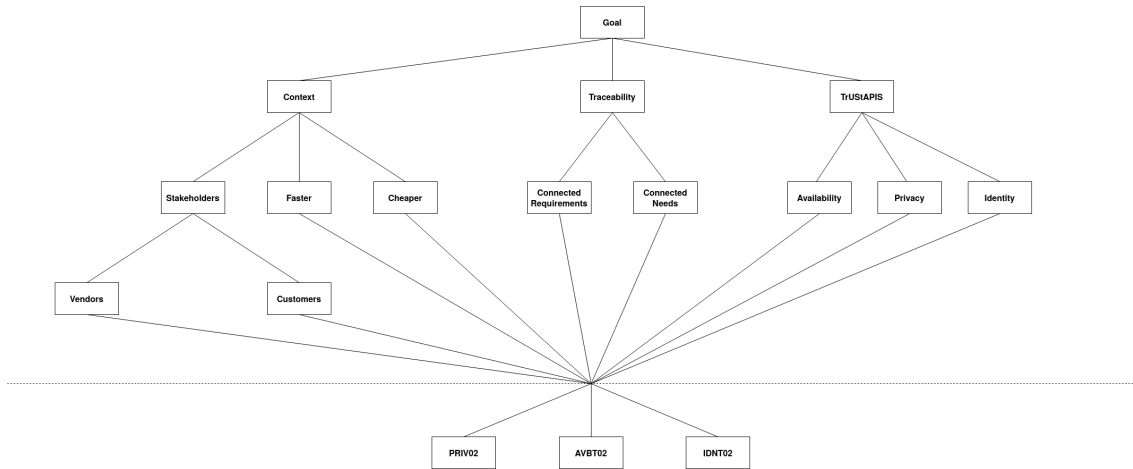


Figure 5.13: POM model related to our use case scenario

Now that we have presented criteria and alternatives, we can apply POM according to the proposed use case scenario. It is shown in Figure 5.13 where for the sake of simplicity, we represent the connection among the alternatives and the criteria with a single point of contact. The dotted line represents a many-to-many connection among them.

5. 2. 6. 3 Results and discussion

We implement POM as follows: we start the process by comparing the criteria according to the final goal; secondly, we will compare the sub-criteria to their main criterion; finally, we will compare the alternatives to their principal (sub)criterion. Each paragraph is named as the element considered for the comparisons. For the sake of simplicity, we will not represent all the rounds of comparisons in all of the following paragraphs, but we have used the methodology as we have shown in Section 4.8.3.

Goal We can state that the goal is reached considering the following elements:

$$Goal = \{Context, Traceability, TrUStAPIS\}$$

We need to create a ranked order among criteria to show which of them is most important according to the goal. Thus, following our methodology, we start comparing the Context (Cx) with the Traceability (Ty). We decide that Cx is the most important, so then we have to compare Cx with TrUStAPIS (Ts), and we decide that Cx is more important than Ts. We consider the context as crucial for both the decisions because, as we have stated earlier, it is an element always present and strictly connected to a particular parameter. After the first round, we know that Cx is the most important element. We need another round in order to decide between Ty and Ts. We decide that Ty is the most important because the more requirements or needs are connected, the most important is the requirement.

Thus, according to the goal, the criteria are ranked and normalized as follows:

$$Cx = (1/2)$$

$$Ty = (1/3)$$

$$Ts = (1/6)$$

Context The context is composed of three sub-criteria: the stakeholders (Sk) criterion and others strictly dependent on the implementation of the requirements: faster (Fs) and cheaper (Ch). Comparing the stakeholders to the faster and cheaper criteria, we decide that the stakeholders are the most important because they provide the needs of the product. Then, we give to faster and cheaper criteria the same importance.

Thus, the values related to the sub-criteria of the context are:

$$Sk = (3/5)$$

$$Fs = (1/5)$$

$$Ch = (1/5)$$

Stakeholders Stakeholders are very important for any project. They are the actors having an interest in the system. In this use case scenario, we have identified two main stakeholders strictly related to the conflict requirements. They are the vendors (Vn) and customers (Cs). We decided that they are equally important. In fact, it

is true that if the vendors do not deliver the product, it will not be used by the customers. On the other hand, it is also true that if the customers will not trust and buy the product, it will be a failure.

For these reasons, the values are the same for all the stakeholders:

$$V_n = (1/2)$$

$$C_s = (1/2)$$

Vendors We can state that the vendors are the IoT device producers, and the requirements are ordered considering their importance for them.

The best way to make this order is by asking them directly which requirement they prefer, but in this use case, we assume that the developer can perform this task by analyzing the collected needs and requirements.

For the vendors, we find out that AVBT02 is the most important requirement. Secondly, IDNT02. The last one is PRIV02. This order is due because AVBT02 is the only requirement that considers the vendors directly, and it is the one that represents better their interests on the IoT entity. Then, in order to have the needed information, IDNT02 satisfies their need. The last one is PRIV01 because if the patients and doctors remain anonymous, the vendors will not have valuable information for market purposes.

The normalized values are the following:

$$AVBT02 = (1/2)$$

$$IDNT02 = (1/3)$$

$$PRIV02 = (1/6)$$

Customers For the customers, the most important requirement is PRIV02 because it guarantees them to be anonymous. The other two requirements are equally important for them because they do not like to share their information or provide them.

Thus, the rounds of comparisons produce these normalized values:

$$PRIV02 = (3/5)$$

$$AVBT02 = (1/5)$$

$$\text{IDNT02} = (1/5)$$

Faster As for the stakeholders criteria, this criterion and the following one are strictly dependent on the context. The more a requirement is simple to be implemented, the faster it is. For this reason, after the first round of comparisons, we have identified AVBT02 as the most important. In fact, it does not need any filter because it is merely a reading operation. Then, there is IDNT02 because it requires to write the data in the database. Finally, the most complex requirement is PRIV02 because it requires to anonymize the data.

Thus, according to the faster sub-criterion, the final values are:

$$\text{AVBT02} = (1/2)$$

$$\text{IDNT02} = (1/3)$$

$$\text{PRIV02} = (1/6)$$

Cheaper This criterion is essential in the case the budget is limited or the stakeholders want to maximise the income. However, in the early phases of the SDLC, it is possible that even if developers and stakeholders avoid to implement an expensive requirement, there is the possibility that some issues arise in the following phases of the SDLC. In this case, the amount of money spent to solve the issues will be higher than the money that would be spent on the original expensive requirement [56].

Anyhow, in our case, after the first round of comparisons, IDNT02 is considered the cheapest requirement to be developed. Then, AVBT02. Finally, there is PRIV02. The last one is both the slower and the more expensive because implementing anonymity is the most challenging task considering the other requirements. IDNT02 is considered the cheapest because it is the basic requirement to be implemented.

The normalized values are:

$$\text{IDNT02} = (1/2)$$

$$\text{AVBT02} = (1/3)$$

$$\text{PRIV02} = (1/6)$$

Traceability Traceability is a transversal activity of the K-Model [50], and it is crucial in order to connect requirements among them. In our scenario, there are two sub-criteria. They are related to the connected requirements (Con_R) and the connected needs (Con_N). We decide to give more importance to Con_N because the needs are the real motivation behind a product, so if a requirement is strictly connected to a need, it should be more important than another one only connected to a requirement.

The values related to them are:

$$\text{Con_N} = (2/3)$$

$$\text{Con_R} = (1/3).$$

Connected Requirements This criterion can be completely objective. The operation needed is to count how many requirements are connected to the requirement under consideration. However, in this thesis we have elicited only a few requirements in order to illustrate the procedures, so we need to choose them in a subjective way. Thus, we consider AVBT02 and IDNT02 equally important. Finally, there is PRIV01. We assume that even if it is the more challenging requirement to be implemented, it does not need many other requirements to be connected with in order to be developed.

Thus, the final normalized values about Con_R are:

$$\text{AVBT02} = (2/5)$$

$$\text{IDNT02} = (2/5)$$

$$\text{PRIV02} = (1/5).$$

Connected Needs In this case, it is essential to consider how many needs are connected to a single requirement. It is possible that multiple needs originate a single requirement or even that a requirement is not connected to any need but only to other requirements (especially if it is a sub-requirement [49]).

In the case of a real use case scenario, it is possible to count all the connected needs and objectively apply our methodology. However, in this case, we proceed as we did for the previous element.

After the first round of comparisons, we decide that IDNT02 is the requirement connected to more needs because it is related to many user information. For this reason, numerous needs shall be connected to it. The second requirement is PRIV01. In fact, we assume that not only the customers require it, but also data protection regulations needs are connected to the privacy requirement (i.e., GDPR [154]). The final requirement is AVBT02.

The normalized values, according to the connected needs, are:

$$\text{IDNT02} = (1/2)$$

$$\text{PRIV02} = (1/3)$$

$$\text{AVBT02} = (1/6)$$

TrUStAPIS In order to decide which requirement to keep, it is crucial to study the relationships among the requirements domains and the conflicting requirements. This criterion gives a holistic view of how a particular domain requirement can be related to others.

In this case, we compared availability to privacy, deciding that the latter is more important than the former. Then, we compare the privacy requirement to the identity requirement, and we decide that they are equals. For this reason, there is no need for a third round of comparison (privacy is more important than availability, and consequently, because identity is equal to privacy, it is more important than availability, too).

Thus, the values related to the TrUStAPIS sub-criteria are:

$$\text{Id} = (2/5)$$

$$\text{Pr} = (2/5)$$

$$\text{Av} = (1/5)$$

Availability After the availability requirement itself, the second one is IDNT02, because only if the customers provide their information they will be available. Finally, according to availability, PRIV02 is the least important requirement because if the data are anonymous, they could not be easily available.

For availability, the normalized values are:

$$\text{AVBT02} = (1/2)$$

$$\text{IDNT02} = (1/3)$$

$$\text{PRIV02} = (1/6)$$

Privacy According to privacy, the most important requirement is PRIV01. Then, AVBT02 and IDNT01 are equally important because neither one requirement nor the other guarantees privacy.

The final values are:

$$\text{PRIV02} = (3/5)$$

$$\text{AVBT02} = (1/5)$$

$$\text{IDNT02} = (1/5)$$

Identity After the identity requirement, the second one is AVBT02 because it requires that the identity data should be available for the vendors. On the other hand, PRIV01 does not provide any identity information.

The final normalized value about identity are:

$$\text{IDNT02} = (1/2)$$

$$\text{AVBT02} = (1/3)$$

$$\text{PRIV02} = (1/6)$$

Final priority After calculating the normalized local weights, we have to compute the final priority related to each of the alternatives according to the goal. In order to perform this activity, we must sum every value related to the single alternatives multiplying it for each value of the sub-criteria and criteria above them.

Because the values are normalized, the sum of the final results will be 1.

In Figure 5.14, there are the values derived in the previous paragraphs. They are the same calculated earlier.

In this figure, to avoid having many lines and boxes for the alternatives, we have summarized them in a single box. P2, A2, and I2 are privacy, availability, and identity requirements.

Thus, in order to calculate the single priority related to the conflicting requirements, starting from P2, we will have.

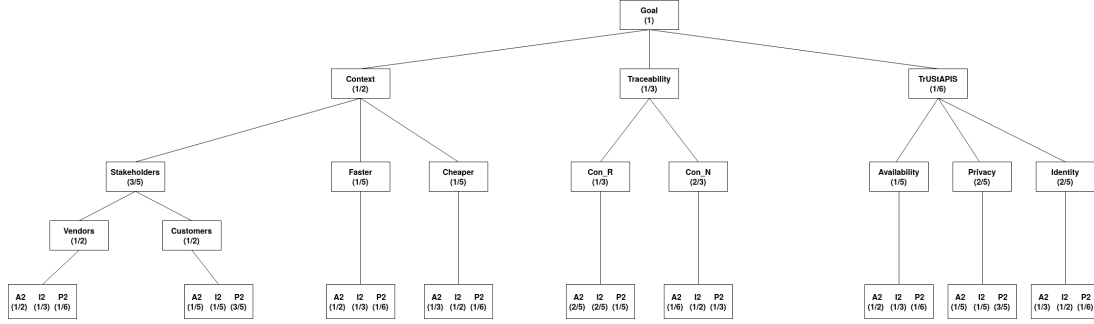


Figure 5.14: POM model related to our use case scenario

$$\mathbf{P2} = (1/6) * (1/2) * (3/5) * (1/2) + (3/5) * (1/2) * (3/5) * (1/2) + (1/6) * (1/5) * (1/2) + (1/6) * (1/5) * (1/2) + (1/5) * (1/3) * (1/3) + (1/3) * (2/3) * (1/3) + (1/6) * (1/5) * (1/6) + (3/5) * (2/5) * (1/6) + (1/6) * (2/5) * (1/6) = \mathbf{0,301}$$

For the other requirements, we have:

$$\mathbf{A2} = (1/2) * (1/2) * (3/5) * (1/2) + (1/5) * (1/2) * (3/5) * (1/2) + (1/2) * (1/5) * (1/2) + (1/3) * (1/5) * (1/2) + (2/5) * (1/3) * (1/3) + (1/6) * (2/3) * (1/3) + (1/2) * (1/5) * (1/6) + (1/5) * (2/5) * (1/6) + (1/3) * (2/5) * (1/6) = \mathbf{0,322}$$

$$\mathbf{I2} = (1/3) * (1/2) * (3/5) * (1/2) + (1/5) * (1/2) * (3/5) * (1/2) + (1/3) * (1/5) * (1/2) + (1/2) * (1/5) * (1/2) + (2/5) * (1/3) * (1/3) + (1/2) * (2/3) * (1/3) + (1/3) * (1/5) * (1/6) + (1/5) * (2/5) * (1/6) + (1/2) * (2/5) * (1/6) = \mathbf{0,377}$$

The results are rounded to three digits after zero. The total is:

$$\mathbf{P2} + \mathbf{A2} + \mathbf{I2} = \mathbf{1}$$

Therefore, the most important requirement is IDNT02, followed by AVBT02. Thus, the least important requirement is PRIV01. For this reason, PRIV01 will be the requirement released or modified in order to solve the conflict requirements.

However, because the customers will not like that anyone will access their data, they will accept the change only if another requirement will be elicited instead of PRIV02. This requirement is related to trust and guarantees that their data will be accessed only by trusted users. Moreover, will be created a new privacy requirement

Table 5.9: Trust requirement: TRST02

Privacy Req.	PRIV03 - User data shall be kept private
Trust Req.	TRST02 - Customer data shall be accessed only by trusted users

stating that the users' data must be kept private: PRIV03 Thus, now we have an additional elicited requirement shown in Table 5.9: TRST02.

After this procedure is finalized, it is possible to proceed to the GR and the third phase of the K-Model.

5.3 Model

After the collection of needs and elicited requirements, we need to model them using the diagrams shown in Section 3.4.

Each diagram will cover different aspects of the SCM utilization. These models will be documented for the following phases of the SDLC. We will show the different cases in each section to illustrate how the diagrams can be used.

Besides, we will present an experiment showing how traceability among requirements works and how it helps avoid domino effects. This example is presented in Section 5.3.8.

5.3.1 Use Case Diagram - UCD1

SCM provides various functionalities. One of them that we consider very important from a trust, privacy and security perspective is the possibility to store users' private data to access them when needed. In this case, the device shall be allowed to store the private data of the user locally. In order to proceed, the user must approve the action. Otherwise, the data will be asked again when they will be needed.

For this use case diagram, we need to consider three domains (privacy, security, and trust) that might affect the privacy of the data, the security of the storage, and

the trust of the user providing his/her personal data.

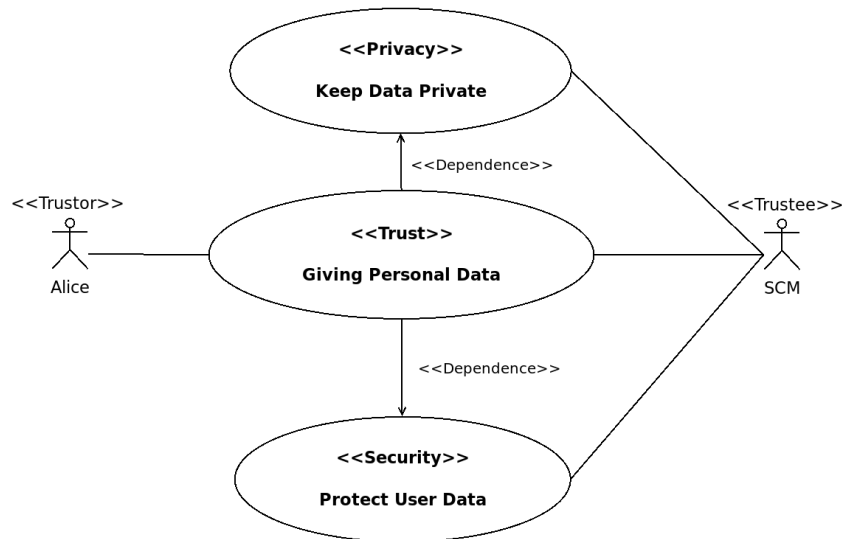


Figure 5.15: UCD1 - User data

Figure 5.15 shows the use case for our example. As we can see, there are two actors involved. One is the user (i.e., Alice), who is considered, from a trust perspective, as the trustor. The other one is the SCM, which in this case is considered as the trustee. We model three use cases related to three different domains: privacy, trust, and security. The trust use case is named “Giving Personal Data”. It is connected through a «Dependence» connection to the privacy use case named “Keep Data Private” and to the security use case named “Protect User Data”. In fact, we can state that in order to share private data, the user must trust the device and the utilization of the same data. Therefore, this trust relationship is strongly dependent on whether the data will be kept private and secure (i.e., an encrypted and protected database). Besides, we can see that Alice is connected to the trust use case. We assume that the user gives the data as trust action. She is not connected directly to the privacy or security use case, but only through the trust use case. The motivation is that the SCM only provides the privacy and security actions, and for Alice, these actions are surely important, but *transparent*. On the other hand, the SCM is involved in all the use cases, firstly as a trustee in order to keep the trust of the trustor. Moreover, the SCM needs to store the data privately and securely.

To conclude, the use cases have general names because it will be the developer who

decides how to implement them. Using this diagram, it is more important to model *what* the system shall do more than *how* the system implements the functionalities. In fact, the UCD is a general diagram, and the development of the rules must be implemented modeling other diagrams (such as an activity diagram or a sequence diagram)

5.3.2 Class Diagram - CD1

We use our CD version to model the entity, methods, and attributes needed to develop this functionality. In order to recognize the owner, the device must register the owner data (i.e., name, birth date, credit card) keeping them private. Moreover, in order to allow users to order goods, the device must get a code and check it in order to recognize a trusted user, according to requirement IDNT01.2. Other recognized users can be guests and children. They can perform actions only if Alice provides them the right code. Finally, the available services must be trusted by the owner and the device to proceed with the transactions.

We summarize all these concepts in the class diagram (CD1) shown in Figure 5.16.

Starting from the contexts, it is possible to see that some classes are related to context number 5. This context is shown later in Section 5.3.7). The classes related to it are **Order** and **SmartSupermarket**. The other two classes are important for every context, so the context box is empty. At the center, we have the **User** class. We can see from the stereotypes that this class is related to the trust, privacy, and security domains. These domains are chosen because the user is a *trustor* of the services. In fact, he/she has a role that concerns security constraints, and the recording of the voice can raise a privacy issue.

The attributes related to the **User** class are:

1. **name**. This attribute represents the name and surname of the user. They must be stored privately and securely.
2. **birth-date**. From this attribute, it is possible to calculate the user's age. It

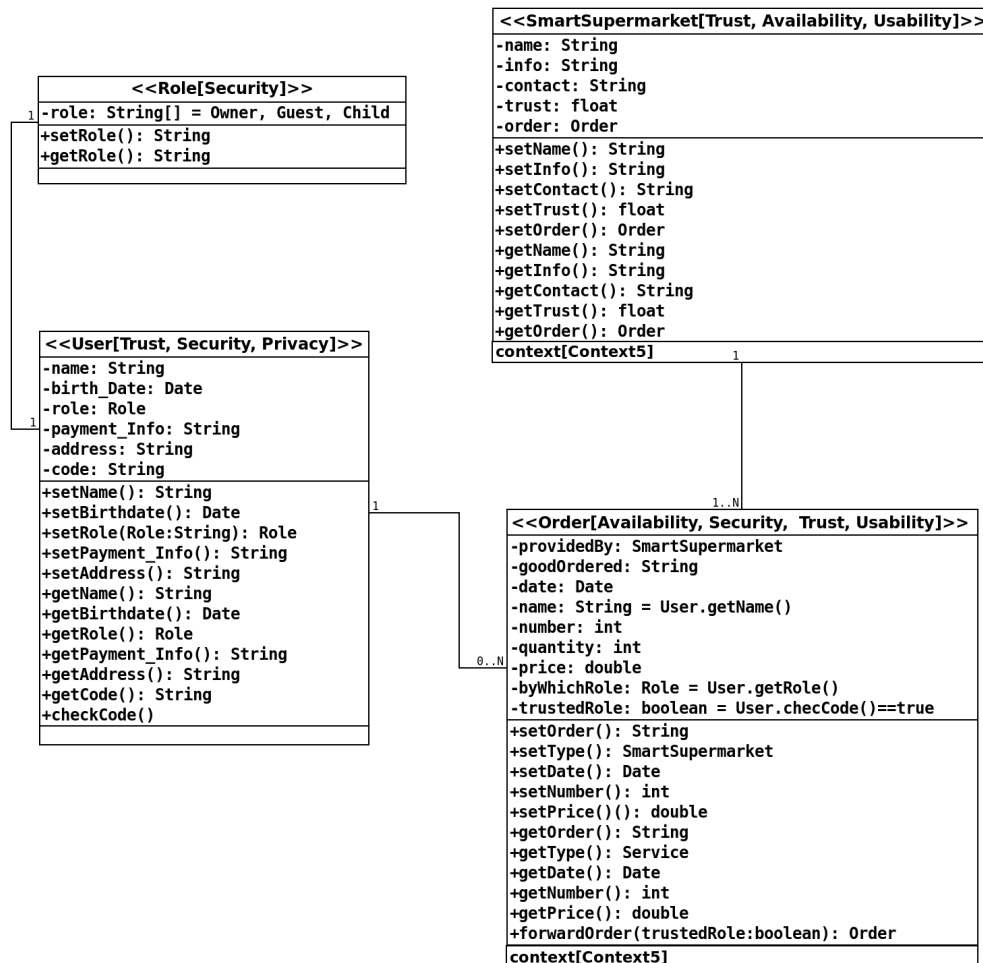


Figure 5.16: CD1 - SCM and SM service classes

is important for roles (i.e., denote if a user is a child), and it must be stored privately and securely.

3. **role**. The possible roles are owner, guest, and child. Each of them must be registered as a user. It is also a class.
4. **payment-info**. The credit card data of a user. They must be stored privately and securely.
5. **address**. The address of the owner. It is important in order to deliver the ordered goods, and it must be stored securely and privately.
6. **code**. It is the only way to perform actions and recognize a trusted user.

The methods are used in order to *set* and *get* the attributes with the proper values. In this case, to model trust, we have considered a decision model. In fact, if the user is allowed or not to perform an action is dependent on the right code. This aspect is related to trust, and it reflects the decision that must be made by the device in order to trust or not the user.

The class **Order** has the following attributes:

1. **providedBy**. It is related to the class SmartSupermarket. In fact, an order must be satisfied by a Smart Supermarket.
2. **goodOrdered**. It is related to the requested product (i.e., flour, butter, milk).
3. **date**. It is related to the day of the purchase.
4. **name**. The name of the user that makes the order.
5. **number**. The id of the order.
6. **price**. The total amount spent.
7. **byWhichRole**. The user role.
8. **trustedRole**. The only way to be trusted is by the right code. It is called the method `checkCode()` of class User in order to validate it. For this reason, in this class, trust is a *boolean* value.

The methods are related to the attributes in order to *set* the values or to *get* them. There is one more method that is related to forward the order (in the case that the user is trusted). In this case, to model trust, we consider decision model rules, and the considered parameter is related to the user's code.

Finally, the class **SmartSupermarket** has the following attributes:

1. **name**. It is the name of the service.
2. **info**. More information about the service.
3. **contact**. It can be a telephone number, a website, or an e-mail.

4. **trust.** It is the trust level of the service. In this case, trust is represented by a *float* value because each service could have different trust levels in order to be trusted. Anyhow, the value is considered between 0 and 1, where the former refers to no trust and the latter to the maximum trust level.
5. **order.** This parameter is related to the order that the SM must satisfy.

Considering the methods, we can focus on the ones strictly related to trust. In this case, we model trust following evaluation model rules. In fact, trust is dependent on the different trust levels of the service organized as reputation values. We decided to consider them as floats in this example, but they can also be considered as double or integer parameters. The chosen metric can create different trust levels. We decide to consider them with the following ranges.

If x (i.e., the trust value) is lower than 0.5, the service is not trusted. On the other hand, if x is higher or equal to 0.5, the service is trusted. The ranges are between 0 and 1. Anyhow, possibilities and ranges could be numerous, but we decide to consider this simple case where the service is simply not trusted or trusted. After the outcome of the order, the user can change the value using the `setTrust()` method.

The connection between *User* and *Role* is 1 to 1 because we assume that the role does not change for a particular context. It should be possible to have a different role for each possible context, but we do not model this case in this scenario. The connection between *User* and *Order* is 1 to 0/N because a user could perform zero orders or more. Finally, the connection between *SmartSupermarket* and the *Order* is 1 to 1/N because a Smart Supermarket can provide at least one order or more.

5.3.3 Activity Diagram - AD1

With this example, we model an AD to implement the users' possibility to store their data in the SCM.

In Figure 5.17, we show how an AD related to UCD1 could be implemented. Specifically, we implement the possibility to secure user data. As shown in Figure 5.15, there are at least three domains involved: trust, security, and privacy. These

activities are modeled to specify the use case actions. In addition to these domains, in this diagram, we also consider the availability domain.

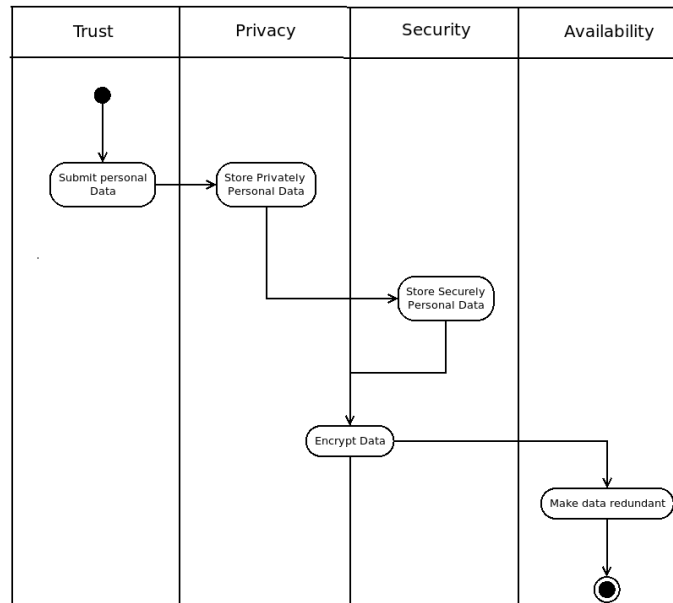


Figure 5.17: AD1 - Securing user data

We use the swimlanes to separate the activities belonging to different domains. About the trust domain, we model the activity “Giving personal Data”. In fact, only if the user trusts the SCM it is possible to perform this activity. It belongs to the trust domain because the user must trust the device in order to reveal personal data. Secondly, the following activity is part of the privacy domain, and it is called “Store privately personal Data”. This means that the SCM must store these data considering privacy aspects. Then, there is an activity belonging to the security domain: “Store securely personal Data”. This is a generic activity, and the developer will decide how to store the data securely. The following activity is related to the encryption of the data. It belongs to the privacy and security domain because, by encryption, it is possible to enhance the security of the data (i.e., avoiding unauthorized use of it) and also the privacy of the data. Finally, the last activity belongs to the availability domain, and it requires that the data could be made redundant. Through redundancy, the possibility of losing data are minimized. There is no written rule on how this redundancy will be implemented. Through this diagram, it is more

important to model the *what* rather than the *how*.

5.3.4 Sequence Diagram - SD1

The SCM is built to interact with other IoT entities offering many services to the user. One of them is the interaction with the smart fridge belonging to the same smart-home. Thus, using the SD, we model the interaction between the user, the SCM, and the SF in order to check if any good is missing to make a cake.

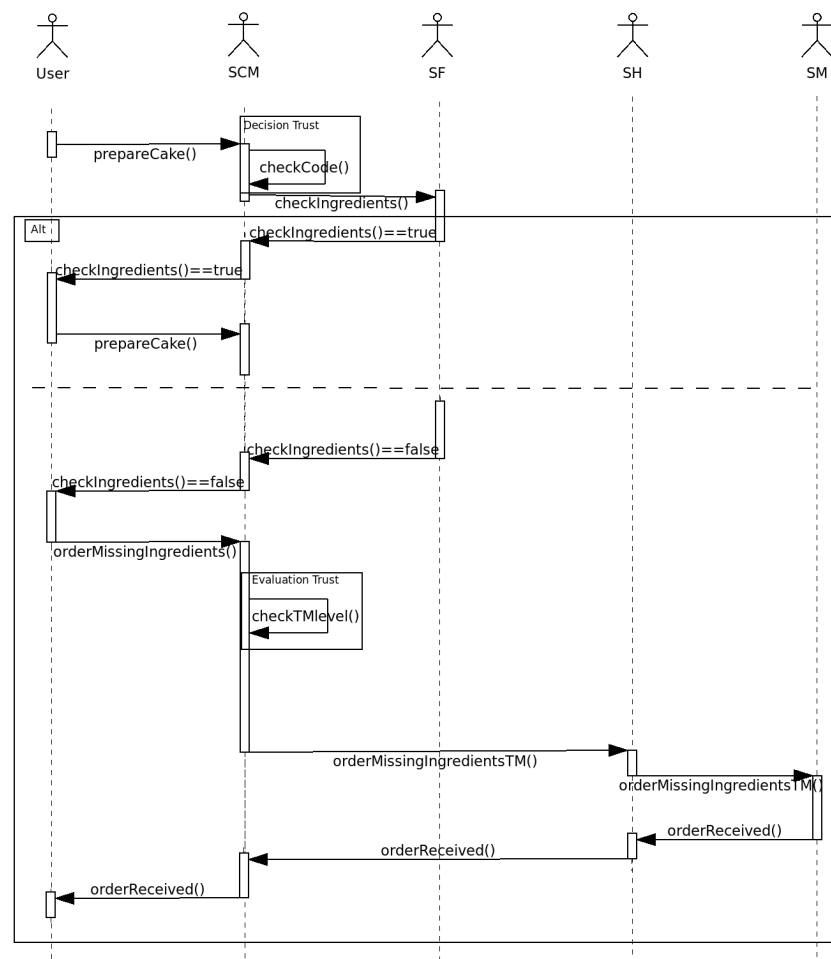


Figure 5.18: SD1 - User, SCM, SF, SH and SM interactions.

In the diagram shown in Figure 5.18, the interaction is started by the user asking the SCM to prepare a cake. To proceed, the SCM checks if the user is allowed to perform this action by code. In this interaction, there is a decision trust computation

because the code match can be considered as an authentication process. This diagram shows that the user code is correct, so the SCM checks the smart fridge for the needed ingredients. SF replies with the information required.

Now, we model two different cases. One is that the needed ingredients are present. The second one is that they are not available, so the SCM will ask the authenticated user to order the missing ingredients from a trusted SM. This operation will be performed through a smart hub in order to protect the IoT entities. However, before proceeding with the missing ingredients' order, the SCM checks with an evaluation trust process, which smart supermarket to contact. Then, it communicates with the SH that will be delegated to order the missing ingredients.

In this scenario, we modeled the possibility that the smart supermarket replies that the order has been received. This information is forwarded up to the user, and the SD ends.

5.3.5 State Machine Diagram - SMD1

Through this diagram, which we show in Figure 5.19, it is modeled an important feature of the SCM, the upload of new recipes.

Firstly, there is a state related to login to the SCM. In order to pass to the following state, a trust trigger is needed. In fact, without the proper credentials, it is not possible to access the system. Here, we have two possibilities: the state machine ends, or it is possible to provide other credentials to gain trust to access the SCM. We do not model how and which credentials are needed. We model the possibilities to have an extra step in order to have access to the SCM (i.e., a secret question). If the extra credentials are provided, or the first credentials are enough, the state machine passes to the next state, which is a composite state. A composite state has another starting point, and it is like a state machine inside a single state. In this composite state, the SCM asks to insert a title and then specify which item is needed and its quantity. Then, in order to insert all the ingredients, there is a loop that will terminate when all the ingredients are considered. In this case, the composite state ends, and it will be reached a state considering that the new recipe is correctly

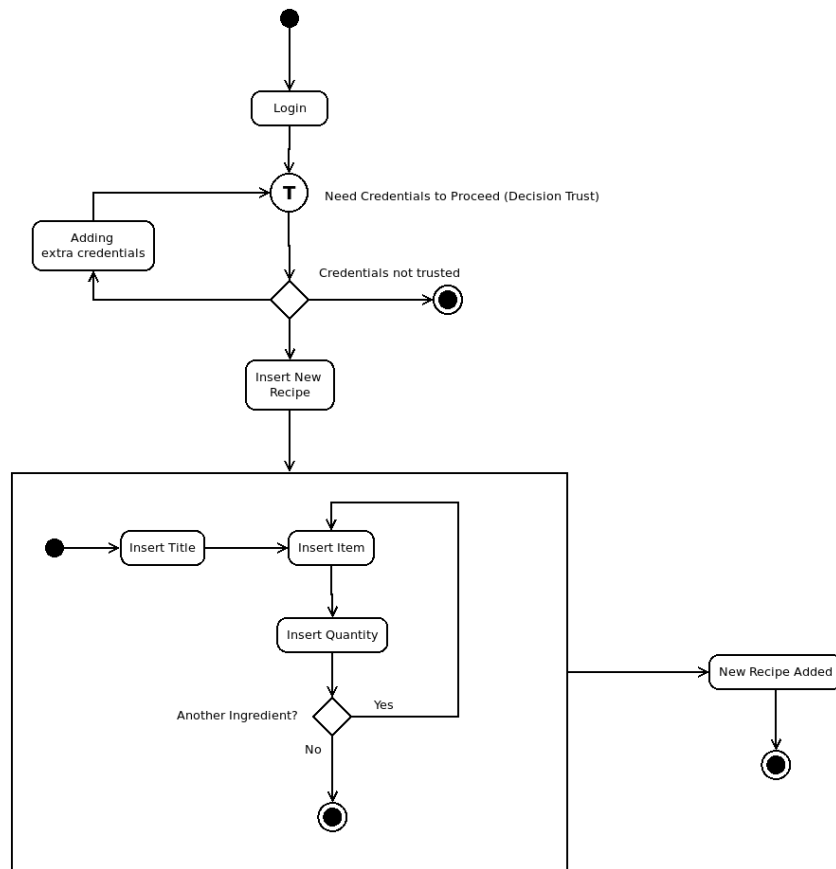


Figure 5.19: SMD1 - Upload a recipe into the SCM

added to the SCM recipes database. After this state, the flow ends.

5.3.6 Requirement Diagram - RD1

In this example, we show how a Requirement Diagram is drawn and how its elements are connected. The requirements specified here must be elicited in the previous phase of the K-Model, as explained earlier. In our case, we consider an RD that is shown in Figure 5.20. This RD is connected to UCD1. In fact, there are requirements connected to the privacy action belonging to UCD1. This example is important to understand how the requirement diagram can be connected to the other diagrams and how it can be drawn.

RD1 is composed of three main requirements. The Ids and the texts of the requirements are shown in Figure 5.20. The trust requirement (id: TRST02) satisfies

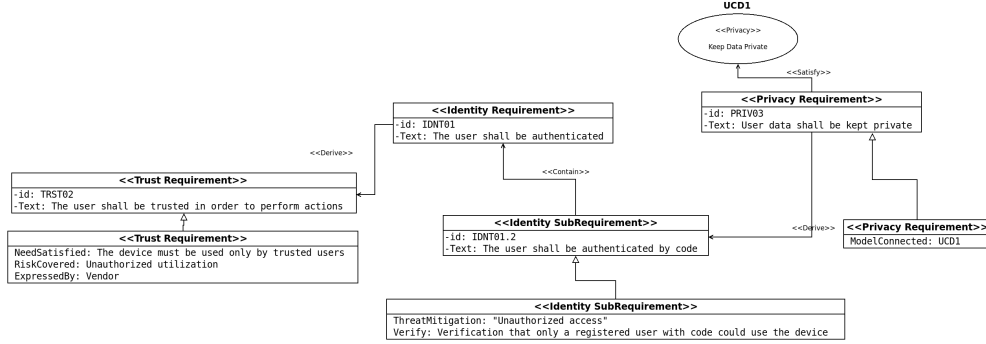


Figure 5.20: RD1 - Trust, Identity and Privacy Requirements

a need expressed by the vendor. Moreover, this requirement mitigates the risk of unauthorized use. The Identity Requirement (id: IDNT01) derives from TRST02, and it is related to the fact that a user shall be authenticated in order to be trusted (decision trust). IDNT01 has a sub-requirement (id: IDNT01.2) that specializes the authentication process through a provided code. The sub-requirement must be verified in the verification phase. Furthermore, it mitigates the threat related to "Unauthorized Access" because only a legitimate user can perform actions.

To enhance the protection, the privacy requirement (id: PRIV03) deriving from IDNT01.2 states that user data shall be stored privately. This requirement is connected to the UCD1 element "Keep Data Private" through a satisfy connection. This connection is represented also by the element *ModelConnected*.

5.3.7 Context Diagram - XD1

The context diagram is related to the possible contexts belonging to an IoT entity. It is not important to model all the contexts in a single diagram. The developer can choose which context to model in each diagram according to its task.

In Figure 5.21, we consider four contexts that can be implemented in the SCM. Some of them have been presented in the previous diagrams. The contexts are cook, interaction, recipes, and User Data. For each of these contexts, there are one or more domains related to it. The contexts are defined as follows:

- **Context 1** (*Cook*). In this case, context 1 domains are related to usability

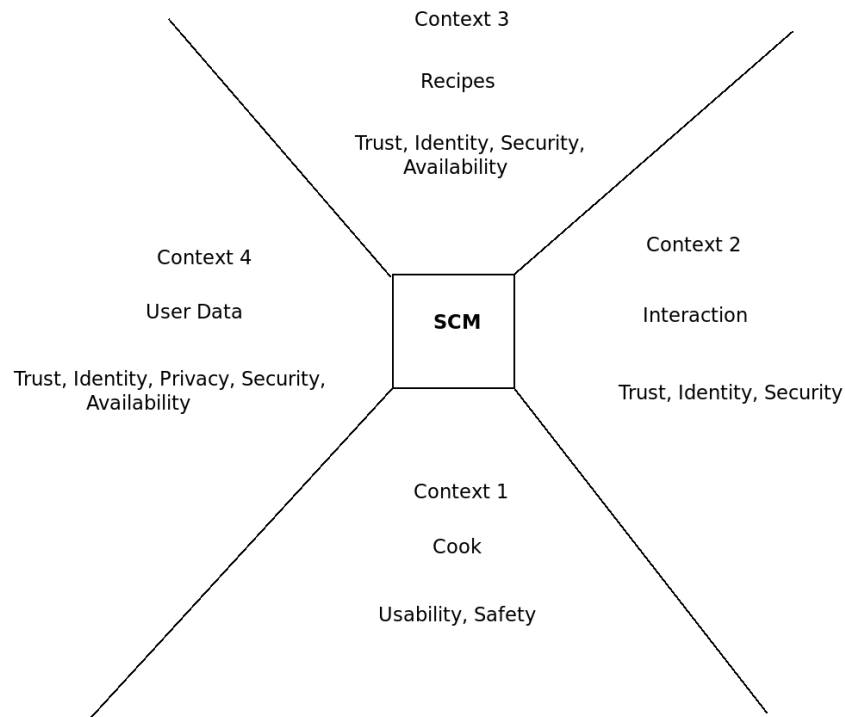


Figure 5.21: XD1 - Contexts: Cook, Interaction, Recipes, User Data

and safety. It is strictly connected to the physical part of the IoT entity, and it also specifies the final goal of the SCM (i.e., to bake a cake).

- **Context 2** (*Interaction*). This context is related to the interaction that the SCM must have with other IoT entities or authenticated users. The domains are trust, identity, and security. In fact, in order to allow the interactions, the users/IoT devices must be trusted and authenticated. Moreover, the communication must be securely protected.
- **Context 3** (*Recipes*). For this context, we select the trust, identity, availability, and security domains. In fact, in order to insert new recipes, a user must be trusted. The identity is related to the authentication part. Security, to the protection of the data and availability to the possibility to have these data available in order to be used.
- **Context 4** (*User Data*). For this context, the SCM needs to store trusted user data keeping them private and providing security.

As we said earlier, if there are many contexts, it would be a problem to use a graphical view. So, we can see in Table 5.10 the database view related to XD1.

Table 5.10: XD1 - Database view

Context 1	Cook	[Usability, Safety]
Context 2	Interaction	[Trust, Identity, Security]
Context 3	Recipes	[Trust, Identity, Security, Availability]
Context 4	User Data	[Trust, Identity, Privacy, Security, Availability]

5.3.8 Traceability Diagram - TD1

The TD is a diagram of diagrams, and it can be considered a meta-diagram. As we stated earlier, it is useful to keep track of the connections among the other diagrams to help the developers avoiding domino effects after deleting or modifying the connected diagrams.

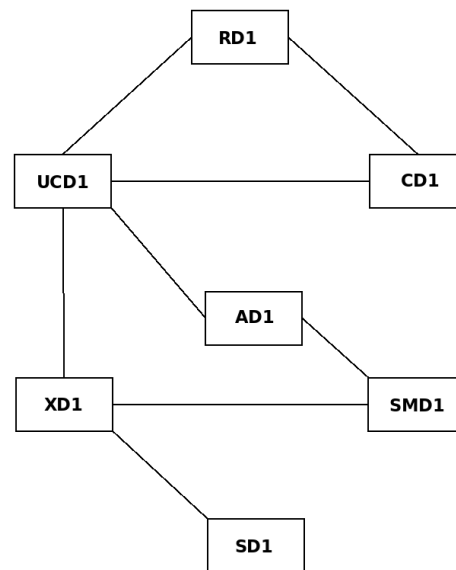


Figure 5.22: TD1 - Traceability among diagrams

Considering our modeling scenario, the correspondent TD is presented in Figure 5.22. We can see that the UCD1 is connected to four diagrams: RD1, CD1, AD1, and XD1. In fact, the use cases presented in UCD1 are modeled in different ways

using the other diagrams. It is then possible to see a connection between CD1 and RD1 because the requirements specified in RD1 are developed in CD1. XD1 is also connected to SD1 because the interaction among IoT entities is considered in both of them. Finally, AD1 is connected to SMD1 because both diagrams consider the store of the user's personal data. For this reason, SMD1 is also connected to XD1, where the user data context is proposed as context number 4.

As for the XD, it could be difficult to represent this diagram graphically, so it is possible to use a database view as presented in Table 5.11.

Table 5.11: TD1 Example - Database view

AD1	[UCD1, SMD1]
CD1	[RD1, UCD1]
RD1	[CD1, UCD1]
SD1	[XD1]
SMD1	[AD1, XD1]
UCD1	[AD1, CD1, RD1, XD1]
XD1	[SD1, SMD1, UCD1]

In the following section, we will focus on the traceability database in order to show how it is structured with a higher number of models.

5.3.8.1 Traceability Database

This section presents the third part of the step-by-step methodology related to the model phase illustrated in Section 3.4.9. We show how the traceability database must be structured, considering the diagrams presented in the previous section plus other dummy diagrams to show how traceability works with a larger number of diagrams. In fact, the more complex the scenario is, the more elements will be connected among them.

Thus, even if there are other possibilities to show how the diagrams are connected among them (i.e., visual goal diagrams), we have chosen the database visualization because we think it is the most effective way to show how the diagrams are connected

among them. Moreover, by creating the database tables, we can also represent important aspects of any diagram (i.e., the domains) or specific ones (i.e., activities for the AD).

Each database table is related to a particular diagram (i.e., CD or SMD), where the primary key is the ID of the related diagram (i.e., CD1 for a CD diagram). The traceability diagram is represented by a table related to the connections among diagrams. In the rows of this table, we have all the IDs of the connected diagrams. If there is a connection among requirements, they will be represented in the same row. In the relational databases is represented as a multi-multi relationship database.

In Figure 5.23, we can see the database chart where we have a traceability database connected to all the databases of the other diagrams. In this scenario, we assume that there is no connection among diagrams related to the same type (i.e., UCD1 cannot be connected to UCD2).

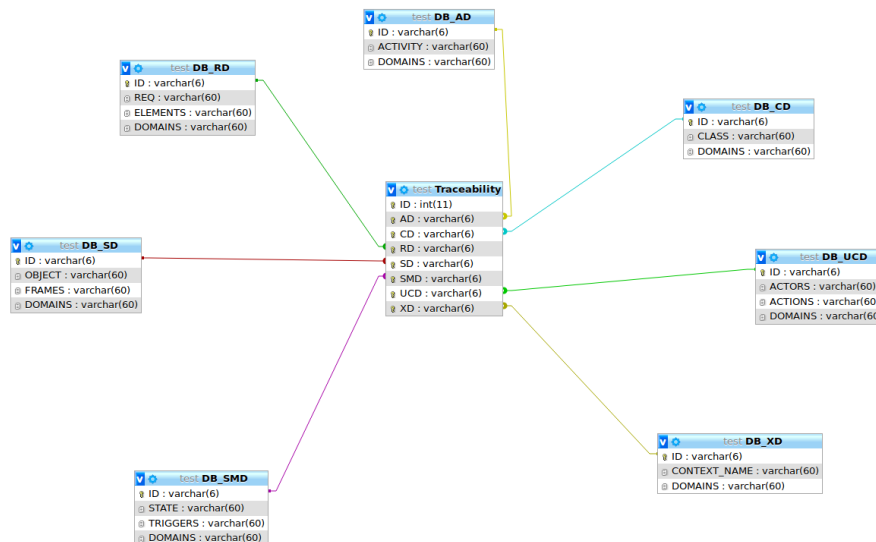


Figure 5.23: Database Chart

As shown in Figure 5.23, each diagram is stored in their table where the diagrams belonging to the same type are allocated. Each of them is connected by their ID to the traceability table. In each row of the traceability table, we have only the diagrams connected among them. For example, if we connect two diagrams, we will have only them represented in a single row. On the other hand, if we have a connection among

three or more diagrams, we will have a row containing three or more IDs. The missing diagrams are represented by “na” that means “Not Available”. For example, referring to the case proposed in Figure 5.22, we can see that there is a triple connection among RD1, UCD1, and CD1. In fact, they are all connected among them. RD1 is connected with both CD1 and UCD1. UCD1 is connected with CD1 and RD1, and, finally, CD1 is connected with RD1 and UCD1. One consideration is needed. Checking the diagrams UCD1, AD1, SMD1, and XD1, we can see that they are connected in a circle among them, but there is no direct connection between AD1 and XD1 or SMD1 and UCD1. For this reason, they cannot be represented all together in a single row. In Figure 5.24, we can see how the traceability table is populated according to the rules that we have mentioned earlier.

ID	1	AD	CD	RD	SD	SMD	UCD	XD
1	na	CD1	RD1	na	na	na	UCD1	na
2	AD1	na	na	na	na	na	UCD1	na
3	AD1	na	na	na	na	SMD1	na	na
4	na	na	na	na	na	na	UCD1	XD1
5	na	na	na	na	na	SMD1	na	XD1
6	na	na	na	SD1	na	na	na	XD1

Figure 5.24: Traceability Table

In Figure 5.24, we have represented all the diagrams (AD1, RD1, CD1, XD1, SD1, SMD1 and UCD1) related to our use case scenario. Now, we add other dummy diagrams in order to show how the traceability relationship works. These dummy diagrams are the following:

- Activity Diagrams: from AD2 to AD9;
- Requirement Diagrams: from RD2 to RD9;
- Class Diagrams: CD2 to CD5;
- Context Diagrams: XD2 to XD3;
- Sequence Diagrams: SD2 to SD9;

- State Machine Diagrams: SMD2 to SMD8
- Use Case Diagrams: UCD2 to UCD8

We do not represent the tables related to the single diagrams for space limitations. Anyhow, their connections are represented in order to show how the traceability database works, avoiding the loss of important information after deleting a connected diagram.

The databases related to the activity and requirement diagrams are represented in the following Figures 5.25 and 5.26.

Diagrams Table In Figure 5.25, we represent the AD table. The columns are three. The first column is related to the ID of the Activity Diagrams. The column ACTIVITY contains the AD name. Finally, the DOMAINS column presents all the domains related to AD. As we explained earlier, AD1 is the same diagram presented in Section 5.3.3, then we added other diagrams to show how the table is populated. The IDs are written using a sequence number, the activity names have a dummy definition, and the domains are randomly inserted.

Later, we will show how the traceability table is enriched with these dummy diagrams and how it works if an update or a deletion of these diagrams occurs.

ID	ACTIVITY	DOMAINS
AD1	Securing user data	Availability, Privacy, Security, Trust
AD2	dummy	Privacy, Trust
AD3	dummy2	Privacy, Trust, Usability
AD4	dummy3	Availability, Security, Trust, Usability
AD5	dummy4	Availability, Security, Usability
AD6	dummy5	Identity, Security, Usability
AD7	dummy6	Security, Trust, Usability
AD8	dummy7	Trust, Usability
AD9	dummy8	Trust

Figure 5.25: Activity Diagrams Table

Considering Figure 5.26, the column REQ contains the IDs related to the requirements represented in Section 5.3.6. These IDs are written following the TrUStAPIS

methodology [49]. Then, as for the other tables, there is a column populated with the RDs domains. Moreover, in this table, we represent eight dummy RDs that will be added to the traceability table.

ID	REQ	DOMAINS
RD1	pri_01, sec_01, trst_01	Privacy, Security, Trust
RD2	dummy_req	Availability, Trust
RD3	dummy_req2	Availability, Security, Trust
RD4	dummy_req3	Security, Trust
RD5	dummy_req4	Availability, Privacy, Security, Trust
RD6	dummy_req5	Availability, Security, Trust
RD7	dummy_req6	Availability, Trust
RD8	dummy_req7	Privacy, Security
RD9	dummy_req8	Privacy, Trust, Security

Figure 5.26: Requirement Diagrams Table

Some of the dummy diagrams are injected into the traceability table to simulate the connection among them. The new table is shown in Figure 5.27.

As we can see, the diagrams related to the first six rows of the traceability table are the same presented in the previous section and Figure 5.24. Moreover, we represent the dummy diagrams from rows seven to twenty. The diagrams not represented in the table are the ones not connected to the others. This means that they do not contain any information in common with other diagrams, and they can be deleted or modified without representing any domino effect issue for the other diagrams.

A graphical view of the diagrams' connections is shown in Figure 5.28. As we mentioned before, the diagrams represented here are the ones connected with at least another diagram.

Analyzing the structure of the diagrams, we can notice that there are four clusters.

On the left, we can see the same cluster presented in Figure 5.22. Then, we can see a small cluster composed of only two diagrams (AD8 and UCD3), and it is related to row 15 of the traceability table presented in Figure 5.27. The third cluster is composed of four diagrams. Three of them are connected in a circle, and they are represented in row number 7 of the traceability table. The fourth diagram is CD5, and it is only connected to AD3. Finally, on the right, we can see that there

ID	1	AD	CD	RD	SD	SMD	UCD	XD
1	na	CD1	RD1	na	na	na	UCD1	na
2	AD1	na	na	na	na	na	UCD1	na
3	AD1	na	na	na	na	SMD1	na	na
4	na	na	na	na	na	na	UCD1	XD1
5	na	na	na	na	na	SMD1	na	XD1
6	na	na	na	SD1	na	na	na	XD1
7	AD3	CD4	na	na	na	SMD7	na	na
8	na	CD3	RD8	na	na	na	UCD2	XD3
9	AD2	na	na	SD4	na	na	na	na
10	AD3	CD5	na	na	na	na	na	na
11	AD5	CD2	RD7	na	na	na	na	na
12	na	na	na	SD4	SMD3	na	na	XD2
13	na	na	na	na	na	SMD4	na	XD2
14	AD6	na	na	na	na	SMD4	na	XD3
15	AD8	na	na	na	na	na	UCD3	na
16	na	na	RD4	SD5	na	na	UCD2	na
17	na	na	na	na	na	na	UCD7	XD3
18	na	CD2	RD4	na	na	SMD4	na	na
19	na	na	RD3	na	na	SMD4	na	na
20	na	CD2	na	SD5	na	na	na	na

Figure 5.27: Traceability Table (Extended)

is a fourth cluster. Considering all the elements, we can state that one of the most connected diagrams is SMD4. In fact, if we make the following query in the extended traceability table, we can check which diagrams are directly connected with it:

```
SELECT * FROM 'Traceability' WHERE SMD = "SMD4"
```

The result is shown in Figure 5.29, and it means that SMD4 is connected to AD6, CD3, RD3, RD4, XD2, and XD3. In fact, if we try to cancel SMD4, we will receive an error message telling us that it is not possible to cancel the diagram because of existing external references.

It is possible to cancel or modify the diagram only after relaxing the existing connections. This is a powerful measure that avoids domino effects after the deletion

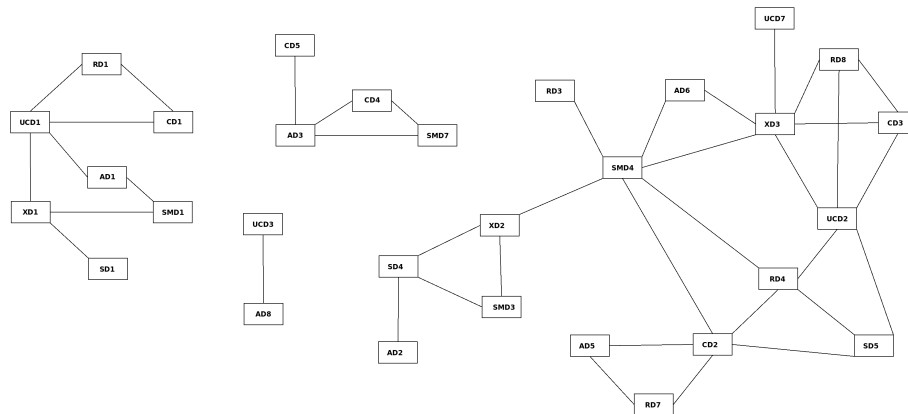


Figure 5.28: Graphical view of the Traceability Table (Extended)

ID	AD	CD	RD	SD	SMD	UCD	XD
13	na	na	na	na	SMD4	na	XD2
14	AD6	na	na	na	SMD4	na	XD3
18	na	CD2	RD4	na	SMD4	na	na
19	na	na	RD3	na	SMD4	na	na

Figure 5.29: Traceability Table related to SMD4

of important pieces of data. After this step, the modeling phase is concluded, and, in case no further actions are needed (i.e., modify a model or delete it), it is possible to perform the GR. If it ends with a positive decision, it is possible to proceed to the following phase of the K-Model: the development phase.

5.4 Development

During the development phase, the SCM is built following the previous phases and the context.

We need to develop the IoT entity (SCM) considering its tasks (i.e., bake a cake) and the interaction with other IoT entities: a smart fridge in the same smart home and smart supermarkets belonging to the smart city environment. These connections are useful to check and order a particular ingredient if it was needed for a recipe. Moreover, the IoT entity must allow trusted users to interact with it and deny the

interaction for the untrusted users.

In this section, we show how it is possible to apply the approaches proposed in Section 3.5 in order to develop the desired IoT entity. In any case, it is the developer's task to choose which contexts and functionalities must be considered in order to develop the IoT entity according to the previous phases of the K-Model.

5.4.1 Top-Down approach

According to this approach, we need to follow a descending path starting from considering the IoT entity as a whole and then going deep into the functionalities.

Analyzing Section 5.2, we can state that some of the elicited requirements were related to the access control mechanisms. Furthermore, other requirements are necessary for the baking functionalities of the IoT entity.

Figure 5.30 presents the FDBS related to the Smart Cake Machine. Thus, we have the IoT entity on the top level. Then, on the second level, we have two general functionalities: access control and baking functionalities.

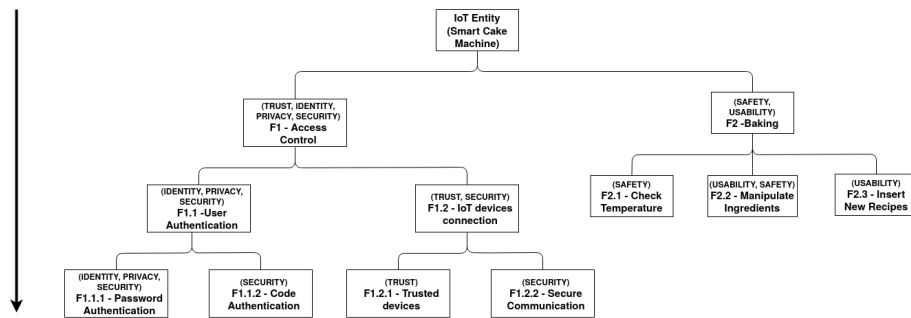


Figure 5.30: FDBS - Use Case: Smart Cake Machine

Considering access control, the related functionalities are divided into two fundamental parts: user authentication and IoT device connections. In fact, SCM can interact with trusted users and IoT devices. Concerning the users and according to Section 5.2, we can consider two authentication types: password and code authentication. Password authentication has indeed been discharged in favor of code authentication. However, it is possible to consider both of them in this phase, in the case stakeholders would like to implement both of them. However, the first one

is related to the domains of identity, security, and privacy. In fact, a password is related to a single user, and it must be stored securely. Moreover, the data of the users must be collected and kept private. On the other hand, the code authentication can be considered without storing user information, and it can be shared with other users who are trusted by the owner of the IoT device. Moreover, this code must be securely provided.

Regarding the connections among the Smart Cake Machine and the other devices, we need to consider the trusted devices guaranteeing that the communication among them is secure. So, the functionalities needed to create trust models for the devices will belong to the trust domain. The communication among these devices will belong to the security domain.

It is important to note that any functionalities belonging to this part of the “tree” is connected to the main functionality “Access Control”. They are fundamental to grant device access only to those who are trusted and can provide the credentials to interact with the device.

Figure 5.30 shows that the right part of the tree is related to the baking functionalities. In order to bake a cake, there are three fundamental functionalities. The first one is to check the temperature. It is a function for the correct preparation of the cake and concerning the safety of users and devices. The second one is related to the manipulation of the ingredients. This functionality is related to the safety and usability domain. In fact, safety is also connected to health aspects. In this case, usability and safety domains are related to the correct utilization of the machine manipulating the ingredients. Finally, the third functionality is related to the possibility of inserting new recipes. In this case, it is essential for the usability domain because the interface must be user-friendly. As we can see, the domains are collected and separated according to different functionalities.

5.4.2 Bottom-Up approach

This approach is useful to define the different contexts according to the Smart Cake Machine use case.

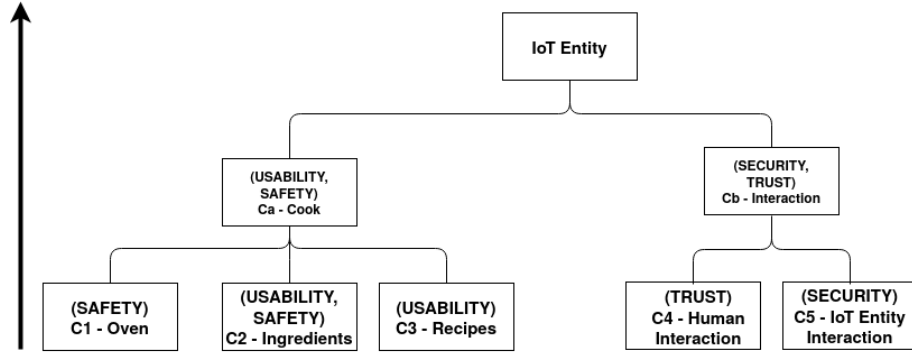


Figure 5.31: Bottom-Up Approach - Use Case: Smart Cake Machine

We can see the proposed contexts in Figure 5.31. We describe the contexts from right to left.

One context is related to smart entities' interaction (i.e., smart supermarket, smart fridge). Then, another context is related to the utilization of the device by human users. These two contexts can be considered together in a super-context related to the interactions. Trust and security are fundamental for these interactions, and they are appropriately considered as the primary domains for these contexts. Then, we have a third context related to the recipes, another one about the ingredients, and a fifth context related to the oven. These three contexts can be summarized into a super-context named cook context where the considered domains are usability and safety. The two super-context are fundamentals for the trusted IoT Entity.

5.4.3 Block Development

After utilizing the bottom-up and top-down approaches, the developer must create the code according to them and to the previous phases of the K-Model.

In our case, we start from the definitions of the contexts and functionalities, creating separate code blocks according to them. We can see that contexts C1, C2, and C3 proposed in the bottom-up approach are also covered in the top-down approach with the functionalities F2.1, F2.2, and F2.3. In fact, it is possible to consider the same or similar aspects of both approaches.

Thus, we will present the code block containing these highlighted aspects. For

the first context and functionality related to the oven and the temperature, we need to set temperature attributes in order to fix predetermined levels related to different states during the cooking process. Then, for the ingredients, we need to consider them in order to be cataloged and inserted by the users. Finally, there are the recipes. We assume that they must be uploaded by the users or memorized in the device by the vendors.

The domains are the same considered in the previous approaches (i.e., safety, usability), and they are declared at the start of the code blocks. We want to remind the reader that these domains are strongly connected to trust, and their consideration allows developers to implement trust in the IoT entity.

```

Block "Cook"

Code Oven (Safety):
    attr temp;
    bool light;
    setTemp(temp);
    getTemp();
    setLight(light);
    getLight();

Code Ingredients (Safety, Usability):
    attr[] ingr&qty;
    attr qty;
    attr ingr;
    setIngr&qty(ingr,qty);
    getIngr();
    getIngr&qty();

Code Recipe (Usability):
    attr process;
    attr author;
    attr[] neededIngrs;
    attr[] availableIngrs = Ingredients.ingrs;
    setProcess();
    getProcess();
    setAuthor();
    getAuthor();
    setNeededIngrs();
    checkIngrs();
  
```

Figure 5.32: Block Development (BD) - Block Cook

In our example, we can see in Figure 5.32 that the block named *Cook* is composed of three parts: Oven, Ingredients and Recipe. We use the generic terminology *attr* (i.e., attribute) to consider characters, strings, or numbers (i.e., integers, doubles, floats). Moreover, we use a Boolean variable and arrays of attributes. Then, we

create methods useful to manipulate and check the attributes. For example, the attribute *setIngrs&qty(ingr,qty)* is fundamental to set which ingredient and its amount is considered. Another interesting method is *checkIngrs()*. It can be used to compare if the needed ingredients are available. We do not deeply specify the methods, but we only declare them.

5.4.4 Metrics

In the development phase, it is very important to consider metrics useful to organize the IoT entity's behaviour.

In joint work with British Telecom, we have proposed a trust metric to evaluate the interaction among entities and solve the differences among the different trust models developed by different devices and vendors (i.e., Amazon, Google). In order to solve these differences, we have proposed a trust model implementing a straightforward trust metric, allowing each user to interact with the devices according to a determined trust value. This implementation can be generalized to other IoT devices such as the SCM.

5. 4. 4. 1 Trust Metric

We can have three different trust levels: high, medium, and low. The higher level allows users to control the device. The medium level enables the user only to check the device status or activity, but it does not allow any control of the device. Finally, the lower level certifies that the user is not rusted. For this reason, any connection is refused.

Going deeply into the trust metric analysis, it is composed of three main parameters and two sub-parameters. The formers are a role for each actor, a score, and context values. The latter is composed of a context index and the functionality related to the device.

The device owner must have the ability to remove or delimit the actions that another actor could perform. This is possible by giving them a score value related to a particular context of the actions.

All of these parameters are represented in our trust model by the implementation of the trust metric for each user.

The trust metric is used to define rules for each actor, and the following function represents it:

$$Trust_Metric_x : TM(R, C(DF, XI), Sc)$$

where the function $Trust_Metric_x \in \mathbb{R}$ and its parameters are:

1. **Role (R).** The role of the actors involved can be different. Anyhow, we consider some of them as always present. The Home Owner (HO) is the one that owns the smart home and the IoT devices in it. A House Member (HM) is another actor living in the house. Then, a House Guest (HG) is someone staying in the smart home for a limited amount of time. Then, a Malicious User is an actor that does not belong to any of the other categories.
2. **Context (C).** It is related to the device or functionalities and its or their importance perceived by the HO.
 - (a) **Device/Functionality (DF).** A device may have one or more functionalities. According to them and to the user involved, it is necessary to set the following parameters for each of them. We define them with natural numbers for both the device and the functionalities. Thus, we can have the device number 1 and 2 with several functionalities. For example, to represent the second functionality of the first device, we will define the parameter DF as 1.2.
 - (b) **Context Importance (XI).** It is related to the importance of the context according to the HO. It is represented by a number given by the HO. The higher the context, the higher the score or role needed. This value belongs to the following set: $C \{1,2,3,4\}$. The lower the value, the less important is C.
3. **Score (Sc).** It is the rank given to the users by the HO. It is similar to a reputation value. The more trusted the user is, the higher the score given. It belongs to the following set: $Sc \{0,1,2,3,4,5\}$.

Regarding the roles, the HO is allowed full control of the device regardless of Sc and XI, given that she is fully trusted. For a MU, the metric works in the opposite way since he is not allowed to control or check anything. In the case an HG or an HM turns into malicious, he will be treated as an MU and basically “banned”. For this reason, we can state that this model wants to reach two goals. The first one is to prevent any activities from an external user (i.e., MU). Then, the second goal is to be able to avoid attacks from internal users; if they happened, the model treats the internal users as external ones preventing them from continuing to use the device. Even for this role, XI and Sc are optional. On the contrary, for the other roles (HM and HG), XI and Sc are fundamental. The metric is straightforward, and it can be easily performed by any IoT device, even considering a limited computational power [96, 134]. It computes a value that will be used to check which actions are allowed for a particular user and for a particular context. It is basically a subtraction of the value Sc with respect to the value XI. If the result is positive, the trust value is ranked as high. If the result is zero, the trust level is medium. Otherwise, the trust level is low. If a XI parameter has a value of 1, it means that it is not crucial for the owner (i.e., check a recipe). On the other hand, if a XI parameter has a value of 4, it is crucial (i.e., to order missing ingredients). Therefore, we can say that if an HM has a score of 5, he or she is similar to an HO, considering that with this score value, it is possible to control everything, whatever XI is:

$$Trust_Metric_1 : TM(HM, C(DF, XI), 5) > 0$$

Conversely, if a user has a score of 0, he or she is considered as an MU and it is not possible to perform any actions (no matter the value of XI):

$$Trust_Metric_2 : TM(MU, C(DF, XI), 0) < 0$$

We chose these values according to the explanation of the trust metric parameters given earlier. The score must have a more significant bound to include the role MU (score = 0) and the role HO (score = 5). For an MU, no matter the context, it must be impossible to perform actions. On the contrary, for an HO, everything must be permitted. The values from 1 to 4 are used both for XI and Sc to define the

boundaries related to the other users (i.e., HM and HG). Using these roles, we cover all the possible actors, and for each of them, the scores could be different also for the same context.

According to the authentication process that we have identified in the previous phases (i.e., code authentication), it will be created according to the trust metric implementation. Thus, there will be a code for each user according to the trust metric computation. Anyhow, in the case of an MU, no code is provided, and this will not allow the user to be trusted and authenticated.

5.5 Verification and Validation

During the verification phase, the SCM is tested about its functionalities. Then, the SCM must be validated in its intended environment.

5.5.1 Verification

In this phase, we need to compare the model documentation and the elicited requirements according to the developed functionalities. Secondly, tests are performed in order to check that the functionalities work as expected. Then, traceability must be respected in order to connect requirements and functionalities. We follow the step-by-step methodology proposed in 3.6.1 presenting an example of how to verify functionalities and requirements.

We present how to verify the model documentation and the requirements.

5. 5. 1. 1 Step 0

The first step is to compare the model documentation and the elicited requirements.

In Sections 5.2 and 5.3, we have proposed several requirements and models. In order to show how to perform the tasks in this step, we will consider only a subset of them.

Table 5.12: Requirements and model connections.

Domain	Requirement	Model
Usability	USAB01 - The user shall be able to insert new recipes	SMD1 : Upload a recipe into the SCM
Privacy	PRIV03 - User data shall be kept private	AD1 : Securing user data
Security	SEC02 - The SCM shall delegate the Smart Hub to order the missing ingredients	SD1 : User, SCM, SF, SH and SM interaction
Identity	IDNT02 - The user shall provide his/her data in order to be registered	UCD1 : User data
Trust	TRST01 - The SCM shall trust a Smart Supermarket with a trust level above 0.5	CD1 : SCM and SM service classes

In Table 5.12, we show this subset and its connection with the models. However, we will show only how to verify the requirement USAB01 according to the modeled SMD1 diagram (it was presented in Figure 5.19). Moreover, in Figure 5.33, we show again the JSON related to USAB01. In fact, it is useful to speed up the verification process by checking if the goal and the characteristics are met.

```

17-  "IoT_requirement_USAB01" : {
18-    "Context" : {
19-      "Domain" : [{
20-        "Usability" : {
21-          "Characteristic" : ["Simplicity, Understandability"] } }],
22-      "Environment" : "Smart Home",
23-      "Scope" : "User Interface" } ,
24-    "Actor" : {
25-      "Role" : "User",
26-      "Type" : ["Human User"] },
27-    "Action" : {
28-      "Type" : ["Fulfill"] ,
29-      "Measure" : [" " ]},
30-    "Goal" : "Let the user insert new recipes"
31-  },

```

Figure 5.33: JSON code for USAB01

5. 5. 1. 2 Step 1

After the collection of the requirements and the models performed in the previous step, the developers must perform the functionality tests.

In this case, it is necessary to verify different aspects. In the first instance, only a trusted user can access the SCM. In fact, only after this important step, it is possible to insert a new recipe. Then, if the first control is positive, the functionality tests must check if the recipes are correctly uploaded. SMD1 was designed in the following way. Firstly, for any new recipe, it was mandatory to insert a new title. Secondly, the items and their quantity. The recipe upload ends when all the items have been inserted. Thus, the test must check if all the items are correctly inserted. If not, the test fails, and the code must be checked in order to find the error. Otherwise, the functionality test passes, and it is possible to proceed to the following step.

5. 5. 1. 3 Step 2

This step is a second check that guarantees that the requirements are met and the functionalities have been correctly implemented. Traceability guarantees this aspect as we have seen in Table 5.12.

In the case the previous step has ended correctly, it is possible to confirm that the requirement USAB01 and its derived functionalities have been correctly implemented. On the other hand, it will be raised an issue that will be checked in order to adjust the wrong functionality.

After all the functionalities have been checked positively or negatively, it is possible to proceed to the final step of the verification process.

5. 5. 1. 4 Step 3

The verification response is positive if all the previous steps have been correctly performed. Thus, a document certifying which tests have been positively or negatively performed will be the output of this phase. In the case some tests have been negatively performed, there will be a modification of the functionalities, and a new verification process will be performed after the needed modifications.

5.5.2 Validation

In this phase, the need documentation and the elicited requirements are collected and compared according to the developed functionalities executed in their intended environment. The functionality tests are performed in the first step, and traceability is required in the second step. The validation response, either positive or negative, is the output of the final step. We present the step-by-step methodology proposed in 3.6.2 according to our proposed scenario.

5. 5. 2. 1 Step 0

According to 5.1, we collect a subset of the originating needs:

1. **Need 1:** The temperature of the SCM must be checked and it could not overcome 250°C.
2. **Need 2:** The recipes must be downloadable from the vendor website or inserted manually by authenticated users. Authentication must be done by code.
3. **Need 3:** The SCM could interact with a Smart Fridge (SF) to check whether a particular ingredient is in the fridge or not. If not, the SCM could interact with a trusted Smart Supermarket (SM) through the home Smart Hub (SH) and order the missing ingredient.
4. **Need 4:** The communication among the smart home entities must be guaranteed and encrypted.

Then, a subset of the requirements elicited in 5.2 are shown in Table 5.13. Moreover, in the third column, we can see the need connected to the requirement. We can see that there are needs coded by an X. These needs are not strictly connected to the defined needs, but they can be necessary in order to develop the entity according to the defined context.

For the validation test, we will consider the requirement TRST01 and Need number 3. The JSON code related to TRST01 is presented in Figure 5.34.

Table 5.13: Requirements and needs connections.

Trust Req.	TRST01 - The SCM shall trust a Smart Supermarket with a trust level above 0.5	Need 3
Usability Req.	USAB01 - The user shall be able to insert new recipes	Need 2
Security Req.	SEC01 - The user shall be authenticated	Need 2
Security Req.	SEC02 - The SCM shall delegate the Smart Hub to order the missing ingredients	Need X
Availability Req.	AVBT01 - The SCM shall be able to connect to the Smart Hub	Need X
Privacy Req.	PRIV01 - The SCM shall perform an encrypted communication with the Smart Fridge	Need 4
Identity Req.	IDNT01 - The user shall be authenticated	Need 2
Identity Req.	IDNT01.2 - The user shall be authenticated by code	Need 2
Safety Req.	SFT01 - The SCM shall be able to check its temperature level	Need 1
Safety Req.	SFT01.1 - The SCM temperature level shall be lower than 25°C	Need 1

```

"IoT_requirement_TRST01" : {
  "Context" : {
    "Domain" : [{
      "Trust" : {
        "Characteristic" : ["Direct", "Indirect", "Global", "General",
          "Measurable"] } }],
    "Environment" : "Smart City",
    "Scope" : "Establish trust between SCM and SM" } ,
  "Actor" : {
    "Role" : ["Trustor, Trustee"],
    "Type" : ["SCM", "SM"] },
  "Action" : {
    "Type" : ["Fulfill"] ,
    "Measure" : ["Trust level"]},
  "Goal" : "To fix a trust value"
},

```

Figure 5.34: JSON code for TRST01

5. 5. 2. 2 Step 1

As we will present in the following and final phase of the K-Model, an algorithm is needed to introduce the new IoT entity in a smart home context. Anyhow, in this case, we need to focus only on the functionalities that must be validated. In this case, it is related to the interaction between the SCM and a trusted SM. The trust value is provided by the SH or from the user (i.e., because the user has a direct past relationship with an SM). The value can be computed after considering multiple parameters (i.e., cost, distance, quality), and the service is satisfied if any interaction among SCM and the SM is performed if and only if the computed trust value is more than 0.5. If the interaction is performed at a lower level, the validation test fails. Otherwise, it succeeds.

5. 5. 2. 3 Step 2

This step is fundamental in order to certify that the tested functionalities satisfy the requirements and fulfil the originating need. Traceability is guaranteed also following Table 5.13.

If the previous step ends correctly, it confirms that the originating need is met. Otherwise, an issue is raised, and the developers will need to check and correct it.

After all the functionalities have been checked in their environment and compared with the originating needs, it is possible to continue to the final step of the validation process.

5. 5. 2. 4 Step 3

The validation response is positive if all the previous steps have been performed positively. Thus, documentation is produced to certify that the stakeholders' needs have been met, and it is possible to proceed to the K-Model's final phase (i.e., Utilization). Otherwise, in the case some validation tests have failed, it will be produced documentation that will highlight the functionalities that have failed the originating needs to be checked by the developers that will need to fix the raised issues.

5.6 Utilization

Once the SCM has been correctly validated, and every issue that has been possibly raised has been fixed, it is possible to sell it, satisfying the stakeholders' originating needs (i.e., vendors and customers).

Anyhow, in this phase, the SCM will be placed in its intended environment and context (i.e., a smart home), and it will join other smart devices (i.e., if a trusted SF is also present in the smart home, the SCM will interact with it checking the needed ingredients; otherwise it can check a trusted supermarket to buy them).

Anyhow, as we presented earlier, it is essential to consider also the architecture where the SCM will be used and its relationship with the other IoT devices populating it.

Thus, we show a possible smart home architecture in Figure 5.35, where the SCM will be placed. On the right side of the figure, there are marked the levels related to Figure 5.2. Moreover, as required in Section 5.1, it is composed of two networks:

one internal and another external.

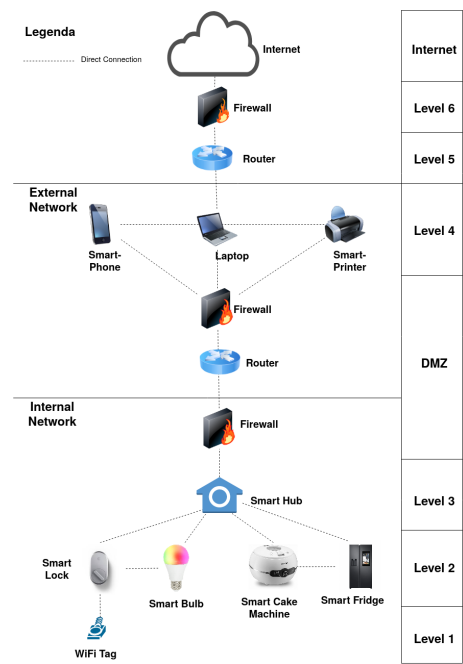


Figure 5.35: Smart Home: Segregated Trust Architecture

The internal network is composed of a smart-lock, an SCM, a smart-bulb, and an SF. These entities are only allowed to communicate with the other entities according to their purpose. The dotted lines represent a direct link for the communication. Thus, the smart-lock has no connections with the SCM or the SF because there is no reason for them to communicate directly with each other. On the contrary, the smart-bulb can interact with the smart-lock. In fact, when the door is opened, the smart-lock can send a signal to the smart-bulb to switch the lights on (in the case it is night). Then, as we have planned writing the scenario, the SCM and the SF can interact.

However, all these devices have direct communication with the smart hub, which monitors their activities according to the trust model presented in Section 3.7, allowing them to communicate directly only for determined purposes.

Moving up in the network, we can see that the internal network is separated from the external one that contains a smartphone, a laptop, and a smart-printer. These three objects belong to the BYOD paradigm. They cannot be placed in the internal

network because they can join other networks and can be compromised.

5.6.1 Join

However, in order to show how the trust model work, let us assume that the homeowner needs a new smart-lock, our developed SCM, and a smartphone.

5. 6. 1. 1 *Smart-Lock*

When the smart-lock joins the network for the first time, a broadcast message is sent to all the network devices. The smart devices are not allowed to answer because they do not recognize the ID of the new device. On the other hand, the smart hub recognizes the new device as belonging to the homeowner. In fact, we assume that before joining a network, the smart homeowner validates the devices. Thus, after this first message, the smart hub starts the trust computation following the trust model proposed in Figure 3.24. Thus, the smart hub checks the reputation DB in order to control if the smart-lock has previously been connected to the network, but we assume that the device is entirely new. Then, the threat DB is checked to find out if there are known vulnerabilities related to the smart-lock model finding a known vulnerability. The risk calculation takes into consideration the parameters L, S, and D. About L, the smart hub assigns a *medium* value because the known vulnerability can be exploited. Regarding the parameter S, the smart hub decides to give a *high* value because if the vulnerability is exploited, the smart-lock will completely lose its functionalities. Finally, the D value is *low* because the vendor has designed the smart-lock to provide feedback on its functionality. Therefore, we have a high value for S (9), a medium value for L (3), and a low value for D (1). According to these values, the overall risk estimation, as presented in Section 4.7 is medium (27). After this step, the smart hub checks the contexts of the device. Firstly, the context is related to the smart lock cooperation with the smart bulb. Thus, in the case of malicious behaviour or possible malfunctions related to a hijack of the device, the smart bulb can also be affected. In addition, the smart lock is critical for the smart home environment because, in the case of malicious behaviour, it can allow strangers

to enter the house, or it can keep the homeowner outside.

So, the trust estimation ends after considering the following parameters: the risk value is medium. The context refers to the cooperation with the smart-bulb and access to the house. Finally, a known vulnerability is found in the threat DB. Considering all these parameters, the trust estimation output is not to allow the smart-lock to join the network. This action protects the internal entities to be threatened by the new device and guarantees the trust level in the internal network.

5. 6. 1. 2 SCM

The second device bought by the homeowner is our SCM. As for the smart lock, it sends a broadcast message after joining the network for the first time, and the smart hub starts the trust computation. Checking the reputation DB, it finds nothing because the device is all new. Then, the threats DB shows that there are no known issues. The risk calculation gives L, S, and D the minimum value, so the final value is low (1). The context is considered according to the collaboration with the smart fridge and the communication with the smart hub in the case the SCM needs to interact with an SM. Moreover, the context cook is not dangerous for the security and safety of the house according to the security and safety requirements elicited and satisfied. For this reason, the trust estimation ends with success, and the SCM can join the internal network.

5. 6. 1. 3 Smartphone

In order to cover also the external network part, we present a use case with a third device: a smartphone.

This new device has never joined the network earlier. Thus the reputation DB has no data for it (in the case a device has previously joined the network and it has been banned due to a low reputation level, it cannot join the network again). Then, the threat DB has no known attacks related to the smartphone model and version. The risk value is calculated as low (3) because the L and D parameters are considered low (1), and the S parameter is considered medium (3) because, in

the case of malfunction or malicious activity, the network can be partially damaged. The context is related to all the devices belonging to the external network and the smart hub of the internal network. Finally, the smartphone belongs to the BYOD paradigm. Thus in the case of acceptance, it will be allowed to join the external network.

After considering all these parameters, the smartphone is allowed to join the external network, and its behaviour is monitored to anticipate possible threats.

5.6.2 Stay

We will see now for the SCM and the Smartphone what can happen after they have been accepted in the network.

5. 6. 2. 1 *Smartphone*

Let us assume that after a few weeks, the smartphone has been manipulated by a malicious entity and tries to communicate with the other smart entities to take their control. The architecture allows the smartphone to pass through the smart hub to communicate with the smart entities in the internal network. We assume that the smartphone repeatedly sends a command to the smart-bulb to switch the lights on and off every five seconds. The smart hub catches this abnormal behaviour and, by using the adaptive model, decides to block the communications belonging to the smartphone and set it in quarantine. The smart hub, checking the threat DB, recognises that the smartphone has carried out a replay attack. The reputation DB is set with a low value, and the smart home owner is notified of the event.

5. 6. 2. 2 *SCM*

Let us assume that the SCM communicates with the SF in order to check the ingredients for a requested cake, and the milk is missing. The SCM checks the trusted SM list and sends the request to order the missing ingredient to the smart hub. The smart hub checks the behaviour of the SCM and forwards the communication to

the SM as we modeled in Figure 5.18. After the interactions, the SH update the reputation DB with a high value. In fact, the interaction is recognized as genuine, and legitimate behaviour has positive consequences for the trust level of our SCM.

5.6.3 Threat Analysis

Besides organizing the relationships among IoT entities, our segregated trust architecture has been built to protect them from known IoT attacks. Thus, in this section, we provide a review of known IoT attacks and possible mitigations related to our segregated trust architecture.

In our model, the knowledge of such attacks is a focus point, mostly to decide if a new entity can join the network and in which level as shown in Figure 3.24 or to decide if an entity can stay in the internal or the external network as shown in Figure 3.26.

In “Security and Privacy in the IoT” [71] there are collections of known attacks. These attacks and other known attacks can be prevented by implementing our architecture. Listed below, we show these attacks and their possible mitigation offered by our segregated trust architecture.

- **External attack.** This is an attack that does not belong directly to the network, according to Hu et al. [71] this can be provided with the cloud that can share the information with external entities. In our approach, we avoid sharing private information with the external network because it is stored in the internal architecture.
- **Wormhole attack.** This attack depends on a malicious IoT entity that can be part of the network that throws all the information passing through this entity. In our approach, this behaviour is avoided because all the entities can do a specific set of things, and the smart hub traces the communications.
- **Sinkhole attack.** The malicious node attracts information and communications from the other nodes. In this case, the architecture prevents this behaviour because a node can attract only a subset of information depending on

its scope.

- **Witch attack.** A malicious node can take advantage of another node's failure taking all the communications related to the failed node. Our architecture prevents this attack because it is not possible for a node to communicate with another node unless this action was allowed by the central hub.
- **HELLO flood attack.** With this attack, a malicious node can flood the network with a HELLO to nodes that are not neighbours behaving as a neighbour. This attack cannot take place with our architecture because each entity can communicate only with a subset of entities thanks to the join procedure.
- **Flash crowd.** With a flash crowd, there is a rapid increase in traffic to a specific website on the Internet. Our architecture prevents this behaviour because the entities are not allowed to contact the Internet without passing through the central hub. Besides, the entity cannot contact a website outside its scope.
- **Distributed denial of service (DDoS).** As for the previous attack, with our architecture, because the communication among the internal and external network is controlled by the Smart Hub, this attack can be avoided. In fact, if a malicious IoT entity belonging to the internal network attempts to reach the external network or the Internet for malicious purposes, the adaptive trust model would be triggered.
- **Botnet.** Because the architecture prevents DDoS attacks; this configuration equally prevents the network from becoming a botnet controlled by an external malicious entity to make it perform malicious actions.
- **Eavesdropping.** With an eavesdropping attack, it is possible to intercept traffic between entities, as a man in the middle attack. With this architecture, communication is allowed between entities only if the central hub certified it, so another entity cannot intercept the communication of other entities.
- **Replay attack.** The replay attack is performed by a malicious entity to repeat an instruction causing malfunctions or harm. In our architecture, the activities

are monitored by the smart hub through the adaptive trust model. Thus, in the case of repeated, unexpected instructions, the SH applies countermeasures (i.e., ban the entity or put it in quarantine).

- **Blackhole.** As stated by Hu et al. [72], the black hole attack permits an attacker to drop receiving routing messages instead of relaying them. This attack cannot be performed in our proposed architecture because of the join, stay, and leave implementations as for the adaptive trust model.
- **Brute force attack.** Without a limited number of attempts to access a resource, it is possible to try a brute force attack. This can be dangerous for weak or medium passwords. The smart hub monitors these activities in our architecture. For this reason, a brute force attack would be discovered.
- **Whitewashing attacks.** This attack is performed when an object leaves and rejoins a network. The adaptive trust model prevents this behaviour.
- **On-Off Attacks.** This attack occurs when an entity performs a good or bad service randomly. In our architecture, the adaptive trust model can recognize this behaviour and take corrective actions.

5.7 Conclusion

In this chapter, we have presented a use case scenario implementing our trust-by-design framework. In the first phase, we have presented the needs related to the SCM. Then, in the requirements phase, we have elicited the proper requirements following the TrUStAPIS methodology. Therefore, we have developed the models following our trusted model-driven approach in order to design the functionalities and relationships among our SCM and the other IoT entities. Furthermore, we presented the development phases implementing both the top-down approach considering the functionalities of the SCM and the bottom-up approach about the contexts. Moreover, we have developed a block of code following the trusted finite state development technique. We have then illustrated how to verify and validate several models, requirements, and needs in our use case scenario. Finally, we have illustrated what happens when the developed SCM is placed in an IoT network considering its relationships and other IoT entities. For each phase, we have also considered the related transversal activities.

CHAPTER 6

Conclusions, Future Works and Future Lines of Research

In this final chapter, we will present our conclusions about this thesis work, discussing our aim and which benefits we have brought to the state of the art, filling the gap that we have previously identified. Then, we present the future works that we have planned to perform in order to continue and improve this thesis work. Moreover, we present some future lines of research that remain open and will need extra effort in the future in order to solve issues that are still open.

6.1 Conclusions

In this thesis, we have proposed a trust-by-design framework to consider trust and related domains during the SDLC of an IoT entity because we believe that there is still little consideration of trust in the system and software engineering community.

Our aim with this thesis work is to fill this gap and provide insights on trust concepts connecting it to essential and related domains such as privacy and security. This is most important in an environment such as the IoT which brought new possibilities but also security challenges. For this reason, guaranteeing trust in IoT has become a critical task that we want to mitigate with our work. In fact, we believe that in order to guarantee trust, it must be fundamental to consider it in the whole SDLC and not only when the IoT device has been developed and used by the customers.

For this reason, our framework ensures that trust and other domains are considered during the whole SDLC since the earliest phases to the utilization of the IoT devices. The framework comprises a K-Model and seven transversal activities: documentation, traceability, risk management, threat modeling, decision making, metrics, and gates. The K-Model has seven phases, plus the context that is always present. This framework allows developers to build the right trusted IoT product reflecting what the users and vendors need thanks to their consideration in the first phase of the SDLC.

Moreover, in this phase, it is essential to have clear in mind the architecture in which the intended IoT entity will be used. Furthermore, all the stakeholders give their input about their specific needs collected by the developers, which will elicit the proper requirements in the second phase of the SDLC according to these needs. In order to consider properly trust and its connected domains, we define seven types of requirements (i.e., trust, usability, security, availability, privacy, identity, and safety), proposing a JSON-based requirement elicitation method: the TrUStAPIS approach.

Following this approach, the developers can elicit the proper requirements according to the stakeholder needs identified in the previous phase. Therefore, for every

domain, we have highlighted a set of characteristics that must be taken into consideration in eliciting the requirements. Another important feature of the method is traceability that enables the connections among requirements and provides a holistic view of the IoT entity under development. Moreover, traceability guarantees control, avoiding domino effects in the case of relaxing requirements. After this phase is completed and, in the case of conflict among requirements, they have been solved following the POM methodology, it is possible to proceed to the third phase of the framework: the model phase.

In order to provide developers with a proper tool in designing the models, we have proposed a model-driven approach that considers all the trust-related domains. About trust, we consider the distinction between evaluation and decision models in order to model the proper features related to trust stereotypes. In addition, we enhance and extend UML and SysML diagrams proposing new ones and adding new features and stereotypes. The enhanced models are the use case diagram, the class diagram, the sequence diagram, the activity diagram, the state machine diagram, and the requirement diagram. The new diagrams are the context and the traceability diagram. The former helps developers highlight each possible context and its related domains to consider the different functionalities belonging to an IoT entity. The latter is needed to control the connection among the other diagrams and helps developers avoid domino effects due to the modification of diagrams connected to others.

After the model phase, there is the development phase where the developers will examine the documentation produced in the previous phases in order to write the proper code that will define the behaviour of the IoT entity. To suggest a systematic way to complete this phase, we propose two approaches to implement functionalities and contexts. About the former, we have proposed a top-down approach (FDBS) that is useful to specify domains and functionalities belonging to the IoT entity under development. About the latter, we have presented a bottom-up approach considering domains and contexts belonging to the IoT entity under development. Here, we need to start from the single contexts aggregating them according to their scope under super-contexts. Finally, all the contexts will compound the IoT entity as its whole. As for the top-down approach, it is represented as a tree. Besides, to merge

and consider together the top-down and bottom-up approaches, we present a block development. This development style is useful for the developers in order to consider all particularities of contexts and functionalities in single blocks of code according to contexts and functionalities. These three approaches are useful for developing the trusted IoT entity according to the documents produced in the previous phases of the K-Model and the architecture where it will be used.

After the development phase is completed, we have the verification phase. It is useful to guarantee that all the functionalities are correct, the domains (i.e., trust, security) are well considered, and the IoT entity has been built right. Basically, in this phase, the developers will verify that the functionalities reflect the requirements and the models performing verification tests. The JSON code written in the requirements elicitation process can be useful to automatize the process. In the case some verification test will not be successfully completed, a modification of the functionalities will be performed. Otherwise, it is possible to proceed to the following phase.

During the validation phase, the developers will check that the right entity has been built, analyzing how the developed IoT entity performs in its intended environment. Here, the originating needs are considered to check if the IoT entity fulfils them and the elicited requirements are satisfied. If not, the functionalities must be changed to reflect the needs. On the contrary, if the validation tests end correctly, it is possible to sell the device for its intended use. This leads to the final phase of the SDLC: the utilization phase.

In this phase, the IoT entity will interact with other IoT entities and be placed in its intended network architecture (i.e., a smart home). However, in order to guarantee trust among the IoT entities, we have proposed a segregated trust architecture. It consists of two different networks: one external and another internal. In the external architecture, there are all the BYOD devices. Then, a smart hub is the “bridge” between the two networks. In fact, the IoT devices will be placed in the internal network to guarantee their security and trust. We also consider three possible states in which the IoT entity can perform: join, stay, and leave.

The first state happens when the IoT device joins the IoT network. In this

case, the interaction can be performed only with the smart hub that will check if the new device can be trusted by implementing an adaptive trust model that will check a reputation database, a threat database, the intended context of the device (i.e., which other IoT entities will need to interact with) and will perform a risk assessment. If the IoT device is accepted, it will interact only with the smart hub and a limited number of IoT devices (i.e., in our example, the SCM can interact with the SF but not with the smart lock).

Then, when the IoT devices stay in the network, they are under the control of the smart hub. If something suspicious happens (i.e., the SCM checks every minute if there is milk available in the SF), the smart hub can perform trust decisions in order to limit network access for the device up to put it in quarantine or ban from the network.

Finally, when a device leaves the network, it must notify this action to the smart hub and the connected IoT entities in order to make them aware of its leaving process. Thus, the IoT entity is disconnected from the network, and its trusted status is saved in the reputation database. This measure is useful in the case it will join the network again.

Coming back to the transversal activities of the K-Model, they are critical and can be considered in many or all the phases (especially traceability and documentation). As we explained before, they are seven.

1. **Traceability:** It connects all the phases among them, and in every phase, it connects the elements of the IoT entity under development. For example, thanks to traceability, the requirements are connected among them, a need is connected to the requirements, the models are connected among them and with the requirements. Without traceability, the SDLC can be less effective. In fact, in the case of a modification of an element (i.e., a requirement), traceability helps developers avoid domino effects or unintended consequences due to the modification or deletion of an element for the other connected elements.
2. **Documentation:** It is an important activity because every decision must be documented to be checked or used in the following phases of the K-Model. For

example, a stakeholder and its needs must be collected to elicit the requirements and check if the IoT entity fulfils the needs.

3. **Metrics:** This activity guarantees to improve and check the elements along the whole SDLC. In fact, a metric specified in the requirements phase must be checked in the verification and validation phase, and it is the parameter that allows the smart hub to perform trust decisions in the utilization phase.
4. **Gates:** Between every phase, there is a gate. It is possible to proceed from a phase to the following one only if a phase is completed. Developers and stakeholders perform gate reviews in order to let the product flow proceeds.
5. **Threat Analysis:** This activity is critical during the whole SDLC. In the early phases, it is useful to develop the proper requirements and model to minimize or avoid threats. Moreover, it is useful in the utilization phase in order to perform trust decisions during the join and stay states.
6. **Risk Management:** Trust is strictly connected to risk. This activity is significant, especially in the utilization phase, where risk assessment helps in deciding if an IoT entity can join a network or not.
7. **Decision Making:** It is crucial both for the developers and the IoT entity. In the requirements phase, we have proposed the POM methodology to help developers in deciding which requirement is most important for the trust level of the IoT entity. Furthermore, in the utilization phase, it is enhanced by the adaptive trust model.

In Figure 6.1, we show the whole framework focusing on input, output and the tasks that are performed during the phases. This figure is a resume of the framework that can help the reader understand better the phases and their connections. For each phase, we have deeply described the tasks in their section.

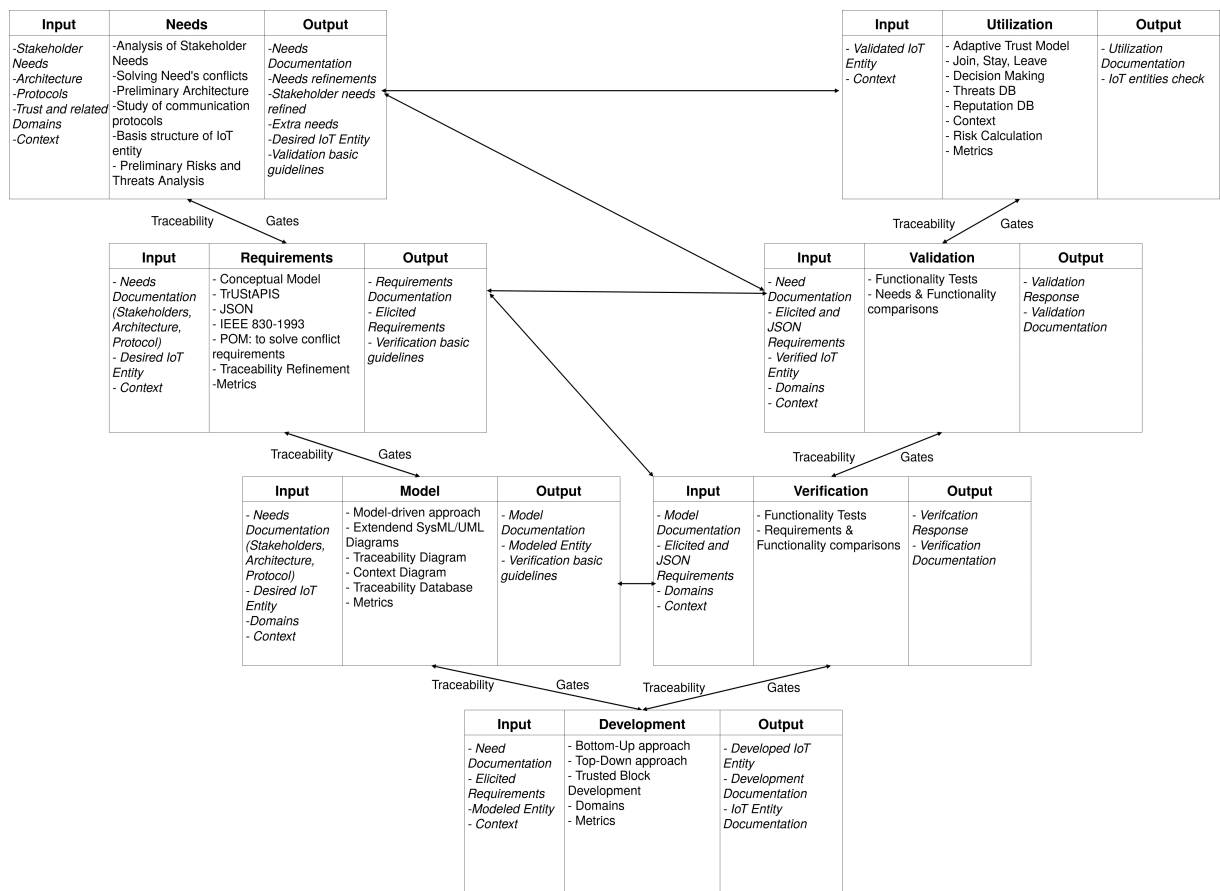


Figure 6.1: Phases, Input and Output of the whole Framework

6.2 Future work

Even if this thesis work answers the need to consider trust more in the SDLC and create a trusted architecture in order to guarantee trust among IoT entities, we need to improve our study in future work. Thus, we identify some research questions that still remain open and require further study.

Segregated Trust Architecture As future work, we will validate this architecture by testing the environment in a real smart home with multiple IoT devices. We will perform a trust analysis for each IoT entity, implementing our adaptive trust model for join and stay states. We will also test the architecture against known attacks in order to provide further protection to the IoT devices. Considering the devices belonging to the internal network, we plan to test them, exploring trust, privacy, and security implications. About them, we will give preference to devices supported by OpenHAB¹ in order to implement and expand our trust model as well as its threat model part. Finally, the new relationships established by the smart devices will help validate the benefits of our trust model and the segregated trust architecture.

TrUStAPIS and Model-driven Approach We will apply the TrUStAPIS methodology in a real and more complex scenario in order to demonstrate its validity and usefulness. Moreover, we will develop a tool to elicit and store requirements using our JSON template. In this way, the requirements elicitation process will be automatized, mitigating the subjectivity issue that can be raised by developers. Then, we will present a survey analysing our work compared to other requirement elicitation existing methodologies. Concerning POM methodology, we will compare our methodology against AHP in the same scenario, and it will be used along with TrUStAPIS in the new scenario. About our model-driven approach, we are developing a tool to draw the proposed diagrams allowing developers to implement them properly. About the traceability diagram, we will propose a methodology to recognize if two or more diagrams should be connected according to proper keywords related to the elements of the diagrams. Moreover, we will develop tool support for

¹<https://www.openhab.org/>

the analysis of clusters in order to improve traceability effectiveness. Finally, we will validate our model-driven approach in a real and complex scenario, avoiding using dummy elements.

6.3 Future Lines of Research

The topic of trust and IoT is wide. With this thesis work we have filled the gap of including trust during the SDLC of an IoT entity. But there are other topics that need to be tackled. The research community is working on them, but there is still a lot of work to do. The lines of research are the following:

Integration of security, trust and reputation requirements and model methodologies With our thesis work, we have moved forward in this line of research. However, we think that a research effort is still needed. Our TrUStAPIS methodology, along with other methodologies for security requirements elicitation (i.e., TROPOS, Secure TROPOS, I* [21, 111, 162]) can be merged to provide developers with a complete tool for requirements elicitation leading to well-established best practices. This consideration can also be useful for the modeling phase, where our model-driven approach and other existing methodologies such as UMLTrust [152], or SecureUML [93] can be analyzed together in order to explore different methodologies that can be helpful in the SDLC of any System. Investigating a way to integrate these methodologies for, including security, trust, and reputation can lead to an excellent benefit for the SDLC and developers. Moreover, because of TrUStAPIS considers also other domains such as privacy, usability, identity, safety and availability, it will be useful to consider also them to develop a complete model.

Configuration and visual support for trust and reputation implementation With our thesis work, we have covered partially also this aspect by proposing tools (i.e., JSON templates, context, and traceability graphic diagrams) to give developers and stakeholders a visualization tool for the development of a trusted IoT entity. Anyhow, extra steps in this direction can boost productivity by focusing on

the core functionalities of a trusted IoT entity. It can also be a way to provide developers with tools that will help them in writing code in order to create libraries or frameworks into a well-known practice to be implemented during the development phase. This could be more effective if the frameworks were also integrated into other phases of the SDLC in order to enable an automatic verification and derivation of the entities under development.

Creation of a standard trust model for the IoT With part of our research that we published in [47], we have analyzed the trust models of three different manufacturers (i.e., Google Mini Home, Alexa Echo Dot, and Philips Hue Lights), discovering that their trust models are very different among them. For this reason, we believe that it is essential that in the near future a standard trust model will be developed for IoT entities according to their contexts. In fact, huge differences among IoT entities will lead to difficulty in implementing both trust and security among IoT entities and users. Furthermore, if a standard protocol will be taken into consideration and developed, it will increase trust in the IoT devices and their users. In our work, we have proposed a general trust model considering the possible users of the device, giving each of them a role and a score according to the context of the required functionality. Anyhow, we think that further investigation will be needed in this direction in order to propose a general trust model for IoT entities.

Trust and Social Internet of Things Social internet of Things (SIoT) is a new concept binding the IoT entities and their users with the IoT entities and users of their friends, family members, or colleagues. This line of research is still in its infancy, and the SIoT concept must be further clarified and explored. Moreover, it can be merged with the previous one because SIoT can be considered as two-dimensional, where the relationships among users are important too and they must be considered in a trust model. Typically, in IoT paradigms, this dimension is not considered. However, in a world strongly connected where users have many relationships among them, this parameter can be helpful in developing trust models that take these connections under consideration. Besides, an exploration of where and how the SIoT can be

applied both in the professional and consumer IoT is needed. In fact, SIoT can also be helpful in business IoT (i.e., industrial IoT), specifying the interaction of IoT entities and users according to their duties (i.e., security clearance).

Resumen en español

7.1 Introducción

El Internet de las Cosas (IoT de sus siglas en inglés) permite que los seres humanos y las entidades inteligentes cooperen entre ellos a través de Internet y en cualquier lugar. Las entidades IoT desarrolladas y utilizadas están creciendo cada año, y “se espera que haya más de 64.000 millones de dispositivos IoT en todo el mundo en 2025” [8]. Esta predicción establece que el paradigma de IoT definirá cómo se conectará el mundo. Por esta razón, surgirán muchas oportunidades, pero también muchos problemas [31]. Las entidades deben establecer una relación para llevar a cabo una acción incluso en casos de incertidumbre, es decir, que sean desconocidas entre ellas. La confianza ofrece una forma de mitigarlos. De hecho, una entidad debería interactuar con otra solo si se establece confianza entre ellas. Debido a la incertidumbre, la interoperabilidad y la heterogeneidad de IoT, lograr la confianza sigue siendo un desafío. Además, considerando que comunidades de investigación aisladas han abordado estos aspectos por separado, es deseable un enfoque holístico [46].

La confianza es difícil de definir ya que el concepto se puede usar en áreas desde Filosofía a Informática [46]. Además, depende en gran medida del contexto. De hecho, confiar “significa muchas cosas para muchas personas” [43]. Esta premisa es

más que cierta para IoT. De hecho, las entidades de IoT pueden funcionar en varios contextos y, si consideramos la confianza en estos contextos, podemos mejorar la protección de estos dispositivos.

Hoffman et al. [70] y Pavlidis [124], consideraban que la confianza dependía fuertemente de otras propiedades como la seguridad y la privacidad. Ferraris et al. [50] declararon que estas relaciones son aún más importantes durante el desarrollo de una entidad de IoT. De hecho, como también lo afirman Mohammadi et al. [109] los mecanismos de confianza pueden ser fundamentales y requieren más investigación en este campo. Por ello, en nuestra opinión, es crucial considerar la confianza desde las fases iniciales del ciclo de vida del software (SDLC de sus siglas en inglés) para poder desarrollar correctamente las relaciones de confianza entre las entidades inteligentes. Este enfoque podría ayudar a proteger las entidades inteligentes y dar importantes reglas de comportamiento durante las interacciones con otras entidades inteligentes.

En una relación de confianza, hay básicamente dos actores involucrados: el que confía y el que deposita la confianza. El que confía es el que confía activamente y el fideicomisario es el que mantiene la confianza. Podemos afirmar que esta colaboración es necesaria cuando el que confía necesita que el que deposita la confianza realice una acción o cumpla un objetivo considerando un contexto particular. Este objetivo no es alcanzable por el que confía solo. Por esta razón, se necesita el que deposita la confianza. Las métricas de confianza son necesarias para calcular un nivel de confianza que ayude al que confía a decidir si se puede confiar en un que deposita la confianza [89]. Este valor debe calcularse antes de que los dos actores comiencen la colaboración. Además, el nivel de confianza podría cambiar con el tiempo de manera positiva o negativa debido al comportamiento correcto o incorrecto de los que están en la relación de confianza [65].

Por ejemplo, en un entorno de IoT, el que confía puede ser el usuario y el que deposita la confianza puede ser el dispositivo de IoT.

Sin embargo, la confianza, la seguridad, la privacidad y otros aspectos importantes generalmente se consideran solo durante las fases finales del SDLC, y esto puede generar problemas. Creemos que es crucial considerar la confianza no solo durante la utilización de un dispositivo de IoT sino también desde las primeras fases del ciclo

de vida de desarrollo de software y sistemas (SDLC).

7.2 Objetivos de la tesis

Como dijimos anteriormente, creemos que para considerar la confianza correctamente en el IoT, necesitamos integrarla en los dispositivos de IoT no solo durante la implementación y utilización de dichos dispositivos, sino también en todo el ciclo de vida de desarrollo del sistema (SDLC) de una entidad IoT. De hecho, hasta ahora, no existen tales enfoques que cubran todo el SDLC con confianza, sino solo una parte. Por esta razón, nuestro objetivo es redefinir los enfoques y herramientas que ayudan a los desarrolladores en todo el SDLC considerando la confianza en cada fase.

Tanto en el ciclo de vida de desarrollo de sistemas [67] como en el ciclo de vida de desarrollo de software [106], una de las primeras fases del SDLC está relacionada con la ingeniería de requisitos. De hecho, la recogida de requisitos en las primeras fases del SDLC es una tarea importante que aporta beneficios a las siguientes fases del SDLC y evita problemas que podrían ocurrir en fases posteriores. Los desarrolladores suelen obtener los requisitos según las necesidades de las partes interesadas en la entidad IoT.

Los lenguajes de requisitos existentes se han utilizado ampliamente con la introducción de metodologías orientadas a objetivos [21, 100, 111, 163], pero no se han desarrollado para IoT y no consideran la confianza en otros dominios de seguridad. De manera similar, la confianza y los dominios relacionados como la seguridad, la identidad, la usabilidad y la privacidad no se consideraron adecuadamente en las primeras fases de SDLC [120]. Por el contrario, para garantizar la confianza, es importante considerar otros dominios relacionados con ella, como afirman Hoffman [70] y Pavlidis [124]. Siguiendo esta premisa, Rios et al. [132] han propuesto un trabajo que considera la privacidad en la negociación de confianza, y Gago et al. [46] avanzó considerando tanto la identidad como la privacidad relacionadas con la confianza en el campo de IoT.

Nuestro objetivo es continuar en esta dirección, considerando los dominios relacionados con la confianza de manera integral en IoT, considerando la ingeniería de

requisitos como un elemento crucial en nuestro marco para garantizar la confianza en una entidad de IoT durante todo el SDLC.

Después de la fase de obtención de requisitos, está la fase de modelado donde tanto UML [137] como SysML [56] son ampliamente utilizados por los desarrolladores. De hecho, estos diagramas se han creado para explorar las diferentes funcionalidades de un software/sistema genérico en desarrollo. De todos modos, estos lenguajes de modelado originales no tenían características para implementar seguridad, privacidad o confianza. Por esta razón, es necesario definirlos para ayudar a los desarrolladores a modelar adecuadamente la confianza y los dominios relacionados.

Además, durante el desarrollo de una entidad de IoT, los desarrolladores pueden considerar varios enfoques para realizar esta tarea crucial. Un enfoque ampliamente utilizado es el denominado de arriba hacia abajo. Básicamente es una forma de considerar el problema desde una perspectiva general a una específica. Además, el enfoque de arriba hacia abajo se puede utilizar incluso para el desarrollo de software a través de una estructura funcional de desglose (FBS de sus siglas en inglés) o una estructura de descomposición del trabajo (WBS de sus siglas en inglés) [77]. Sin embargo, en nuestro caso, es importante considerar no solo las funcionalidades sino también sus conexiones a los dominios como la confianza y la seguridad para poder dividir y realizar el análisis de acuerdo a su alcance.

Por el contrario, el enfoque de abajo hacia arriba comienza desde un punto de vista específico hasta una visión general del sistema. Es un método utilizado especialmente en ingeniería de software [55, 77], pero también se puede utilizar para desarrollar infraestructuras de IoT [129]. En nuestro artículo, avanzamos y lo consideramos de acuerdo con los diferentes contextos y dominios de las entidades de IoT.

En nuestra opinión, estos dos enfoques por sí solos no son suficientes para el desarrollo de una entidad de IoT confiable. De hecho, es fundamental destacar que una entidad de IoT está compuesta por software, y una forma eficaz de desarrollar el código es siguiendo un enfoque de estado finito como lo establecen [156] y [23]. Este enfoque es aún más crítico en un entorno como el IoT, donde generalmente todas las funcionalidades se realizan por separado y siguiendo un proceso paso a paso. En este trabajo, ampliaremos este enfoque además de los enfoques de abajo hacia arriba y

de arriba hacia abajo.

La verificación y validación son dos fases fundamentales para finalizar el desarrollo de una entidad IoT. De hecho, a través de la verificación es posible decir si *la entidad se ha construido de manera correcta*, lo que significa que está funcionando como se esperaba. Por otro lado, mediante la validación podemos afirmar que *se ha construido la entidad correcta*, esto es, la entidad IoT ha sido desarrollada tal y como fue pensada para la necesidad originada.

7.3 Diseño de un Marco de Confianza

En esta tesis, hemos propuesto el diseño de un marco de confianza para considerar la confianza y los dominios relacionados durante el SDLC de una entidad de IoT. El estudio de los trabajos previos nos ha demostrado que la confianza es un aspecto que no se ha considerado en profundidad en el área de Ingeniería de sistemas y Software. Es por esta razón que nuestro objetivo es llenar este vacío y proporcionar información sobre conceptos de confianza que la conectan con dominios importantes y relacionados, como la privacidad y la seguridad. Esto es principalmente importante en un entorno como el IoT que trae nuevas posibilidades pero también desafíos de seguridad. De hecho, pensamos que para garantizar la confianza debe ser fundamental tenerlo en cuenta en todo el SDLC y no solo cuando el dispositivo IoT ha sido desarrollado y utilizado por los clientes.

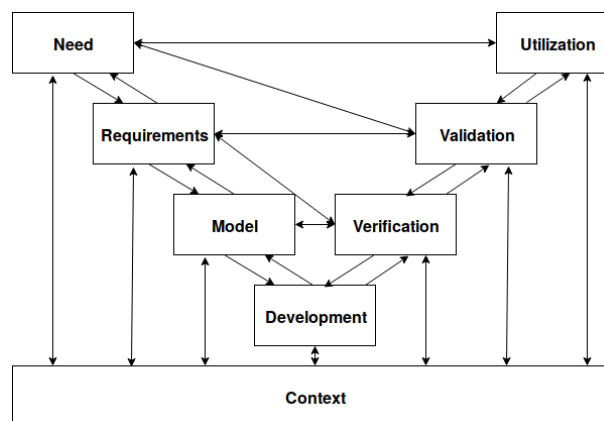


Figure 7.1: Modelo K

Por esta razón, nuestro marco garantiza que la confianza y otros dominios se consideren durante todo el SDLC desde las primeras fases hasta la utilización de los dispositivos de IoT. El marco está compuesto por un Modelo K, propuesto en la Figura 7.1, y siete actividades transversales: documentación, trazabilidad, gestión de riesgos, modelado de amenazas, toma de decisiones, métricas y pasarelas.

7.3.1 Modelo K

El Modelo K tiene siete fases más el contexto que siempre está presente. Este marco permite a los desarrolladores crear el producto de IoT confiable y adecuado de manera que refleje lo que los usuarios y proveedores quieren gracias a su consideración en la primera fase del SDLC: la fase de necesidad (i.e., Need).

7.3.1.1 Need y Requisitos

En esta fase, es muy importante tener clara la arquitectura en la que se utilizará la entidad de IoT prevista. Además, todas las partes interesadas dan su opinión sobre sus necesidades específicas que serán recopiladas por los desarrolladores, que generarán los requisitos adecuados en la segunda fase del SDLC de acuerdo con estas necesidades. Para considerar adecuadamente la confianza y sus dominios conectados, definimos siete tipos de requisitos (es decir, confianza, usabilidad, seguridad, disponibilidad, privacidad, identidad y seguridad) proponiendo un método de obtención de requisitos basado en JSON: este será el enfoque TrUStAPIS.

Para transformar las necesidades en requisitos, hemos desarrollado un método de obtención de requisitos: TrUStAPIS. Al implementar este método, los desarrolladores pueden considerar la confianza y los otros dominios durante todo el proceso de obtención de requisitos. La palabra “TrUStAPIS” es un acrónimo que se origina a partir del uso de las primeras letras de cada uno de los siete dominios tomados en consideración: Confianza (escrito íntegramente porque es el central), Usabilidad, Seguridad, Disponibilidad, Privacidad, Identidad y La seguridad. Este método ayuda a los desarrolladores a obtener los requisitos específicos para su dominio conectándolos a través de la trazabilidad. Además, este método permite la obtención de requisitos considerando aspectos dinámicos relacionados con IoT.

TrUStAPIS considera varios elementos para diseñar y luego declarar los requisitos. Estos elementos son: actores y roles, acciones y medidas, y meta y contexto.

- **Actor.** Un *actor* puede ser un ser humano o una entidad de IoT. Es la entidad que se necesita para cumplir un objetivo. La consecución del objetivo se puede hacer solo o con la cooperación de otro actor. Por lo tanto, un actor podría tener diferentes *roles* (es decir, en un dominio de confianza, los actores son fideicomisarios y que deposita la confianzas). De todos modos, durante el proceso de obtención de requisitos, los consideramos por separado porque para obtener los requisitos adecuados, preferimos hacer esta distinción.
- **Acción.** Una *acción* está relacionada con la tarea realizada por el actor. Una acción puede incluir medidas o no. Una *medida* ayuda a las partes interesadas y desarrolladores a modelar un requisito que será verificado y validado en las fases posteriores del Modelo K.
- **Objetivo.** Un *objetivo* es el propósito final que representan los requisitos. Los actores pueden lograrlo a través de acciones objetivo. Este objetivo puede estar relacionado con una capacidad particular de la entidad de IoT según el dominio de requisitos. Además, de acuerdo con la dinámica del IoT, un mismo actor puede involucrarse en diferentes objetivos o realizar diferentes acciones para cumplirlos. Cuando una acción determina directamente un objetivo, se denomina goalAction.
- **Contexto.** El *contexto* está relacionado con tres elementos. El primero es el dominio de requisitos. Cada uno de estos dominios tiene sus propias características. El segundo está relacionado con el entorno de IoT. Finalmente, el tercero se refiere a la consecución del objetivo.

Para resumir todos estos elementos, proponemos un modelo conceptual que se muestra en la Figura 7.2 para presentar las relaciones entre todos los componentes de TrUStAPIS.

El *actor*, puede ser un ser humano o una entidad de IoT y es el “sujeto” del requisito. Cada actor juega un *papel* que depende del contexto. Una *acción* puede

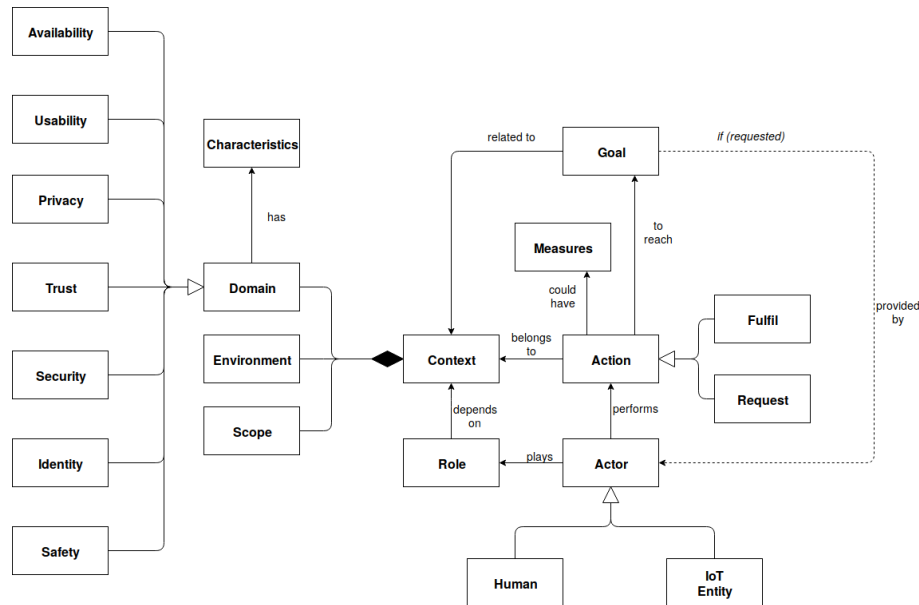


Figure 7.2: Modelo conceptual - TrUStAPIS

considerarse como el “verbo” del requisito y se realiza para cumplir o para solicitar un *objetivo*. En el primer caso, el “objeto” del requisito es directamente el objetivo. De lo contrario, el “objeto” del requisito es otro actor que realiza una acción para cumplir el objetivo propuesto. Una posibilidad es que una acción pueda contener algunas *medidas*. Estas medidas son importantes para alcanzar la meta y verificar y validar los requisitos en las siguientes fases del Modelo K. Finalmente, el *contexto* consta de tres componentes: un propósito, un entorno y un dominio. El *propósito* está relacionado con el propósito del objetivo que debe cumplir un actor. El *entorno* se relaciona con el lugar físico donde se realiza la acción. El *dominio* se refiere a los siete tipos de requisitos que hemos identificado. Finalmente, cada dominio tiene sus *características*, que explicaremos junto a los requisitos. Las flechas representan las relaciones entre conceptos. Cada flecha está descrita por un texto relacionado con la dependencia entre conceptos mientras que la dirección de la flecha representa el orden de la conexión. Además, hay una flecha discontinua opcional que representa el caso de que un actor secundario cumpla un objetivo. El triángulo se utiliza para representar una especialización (es decir, un actor puede ser un ser humano o una entidad de IoT). Finalmente, el rombo representa una composición (es decir, el contexto se

compone de un dominio, un entorno y un propósito).

Además, para ayudar a los desarrolladores a obtener los requisitos, proponemos una plantilla JSON tal y como se muestra en la Figura 7.3.

```

1 * {
2 *   "IoT_requirement" : {
3 *     "Context" : {
4 *       "Domain" : [{
5 *         "Trust" : {
6 *           "Characteristic" : ["CharTrst_1", "...", "CharTrust_N"] },
7 *         "Usability" : {
8 *           "Characteristic" : ["CharUsab_1", "...", "CharUsab_N"] },
9 *         "Security" : {
10 *          "Characteristic" : ["CharSec_1", "...", "CharSec_N"] },
11 *         "Availability" : {
12 *          "Characteristic" : ["CharAvbt_1", "...", "CharAvbt_N"] },
13 *         "Privacy" : {
14 *          "Characteristic" : ["CharPriv_1", "...", "CharPriv_N"] },
15 *         "Identity" : {
16 *          "Characteristic" : ["CharIdnt_1", "...", "CharIdnt_N"] },
17 *         "Safety" : {
18 *          "Characteristic" : ["CharSft_1", "...", "CharSft_N"] } } ],
19 *     "Environment" : " " ,
20 *     "Scope" : " " },
21 *   "Actor" : {
22 *     "Role" : " " ,
23 *     "Type" : [ "Human", "Iot Entity" ] },
24 *   "Action" : {
25 *     "Type" : ["Fulfil", "Request"] ,
26 *     "Measure" : ["Meas_1", "...", "Meas_N"]},
27 *   "Goal" : " "
28 * }
29 }
```

Figure 7.3: TrUStAPIS - plantilla JSON para obtener requisitos

Hemos elegido JSON porque es esquemático y legible por humanos y máquinas. Además, es compatible con muchos lenguajes de programación (por ejemplo, Java ¹). Este aspecto permite compartir un código de requisitos entre stakeholders y desarrolladores a través de aplicaciones [6]. Además, el código JSON es una herramienta útil para mapear las necesidades que se han identificado en la fase anterior del Modelo K y es útil en las fases de desarrollo y verificación, ayudando a los desarrolladores a automatizar los procesos.

Por lo tanto, completar la plantilla JSON puede ayudar a los desarrolladores a escribir los requisitos adecuados. Un requisito escrito debe estar compuesto al menos por:

1. un *actor*;
2. una palabra clave (“*shall*”);
3. un *objetivo* cumplido a través de una *acción*;

¹<https://www.java.com>

Entonces, un requisito también podría tener opcionalmente:

- Uno o más actores secundarios que realizan una acción para cumplir el objetivo.
- Una o más acciones necesarias para alcanzar la meta final Acción.
- Una o más medidas. Los desarrolladores y las partes interesadas los necesitan para verificar y validar los requisitos.

Esta estructura se resume y formaliza utilizando la siguiente *declaración (1)*.

(1) Actor **shall** *predicado*

Por lo tanto, el actor también se conoce como el actor principal y es el sujeto de la *declaración (1)*.

Se ha elegido la palabra clave *shall* en lugar de las palabras *should* o *must*, porque *shall* define que el requisito es contractualmente vinculante y deberá ser implementado y posteriormente verificado y validado.

El *predicado* puede tener diferentes formas. La forma básica está representada solo por un goalAction. De lo contrario, es posible tener un predicado más complejo. Podría estar compuesto por uno o más actores secundarios, acciones y/o medidas. Esta composición depende estrictamente del contexto y define el requisito escrito final.

7. 3. 1. 2 Modelo

Después de la fase de requisitos, tenemos que abordar el problema de la implementación del modelo. Durante la fase de modelado, es útil considerar lenguajes para tal efecto como UML [137] y SysML [56]. Además, para incluir la gestión de la confianza en esta fase, es obligatorio considerar los modelos de decisión y evaluación de la confianza identificados por Moyano et al. [112]. Debido a que diseñamos nuestro marco de trabajo para el entorno de IoT, de acuerdo con su heterogeneidad y dinamicidad, existe la posibilidad de que cada entidad de IoT inteligente deba ser modelada con diferentes modelos de confianza. De hecho, dependiendo de la arquitectura de

IoT prevista (es decir, centralizada o distribuida), estos modelos se implementarán para definir la arquitectura adecuada y el tipo de comunicaciones confiables entre las entidades de IoT.

Así, según el trabajo desarrollado por Uddin [152], hemos identificado la posibilidad de considerar la confianza en los diagramas UML. Consideramos los diagramas UML y SysML en nuestro trabajo. Necesitamos considerar los lenguajes de modelado porque necesitamos el Diagrama de clases (solo presente en UML) y el Diagrama de requisitos (un diagrama SysML).

De todos modos, ampliaremos los diagramas básicos relacionados con UML y SysML y propondremos dos nuevos diagramas: el de trazabilidad y el diagrama de contexto.

Los diagramas básicos de UML que vamos a ampliar serán el diagrama de casos de uso, el diagrama de clases, el diagrama de actividades, el diagrama de secuencia, el diagrama de la máquina de estados y el diagrama de requisitos. Hemos elegido estos diagramas porque permiten a los desarrolladores implementar aspectos cruciales de una entidad de IoT. De hecho, el diagrama de casos de uso proporciona a los desarrolladores una herramienta útil para modelar las interacciones universales de las entidades de IoT. El diagrama de requisitos permite a los desarrolladores considerar los requisitos generados en la segunda fase del Modelo K para modelarlos con conexiones a los otros diagramas agregando características adicionales. En tercer lugar, el diagrama de clases será muy importante para ayudar a los desarrolladores a escribir el software de la entidad de IoT. Finalmente, los diagramas de la máquina de actividad, secuencia y estado permiten a los desarrolladores especificar las funcionalidades e interacciones de la entidad de IoT bajo tres perspectivas diferentes. Además, es importante agregar que, incluso si estos diagramas exploran diferentes aspectos de la entidad de IoT modelada, se pueden combinar. Por ejemplo, un diagrama de secuencia puede representar el flujo de acciones que pertenecen a un diagrama de casos de uso. Las mismas acciones se pueden especificar mediante un diagrama de actividad o un diagrama de máquina de estados. Además, considerando los nuevos diagramas, el diagrama de trazabilidad es fundamental para realizar un seguimiento de las conexiones entre diagramas. Esta función está habilitada porque cada diagrama tiene un

identificador único que permite a los desarrolladores hacer referencia a ellos de forma única. El diagrama de trazabilidad puede considerarse un metadiagrama. Por otro lado, el diagrama de contexto se utilizará para mapear los diferentes contextos que se considerarán para la entidad IoT. Este diagrama permite a los desarrolladores considerar todos los contextos pertenecientes a la entidad IoT, desde la fase de modelado para dividir las funcionalidades según los diferentes contextos, mejorando la confianza y la seguridad.

Para cada diagrama, describimos las características básicas y las mejoras con respecto a los diagramas UML y SysML originales. Un aspecto importante que define los diagramas es la consideración de la confianza y los dominios relacionados. Los dominios relacionados con la confianza son usabilidad, seguridad, disponibilidad, privacidad, identidad y seguridad [49].

7. 3. 1. 3 Desarrollo, verificación y validación.

Después de la fase de modelado, está la fase de desarrollo. En esta fase, los desarrolladores tendrán en cuenta toda la documentación producida en las fases anteriores produciendo el código adecuado que definirá el comportamiento de la entidad IoT. Para dar una forma sistemática, proponemos dos enfoques para implementar funcionalidades y contextos. Sobre el primero, hemos propuesto un enfoque de arriba hacia abajo con una Estructura de Desglose del Dominio Funcional (FDBS de sus siglas en inglés) que es útil para especificar dominios y funcionalidades que pertenecen a la entidad de IoT en desarrollo. En cuanto a los contextos, hemos presentado un enfoque de abajo hacia arriba considerando dominios y contextos pertenecientes a la entidad de IoT en desarrollo. Aquí, tenemos que partir de los contextos individuales agregándolos de acuerdo con su alcance en supercontextos. Finalmente, todos los contextos compondrán la entidad IoT en su totalidad. En cuanto al enfoque de arriba hacia abajo, se representa como un árbol. Además, para fusionar y considerar juntos los enfoques de arriba hacia abajo y de abajo hacia arriba, presentamos un desarrollo de estado finito. Este estilo de desarrollo es útil para los desarrolladores ya que les permite considerar todas las particularidades de los contextos y funcionalidades en

bloques individuales de código de acuerdo con los contextos y funcionalidades. Estos tres enfoques son útiles para desarrollar la entidad IoT confiable de acuerdo con los documentos producidos en las fases anteriores del Modelo K y la arquitectura donde se utilizará.

Una vez completada la fase de desarrollo, abordamos la fase de verificación. Es útil garantizar que todas las funcionalidades sean correctas, que los dominios (es decir, confianza, seguridad) estén bien definidos y que la entidad de IoT se haya construido correctamente. Básicamente, en esta fase los desarrolladores verificarán que las funcionalidades reflejan los requisitos y los modelos realizan las pruebas de verificación. El código JSON escrito en el proceso de obtención de requisitos puede ser útil para automatizar el proceso. En el caso de que alguna prueba de verificación no se complete con éxito, se realizará una modificación de las funcionalidades. De lo contrario, es posible pasar a la siguiente fase.

Durante la fase de validación, los desarrolladores comprobarán que se haya construido la entidad correcta analizando cómo se desempeña la entidad IoT desarrollada en su entorno previsto. Aquí, se tienen en cuenta las necesidades originarias para comprobar si la entidad de IoT las cumple y si se satisfacen los requisitos planteados. Si no, las funcionalidades deben cambiarse para reflejar las necesidades. Por el contrario, si las pruebas de validación finalizan correctamente, es posible vender el dispositivo para el uso previsto. Esto conduce a la fase final del SDLC: la fase de utilización.

7. 3. 1. 4 Utilización

En esta fase, la entidad de IoT interactuará con otras entidades de IoT y se colocará en su arquitectura de red prevista (es decir, una casa inteligente). Sin embargo, para garantizar la confianza entre las entidades de IoT, hemos propuesto una arquitectura de confianza segregada. Consta de dos redes diferentes: una externa y otra interna. En la arquitectura externa, están todos los dispositivos "trae tu propio dispositivo" (BYOD de sus siglas en inglés) [108]. Entonces, un centro de actividad inteligente es el "puente" entre las dos redes. De hecho, los dispositivos IoT se

colocarán en la red interna para garantizar su seguridad y confianza. Consideramos también tres posibles estados en los que la entidad de IoT puede actuar: unirse, permanecer y abandonar la red.

El primer estado ocurre cuando el dispositivo IoT se une a la red IoT, en este caso la interacción solo se puede realizar con el centro de actividad inteligente que verificará si se puede confiar en el nuevo dispositivo implementando un modelo de confianza adaptativo que verificará una base de datos de reputación, una amenaza base de datos, el contexto previsto del dispositivo (es decir, con qué otras entidades de IoT necesitarán interactuar) y realizará una evaluación de riesgos. Si se acepta el dispositivo de IoT, interactuará solo con el centro de actividad inteligente y un número limitado de dispositivos de IoT.

Una vez que los dispositivos IoT permanecen en la red, están bajo el control del centro de actividad inteligente. Si ocurre algo sospechoso, el centro de actividad inteligente puede tomar decisiones de confianza para limitar el acceso a la red del dispositivo para ponerlo en cuarentena o prohibir la red.

Finalmente, cuando un dispositivo sale de la red, debe notificarlo al centro de actividad inteligente y a las entidades de IoT conectadas para que estén al tanto de su proceso de salida. Así, la entidad de IoT se desconecta de la red y su estado de confianza se guarda en la base de datos de reputación en caso de que vuelva a unirse a la red en el futuro.

7.3.2 Actividades Transversales

Las actividades transversales del Modelo K, son muy importantes y pueden ser consideradas en muchas o todas las fases (especialmente trazabilidad y documentación). Como explicamos antes, son siete.

La trazabilidad conecta todas las fases entre ellas y en cada fase conecta los elementos de la entidad IoT en desarrollo. Por ejemplo, gracias a la trazabilidad los requisitos están conectados entre ellos, una necesidad está conectada con los requisitos, los modelos están conectados entre ellos y con los requisitos. Sin trazabilidad, todo el SDLC puede verse comprometido. De hecho, en el caso de la modificación

de un elemento (es decir, un requisito), la trazabilidad ayuda a los desarrolladores a evitar efectos dominó o consecuencias no deseadas debido a la modificación o eliminación de un elemento.

La documentación es una actividad muy importante porque cada decisión debe documentarse para poder ser verificada o utilizada en las siguientes fases del Modelo K. Por ejemplo, se debe recopilar una parte interesada y sus necesidades para obtener los requisitos y verificar si la entidad de IoT satisface las necesidades.

Las métricas garantizan mejorar y comprobar los elementos a lo largo de todo el SDLC. De hecho, una métrica especificada en la fase de requisitos, debe verificarse en la fase de verificación y validación y es el parámetro que permite al centro de actividad inteligente tomar decisiones de confianza en la fase de utilización.

Las pasarelas se encuentran entre cada fase del Modelo K. Entonces, es posible pasar de una fase a la siguiente solo si se completa una fase. Los desarrolladores y las partes interesadas realizan las revisiones para permitir que el producto fluya.

El análisis de amenazas es muy importante durante todo el SDLC. En las primeras fases, es útil obtener los requisitos adecuados y modelar y desarrollar la entidad para minimizar o evitar amenazas. Además, es útil en la fase de utilización para tomar decisiones de confianza en los estados de unión y permanencia.

La gestión de riesgos se considera porque la confianza está estrictamente relacionada con el riesgo. Esta actividad es muy importante, especialmente en la fase de utilización, donde la evaluación de riesgos ayuda a decidir si una entidad de IoT puede unirse a una red o no.

Finalmente, la toma de decisiones es crucial tanto para los desarrolladores como para la entidad de IoT. En la fase de requisitos, hemos propuesto el método de ordenación por pares (POM de sus siglas en inglés) para ayudar a los desarrolladores a decidir qué requisito es más importante para el nivel de confianza de la entidad de IoT. Además, en la fase de utilización se ve reforzada por el modelo de confianza adaptativo.

Si bien este trabajo de tesis responde a la necesidad de considerar la confianza en el SDLC y de crear una arquitectura de confianza para garantizar la confianza entre las entidades de IoT, necesitamos mejorar nuestro estudio en el trabajo futuro. Además,

identificamos algunas preguntas de investigación que aún permanecen abiertas y requieren más estudio.

7.3.3 Futuras líneas de investigación

Integración de requisitos de seguridad, confianza y reputación y modelado Con nuestro trabajo de tesis, hemos avanzado en esta línea de investigación. Sin embargo, creemos que aún se necesita un esfuerzo de investigación. Nuestra metodología TrUStAPIS, junto con otras metodologías para la obtención de requisitos de seguridad (es decir, TROPOS, Secure TROPOS, I * [21, 111, 162]) se pueden fusionar para proporcionar a los desarrolladores una herramienta completa para la obtención de requisitos que conducen a las mejores prácticas bien establecidas. Esta consideración también puede ser útil para la fase de modelado, donde nuestro enfoque basado en modelos y otras metodologías existentes como UMLTrust [152] o SecureUML [93] se pueden analizar juntas para explorar diferentes metodologías que pueden ser útiles en el SDLC de cualquier sistema. Investigar una forma de integrar estas metodologías para incluir seguridad, confianza y reputación puede generar un beneficio excelente para el SDLC y los desarrolladores.

Configuración y soporte visual para la implementación de la confianza y la reputación Con nuestro trabajo de tesis, también hemos cubierto parcialmente este aspecto al proponer herramientas (es decir, plantillas JSON, diagramas gráficos de contexto y trazabilidad) para brindar a los desarrolladores y partes interesadas una herramienta de visualización para el desarrollo de una entidad IoT confiable. De todos modos, pasos adicionales en esta dirección pueden impulsar la productividad al enfocarse en las funcionalidades centrales de una entidad de IoT confiable. También puede ser una forma de proporcionar a los desarrolladores herramientas que los ayuden a escribir código para crear bibliotecas o marcos en una práctica conocida que se implementará durante la fase de desarrollo. Esto podría ser más efectivo si los marcos también se integraran en otras fases del SDLC para permitir una verificación y derivación automáticas de las entidades en desarrollo.

Creación de un modelo de confianza estándar para IoT Con parte de nuestra investigación que publicamos en [47], hemos analizado los modelos de confianza de tres fabricantes diferentes (es decir, Google Mini Home, Alexa Echo Dot y Philips Hue Lights), descubriendo que sus modelos de confianza son muy diferentes entre ellos. Por ello, creemos que de acuerdo con el trabajo futuro que hemos mencionado anteriormente, es fundamental que en un futuro próximo se cree un modelo de confianza estándar a implementar para las entidades de IoT con el fin de mejorar los aspectos de confianza y seguridad. De hecho, las grandes diferencias entre las entidades de IoT generarán dificultades para implementar tanto la confianza como la seguridad entre las entidades y los usuarios de IoT. Por otro lado, si se toma en consideración y se desarrolla un protocolo estándar, aumentará la confianza en los dispositivos IoT y sus usuarios. En nuestro trabajo, hemos propuesto un modelo de confianza general considerando los posibles usuarios del dispositivo, otorgando a cada uno de ellos un rol y una puntuación de acuerdo al contexto de la funcionalidad requerida. De todos modos, creemos que se necesitará más investigación en esta dirección para proponer y probar un modelo de confianza general para las entidades de IoT.

Confianza e Internet social de las cosas Internet social de las cosas (SIoT de sus siglas en inglés) es un nuevo concepto que vincula a las entidades de IoT y sus usuarios con las entidades de IoT y los usuarios de sus amigos, familiares o colegas. Este es un campo temprano y el concepto SIoT debe aclararse y explorarse más. Además, esta línea de investigación se puede fusionar con la anterior porque SIoT puede considerarse como bidimensional, donde también las relaciones entre usuarios son importantes y deben ser consideradas en un modelo de confianza. Normalmente, en los paradigmas de IoT, esta dimensión no se considera. Sin embargo, en un mundo fuertemente conectado donde los usuarios tienen muchas relaciones entre ellos, este parámetro puede ser útil para desarrollar modelos de confianza que tomen en consideración estas conexiones. Además, se necesita una exploración de dónde y cómo se puede aplicar SIoT tanto en el IoT profesional como en el de los consumidores. De

hecho, SIoT también puede ser útil en IoT empresarial (por ejemplo, IoT industrial), ya que especifica la interacción de las entidades y los usuarios de IoT según sus funciones (es decir, autorización de seguridad).

Bibliography

- [1] Wafa Abdelghani, Corinne Amel Zayani, Ikram Amous, and Florence Sèdes. Trust management in social internet of things: a survey. In *Conference on e-Business, e-Services and e-Society*, pages 430–441. Springer, 2016.
- [2] Hasan Abualese, Thamer Al-Rousan, and Bassam Al-Shargabi. A new trust framework for e-government in cloud of things. *International Journal of Electronics and Telecommunications*, 65, 2019.
- [3] A. Frank Ackerman, Lynne S. Buchwald, and Frank H. Lewski. Software inspections: an effective verification process. *IEEE software*, 6(3):31–36, 1989.
- [4] Isaac Agudo, Carmen Fernandez-Gago, and Javier Lopez. A model for trust metrics analysis. In *International Conference on Trust, Privacy and Security in Digital Business*, pages 28–37. Springer, 2008.
- [5] Esubalew Alemneh, Sidi-Mohammed Senouci, Philippe Brunet, and Tesfa Tegegne. A two-way trust management system for fog computing. *Future Generation Computer Systems*, 106:206–220, 2020.
- [6] Aitana Alonso-Nogueira, Helia Estévez-Fernández, and Isaías García. Jrem: an approach for formalising models in the requirements phase with json and nosql

- databases. *World Acad. Sci. Eng. Technol. Int. J. Comput. Electr. Autom. Control Inf. Eng*, 11(3):353–358, 2017.
- [7] Alfredo Altuzarra, José María Moreno-Jiménez, and Manuel Salvador. A bayesian prioritization procedure for ahp-group decision making. *European Journal of Operational Research*, 182(1):367–382, 2007.
- [8] IoT Analytics. State of the iot 2018 - number of iot devices now at 7b - market accelerating, 2018.
- [9] Ludovic Apvrille and Yves Roudier. Sysml-sec: A sysml environment for the design and development of secure embedded systems. *APCOSEC, Asia-Pacific Council on Systems Engineering*, pages 8–11, 2013.
- [10] James D Arthur and Richard E Nance. Independent verification and validation: a missing link in simulation methodology? In *Proceedings Winter Simulation Conference*, pages 230–236. IEEE, 1996.
- [11] Hany F Atlam, Ahmed Alenezi, Robert J Walters, Gary B Wills, and Joshua Daniel. Developing an adaptive risk-based access control model for the internet of things. 2017.
- [12] Farag Azzedin and Mustafa Ghaleb. Internet-of-things and information fusion: Trust perspective survey. *Sensors*, 19(8):1929, 2019.
- [13] Rosnita Baharuddin, Dalbir Singh, and Rozilawati Razali. Usability dimensions for mobile applications-a review. *Res. J. Appl. Sci. Eng. Technol*, 5(6):2225–2231, 2013.

- [14] Mohammed Bahutair, Athman Bougeuttaya, and Azadeh Ghari Neiat. Adaptive trust: Usage-based trust in crowdsourced iot services. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 172–179. IEEE, 2019.
- [15] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security for process-oriented systems. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 100–109. ACM, 2003.
- [16] Alessandro Bassi, Martin Bauer, Martin Fiedler, and Rob van Kranenburg. *Enabling things to talk*. Springer-Verlag GmbH, 2013.
- [17] Armaghan Behnia, Rafhana Abd Rashid, and Junaid Ahsenali Chaudhry. A survey of information security risk analysis methods. *SmartCR*, 2(1):79–94, 2012.
- [18] Thomas Beth, Malte Borchering, and Birgit Klein. Valuation of trust in open networks. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 1994.
- [19] Ion Bica, Bogdan-Cosmin Chifor, Stefan-Ciprian Arseni, and Ioana Matei. Reputation-based security framework for internet of things. In *International Conference on Information Technology and Communications Security*, pages 213–226. Springer, 2019.
- [20] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 164–173. IEEE, 1996.
- [21] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

- [22] Walter Bronzi, Raphael Frank, German Castignani, and Thomas Engel. Bluetooth low energy performance and robustness analysis for inter-vehicular communications. *Ad Hoc Networks*, 37:76–86, 2016.
- [23] Ted Carmely. Using finite state machines to design software. *EE Times*, 2009.
- [24] Junsheng Chang, Huaimin Wang, and Yin Gang. A dynamic trust metric for p2p systems. In *2006 Fifth International Conference on Grid and Cooperative Computing Workshops*, pages 117–120. IEEE, 2006.
- [25] Haiguang Chen, Huafeng Wu, Xiu Cao, and Chuanshan Gao. Trust propagation and aggregation in wireless sensor networks. In *2007 Japan-China Joint Workshop on Frontier of Computer Science and Technology (FCST 2007)*, pages 13–20. IEEE, 2007.
- [26] Elizabeth Chew, Marianne M Swanson, Kevin M Stine, Nadya Bartol, Anthony Brown, and Will Robinson. Performance measurement guide for information security. Technical report, 2008.
- [27] Bruce Christianson and William S Harbison. Why isn't trust transitive? In *International workshop on security protocols*, pages 171–176. Springer, 1996.
- [28] Tony Clear. Documentation and agile methods: striking a balance. *ACM SIGCSE Bulletin*, 35(2):12–13, 2003.
- [29] Piotr Cofta. Confidence, trust and identity. *BT technology Journal*, 25(2):173–178, 2007.
- [30] Piotr Cofta. *Trust, complexity and control: confidence in a convergent world*. John Wiley & Sons, 2007.

- [31] Alem Čolaković and Mesud Hadžialić. Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 144:17–39, 2018.
- [32] US Federal Trade Commission et al. Protecting consumer privacy in an era of rapid change: Recommendations for businesses and policymakers. *FTC Report*, 2012.
- [33] IEEE Computer Society. Software Engineering Standards Committee and IEEE-SA Standards Board. Ieee recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers, 1998.
- [34] Daniel Cvrček and Ken Moody. Combining trust and risk to reduce the cost of attacks. In *International Conference on Trust Management*, pages 372–383. Springer, 2005.
- [35] Pasquale De Meo, Fabrizio Messina, Maria Nadia Postorino, Domenico Rosaci, and Giuseppe ML Sarné. A reputation framework to share resources into iot-based environments. In *Networking, Sensing and Control (ICNSC), 2017 IEEE 14th International Conference on*, pages 513–518. IEEE, 2017.
- [36] Bryan DeHoff, Daniel JH Levack, and Russel E Rhodes. The functional breakdown structure (fbs) and its relationship to life cycle cost. *NASA Kennedy Space Center*, 2009.
- [37] Jeremy Dick, Elizabeth Hull, and Ken Jackson. *Requirements engineering*. Springer, 2017.
- [38] Theo Dimitrakos, B Ritchie, D Raptis, and Ketil Stølen. Model-based security risk analysis for web applications: The coras approach. In *Proceedings of the EuroWeb*. Citeseer, 2002.

- [39] Angelika Dohr, Robert Modre-Opsrian, Mario Drobics, Dieter Hayn, and Günter Schreier. The internet of things for ambient assisted living. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 804–809. Ieee, 2010.
- [40] Robert F Dyer and Ernest H Forman. Group decision support with the analytic hierarchy process. *Decision support systems*, 8(2):99–124, 1992.
- [41] Fourth Edition. Ieee guide-adoption of the project management institute (pmi®) standard a guide to the project management body of knowledge (pmbok® guide). 2011.
- [42] Mahmoud Elkhodr and Belal Alsinglawi. Data provenance and trust establishment in the internet of things. *Security and Privacy*, page e99, 2019.
- [43] John Erickson. Trust metrics. In *Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on*, pages 93–97. IEEE, 2009.
- [44] Michael Fagan. Design and code inspections to reduce errors in program development. In *Software pioneers*, pages 575–607. Springer, 2002.
- [45] Kim Fenrich. Securing your control system: the "cia triad" is a widely used benchmark for evaluating information system security effectiveness. *Power Engineering*, 112(2):44–49, 2008.
- [46] Carmen Fernandez-Gago, Francisco Moyano, and Javier Lopez. Modelling trust dynamics in the internet of things. *Information Sciences*, 396:72 – 82, 2017.
- [47] Davide Ferraris, Daniel Bastos, Carmen Fernandez-Gago, and Fadi El-Moussa. A trust model for popular smart home devices. *International Journal of Information Security*, pages 1–17, 2020.

- [48] Davide Ferraris, Joshua Daniel, Carmen Fernandez-Gago, and Javier Lopez. A segregated architecture for a trust-based network of internet of things. In *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC) (CCNC 2019)*, Las Vegas, USA, January 2019.
- [49] Davide Ferraris and Carmen Fernandez-Gago. Trustapis: a trust requirements elicitation method for iot. *International Journal of Information Security*, pages 1–17, 2019.
- [50] Davide Ferraris, Carmen Fernandez-Gago, and Javier Lopez. A trust by design framework for the internet of things. In *NTMS'2018 - Security Track (NTMS 2018 Security Track)*, Paris, France, February 2018.
- [51] US Food, Drug Administration, et al. Guideline on general principles of process validation. *US FDA: Rockville, MD*, 1987.
- [52] Kevin Forsberg and Harold Mooz. The relationship of system engineering to the project cycle. In *INCOSE International Symposium*, volume 1, pages 57–65. Wiley Online Library, 1991.
- [53] Giancarlo Fortino, Lidia Fotia, Fabrizio Messina, Domenico Rosaci, and GIUSEPPE Sarné. A reputation mechanism to support cooperation of iot devices. In *AI & IoT jointly held with AI* IA 2019, the 18th International Conference of the Italian Association for Artificial Intelligence*, pages 1–12. CEUR, 2019.
- [54] Giancarlo Fortino, Lidia Fotia, Fabrizio Messina, Domenico Rosaci, and Giuseppe ML Sarné. Trust and reputation in the internet of things: State-of-the-art and research challenges. *IEEE Access*, 8:60117–60125, 2020.

- [55] Christopher W Fraser and Robert R Henry. Hard-coding bottom-up code generation tables to save time and space. *Software: Practice and Experience*, 21(1):1–12, 1991.
- [56] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.
- [57] Diego Gambetta et al. Can we trust trust. *Trust: Making and breaking cooperative relations*, 13:213–237, 2000.
- [58] Vangelis Gazis. A survey of standards for machine to machine (m2m) and the internet of things (iot). *IEEE Communications Surveys & Tutorials*, 2016.
- [59] Vangelis Gazis. A survey of standards for machine-to-machine and the internet of things. *IEEE Communications Surveys & Tutorials*, 19(1):482–511, 2017.
- [60] Michael Geisser and Tobias Hildenbrand. A method for collaborative requirements elicitation and decision-supported requirements analysis. In *IFIP World Computer Congress, TC 2*, pages 108–122. Springer, 2006.
- [61] Khusvinder Gill, Shuang-Hua Yang, Fang Yao, and Xin Lu. A zigbee-based home automation system. *IEEE Transactions on consumer Electronics*, 55(2), 2009.
- [62] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Requirements engineering for trust management: model, methodology, and reasoning. *International Journal of Information Security*, 5(4):257–274, 2006.
- [63] Janusz Górski, Aleksander Jarzębowicz, Rafal Leszczyna, Jakub Miler, and Marcin Olszewski. Trust case: Justifying trust in an it solution. *Reliability Engineering & System Safety*, 89(1):33–47, 2005.

- [64] Quandeng Gou, Lianshan Yan, Yihe Liu, and Yao Li. Construction and strategies in iot security system. In *2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing*, pages 1129–1132. IEEE, 2013.
- [65] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *IEEE Communications Surveys & Tutorials*, 3(4):2–16, 2000.
- [66] David J Hand. Statistics and the theory of measurement. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, pages 445–492, 1996.
- [67] Cecilia Haskins, Kevin Forsberg, Michael Krueger, D Walden, and D Hamelin. Systems engineering handbook. In *INCOSE*, 2006.
- [68] Xiali Hei, Xiaojiang Du, Shan Lin, and Insup Lee. Pipac: patient infusion pattern based access control scheme for wireless insulin pump system. In *INFOCOM, 2013 Proceedings IEEE*, pages 3030–3038. IEEE, 2013.
- [69] Debra S. Herrmann. *Complete Guide to Security and Privacy Metrics: Measuring Regulatory Compliance, Operational Resilience, and ROI*. Auerbach Publications, Boston, MA, USA, 1st edition, 2007.
- [70] Lance J Hoffman, Kim Lawson-Jenkins, and Jeremy Blum. Trust beyond security: an expanded trust model. *Communications of the ACM*, 49(7):94–101, 2006.
- [71] Fei Hu. *Security and Privacy in Internet of Things (IoTs): Models, Algorithms, and Implementations*. CRC Press, 2016.

- [72] Yih-Chun Hu, Adrian Perrig, and David B Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless networks*, 11(1-2):21–38, 2005.
- [73] Yasir Hussain, Huang Zhiqiu, Muhammad Azeem Akbar, Ahmed Alsanad, Abeer Abdul-Aziz Alsanad, Asif Nawaz, Izhar Ahmed Khan, and Zaheer Ullah Khan. Context-aware trust and reputation model for fog-based iot. *IEEE Access*, 8:31622–31632, 2020.
- [74] Mohammed Hussein and Mohammad Zulkernine. Umlintr: a uml profile for specifying intrusions. In *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*, pages 8–pp. IEEE, 2006.
- [75] Stuart Jacobs. *Engineering information security: the application of systems engineering concepts to achieve information assurance*, volume 14. John Wiley & Sons, 2011.
- [76] Andreas Jacobsson, Martin Boldt, and Bengt Carlsson. A risk analysis of a smart home automation system. *Future Generation Computer Systems*, 56:719–733, 2016.
- [77] Magne Jørgensen. Top-down and bottom-up expert estimation of software development effort. *Information and Software Technology*, 46(1):3–16, 2004.
- [78] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.
- [79] Aisha Kanwal Junejo, Nikos Komninos, Mithileysh Sathiyarayanan, and

- Bhawani Shankar Chowdhry. Trustee: A trust management system for fog-enabled cyber physical systems. *IEEE Transactions on Emerging Topics in Computing*, 2019.
- [80] Jan Jürjens. *Secure systems development with UML*. Springer Science & Business Media, 2005.
- [81] Mohamad Kassab and Nil Kilicay-Ergin. Applying analytical hierarchy process to system quality requirements prioritization. *Innovations in Systems and Software Engineering*, 11(4):303–312, 2015.
- [82] Paula Katz and Cliff Campbell. Fda 2011 process validation guidance: Process validation revisited. *Journal of GXP Compliance*, 16(4):18, 2012.
- [83] Peter Kenning. The influence of general trust and specific trust on buying behaviour. *International Journal of Retail & Distribution Management*, 36(6):461–476, 2008.
- [84] BeomSeok Kim and Seokhoon Kim. An ahp-based interface and channel selection for multi-channel mac protocol in iot ecosystem. *Wireless Personal Communications*, 93(1):97–118, 2017.
- [85] Richard L Kissel, Kevin M Stine, Matthew A Scholl, Hart Rossman, Jim Fahlsing, and Jessica Gulick. Security considerations in the system development life cycle. Technical report, 2008.
- [86] Wolfgang Leister and Trenton Schulz. Ideas for a trust indicator in the internet of things. In *SMART*, volume 12, pages 31–34, 2012.
- [87] Michael Lesk. Safety risks—human error or mechanical failure?: Lessons from railways. *IEEE Security & Privacy*, 13(2):99–102, 2015.

- [88] Raph Levien and Alex Aiken. Attack-resistant trust metrics for public key certification. In *Usenix Security*, 1998.
- [89] Raphael L Levien. *Attack resistant trust metrics*. PhD thesis, University of California at Berkeley, 2002.
- [90] Nan Li, Vijay Varadharajan, and Surya Nepal. Context-aware trust management system for iot applications with multiple domains. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1138–1148. IEEE, 2019.
- [91] Katrina Ligett, Seth Neel, Aaron Roth, Bo Waggoner, and Steven Z Wu. Accuracy first: Selecting a differential privacy level for accuracy constrained erm. In *Advances in Neural Information Processing Systems*, pages 2566–2576, 2017.
- [92] Marcos V Linhares, Rômulo S de Oliveira, Jean-Marie Farines, and François Vernadat. Introducing the modeling and verification process in sysml. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 344–351. IEEE, 2007.
- [93] Torsten Lodderstedt, David Basin, and Jürgen Doser. Secureuml: A uml-based modeling language for model-driven security. In *International Conference on the Unified Modeling Language*, pages 426–441. Springer, 2002.
- [94] Malamati Louta and Angelos Michalas. Towards efficient trust aware e-marketplace frameworks. *Encyclopedia of E-Business Development and Management in the Global Economy*, 273, 2010.
- [95] Parikshit Mahalle, Sachin Babar, Neeli R Prasad, and Ramjee Prasad. Identity management framework towards internet of things (iot): Roadmap and key

- challenges. In *International Conference on Network Security and Applications*, pages 430–439. Springer, 2010.
- [96] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341. IEEE, 2015.
- [97] Stephen Marsh and Mark R Dibben. Trust, untrust, distrust and mistrust—an exploration of the dark (er) side. In *International Conference on Trust Management*, pages 17–33. Springer, 2005.
- [98] Stephen Paul Marsh. *Formalising trust as a computational concept*. PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1994.
- [99] Ilham Maskani, Jaouad Boutahar, and Souhail El Ghazi El Houssaïni. Modeling telemedicine security requirements using a sysml security extension. In *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 1–6. IEEE, 2018.
- [100] Fabio Massacci, John Mylopoulos, and Nicola Zannone. Security requirements engineering: the si* modeling language and the secure tropos methodology. In *Advances in Intelligent Information Systems*, pages 147–174. Springer, 2010.
- [101] Judith Masthoff. Computationally modelling trust: an exploration. In *Proceedings of the SociUM workshop associated with the User Modeling conference, Corfu, Greece, 2007*.
- [102] Orestis Mavropoulos, Haralambos Mouratidis, Andrew Fish, Emmanouil

- Panaousis, and Christos Kalloniatis. Apparatus: Reasoning about security requirements in the internet of things. In *International Conference on Advanced Information Systems Engineering*, pages 219–230. Springer, 2016.
- [103] Roger C Mayer, James H Davis, and F David Schoorman. An integrative model of organizational trust. *Academy of management review*, 20(3):709–734, 1995.
- [104] D Harrison McKnight and Norman L Chervany. The meanings of trust. *Technical Report MISRC Working Paper Series 96-04*, 1996.
- [105] D Harrison McKnight and Norman L Chervany. What is trust? a conceptual analysis and an interdisciplinary model. *AMCIS 2000 Proceedings*, page 382, 2000.
- [106] Daniel Mellado, Carlos Blanco, Luis E Sánchez, and Eduardo Fernández-Medina. A systematic review of security requirements engineering. *Computer Standards & Interfaces*, 32(4):153–165, 2010.
- [107] Keith W Miller and Jeffrey Voas. The metaphysics of software trust. *IT professional*, 11(2):52–55, 2009.
- [108] Keith W Miller, Jeffrey Voas, and George F Hurlburt. Byod: Security and privacy considerations. *It Professional*, 14(5):53–55, 2012.
- [109] Venus Mohammadi, Amir Masoud Rahmani, Aso Mohammed Darwesh, and Amir Sahafi. Trust-based recommendation systems in internet of things: a systematic literature review. *Human-centric Computing and Information Sciences*, 9(1):21, 2019.
- [110] JL Morrow Jr, Mark H Hansen, and Allison W Pearson. The cognitive and

- affective antecedents of general trust within cooperative organizations. *Journal of managerial issues*, pages 48–64, 2004.
- [111] Haralambos Mouratidis and Paolo Giorgini. Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(02):285–309, 2007.
- [112] Francisco Moyano, Carmen Fernandez-Gago, and Javier Lopez. A conceptual framework for trust models. In *9th International Conference on Trust, Privacy and Security in Digital Business (TrustBus 2012)*, volume 7449 of *Lectures Notes in Computer Science*, pages 93–104. Springer Verlag, Sep 2012.
- [113] Francisco Moyano Lara. *Trust engineering framework for software services*. PhD thesis, 2015.
- [114] MU Farooq Muhammad, Waseem Anjum, and Khairi Sadia Mazhar. A critical analysis on the security concerns of internet of things (iot). *International Journal of Computer Applications (0975 8887)*, 111(7), 2015.
- [115] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. A computational model of trust and reputation. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 2431–2439. IEEE, 2002.
- [116] Michele Nitti, Roberto Girau, and Luigi Atzori. Trustworthiness management in the social internet of things. *IEEE Transactions on knowledge and data engineering*, 26(5):1253–1266, 2014.
- [117] David Nuñez, C Fernandez-Gago, I Agudo, A Pannetrat, J Luna, S Berthold, S Pearson, M Felici, E Cayirci, A TaheriMonfared, et al. D: C-5.1 metrics for accountability. *Project Deliverable D*, 35.

- [118] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46. ACM, 2000.
- [119] L Obregon. Secure architecture for industrial control systems. *SANS Institute InfoSec Reading Room*, 2015.
- [120] Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini. Modelling and reasoning about security requirements in socio-technical systems. *Data & Knowledge Engineering*, 98:123–143, 2015.
- [121] Shantanu Pal, Michael Hitchens, and Vijay Varadharajan. Towards the design of a trust management framework for the internet of things. In *2019 13th International Conference on Sensing Technology (ICST)*, pages 1–7. IEEE, 2019.
- [122] Xi Yu Pang and Cheng Wang. The study of trust evaluation model based on improved ahp and cloud model in iot. In *Advanced Materials Research*, volume 918, pages 258–263. Trans Tech Publ, 2014.
- [123] Jorge Parra, M Anwar Hossain, Aitor Uribarren, Eduardo Jacob, and Abdulmotaleb El Saddik. Flexible smart home architecture using device profile for web services: A peer-to-peer approach. *International Journal of Smart Home*, 3(2):39–56, 2009.
- [124] Michalis Pavlidis. Designing for trust. In *CAiSE (Doctoral Consortium)*, pages 3–14, 2011.
- [125] Zhongmin Pei, Zhidong Deng, Bo Yang, and Xiaoliang Cheng. Application-oriented wireless sensor network communication protocols and hardware platforms: A survey. In *Industrial Technology, 2008. ICIT 2008. IEEE International Conference on*, pages 1–6. IEEE, 2008.

- [126] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. 2010.
- [127] Pawani Porambage, Mika Ylianttila, Corinna Schmitt, Pardeep Kumar, Andrei Gurtov, and Athanasios V Vasilakos. The quest for privacy in the internet of things. *IEEE Cloud Computing*, 3(2):36–45, 2016.
- [128] Stéphane Lo Presti, Michael Butler, Michael Leuschel, and Chris Booth. Holistic trust design of e-services. In *Trust in e-services: Technologies, practices and challenges*, pages 113–139. IGI Global, 2007.
- [129] Paul J Ready, Angappa Gunasekaran, and Alain Spalanzani. Bottom-up approach based on internet of things for order fulfillment in a collaborative warehousing environment. *International Journal of Production Economics*, 159:29–40, 2015.
- [130] Paul Resnick and Richard Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of ebay’s reputation system. *The Economics of the Internet and E-commerce*, 11(2):23–25, 2002.
- [131] Ruben Rios, Carmen Fernandez-Gago, and Javier Lopez. Privacy-aware trust negotiation. In *International Workshop on Security and Trust Management*, pages 98–105. Springer, 2016.
- [132] Ruben Rios, Carmen Fernandez-Gago, and Javier Lopez. Modelling privacy-aware trust negotiations. *Computers & Security*, 77:773–789, 2018.
- [133] Rodrigo Roman, Pablo Najera, and Javier Lopez. Securing the internet of things. *Computer*, 44(9):51–58, 2011.

- [134] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.
- [135] Yefeng Ruan and Arjan Durresi. A survey of trust management systems for online social communities—trust modeling, trust inference and attacks. *Knowledge-Based Systems*, 106:150–163, 2016.
- [136] Yefeng Ruan, Arjan Durresi, and Lina Alfantoukh. Trust management framework for internet of things. In *Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on*, pages 1013–1019. IEEE, 2016.
- [137] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.
- [138] Sini Ruohomaa and Lea Kutvonen. Trust management survey. In *International Conference on Trust Management*, pages 77–92. Springer, 2005.
- [139] Sini Ruohomaa, Lea Kutvonen, and Eleni Koutrouli. Reputation management survey. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 103–111. IEEE, 2007.
- [140] Thomas L Saaty. *Analytic hierarchy process*. Wiley Online Library, 1980.
- [141] Kazi Masum Sadique, Rahim Rahmani, and Paul Johannesson. Trust in internet of things: An architecture for the future iot network. In *2018 International Conference on Innovation in Engineering and Technology (ICIET)*, pages 1–5. IEEE, 2018.

- [142] Avani Sharma, Emmanuel S Pilli, Arka P Mazumdar, and MC Govil. A framework to manage trust in internet of things. In *Emerging Trends in Communication Technologies (ETCT), International Conference on*, pages 1–5. IEEE, 2016.
- [143] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146–164, 2015.
- [144] Dhananjay Singh, Gaurav Tripathi, and Antonio J Jara. A survey of internet-of-things: Future vision, architecture, challenges and services. In *Internet of things (WF-IoT), 2014 IEEE world forum on*, pages 287–292. IEEE, 2014.
- [145] Sachchidanand Singh and Nirmala Singh. Internet of things (iot): Security challenges, business opportunities & reference architecture for e-commerce. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 1577–1581. IEEE, 2015.
- [146] Saurabh Singh, Pradip Kumar Sharma, and Jong Hyuk Park. Sh-secnet: An enhanced secure network architecture for the diagnosis of security threats in a smart home. *Sustainability*, 9(4):513, 2017.
- [147] Rudolph Frederick Stapelberg. *Handbook of reliability, availability, maintainability and safety in engineering design*. Springer Science & Business Media, 2009.
- [148] Christoph Johann Stettina, Werner Heijstek, and Tor Erlend Fægri. Documentation work in agile teams: the role of documentation formalism in achieving a sustainable practice. In *2012 Agile Conference*, pages 31–40. IEEE, 2012.

- [149] Biljana L Risteska Stojkoska and Kire V Trivodaliev. A review of internet of things for smart home: Challenges and solutions. *Journal of Cleaner Production*, 140:1454–1464, 2017.
- [150] Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to industrial control systems (ics) security. *NIST special publication*, 800(82):16–16, 2011.
- [151] Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. Ahp-based quantitative approach for assessing and comparing cloud security. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, pages 284–291. IEEE, 2014.
- [152] Mohammad Gias Uddin and Mohammad Zulkernine. Umltrust: towards developing trust-aware software. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 831–836. ACM, 2008.
- [153] Domenico Ursino and Luca Virgili. An approach to evaluate trust and reputation of things in a multi-iots scenario. *Computing*, 2020.
- [154] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.
- [155] Fredrik Vraalsen, Mass Soldal Lund, Tobias Mahler, Xavier Parent, and Ketil Stølen. Specifying legal risk scenarios using the coras threat modelling language. In *International Conference on Trust Management*, pages 45–60. Springer, 2005.
- [156] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, and Peter Wolstenholme. *Modeling software with finite state machines: a practical approach*. Auerbach Publications, 2006.

- [157] Eric Ke Wang, Chien-Ming Chen, Dongning Zhao, Wai Hung Ip, and Kai Leung Yung. A dynamic trust model in internet of things. *Soft Computing*, pages 1–10, 2019.
- [158] David S Watson, Mary Ann Piette, Osman Sezgen, Naoya Motegi, and Laurie Ten Hope. Machine to machine (m2m) technology in demand responsive commercial buildings. 2004.
- [159] Marianne Winslett, Ting Yu, Kent E Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust in the web. *IEEE Internet Computing*, 6(6):30–37, 2002.
- [160] Zheng Yan and Silke Holtmanns. Trust modeling and management: from social trust to digital trust. *IGI Global*, pages 290–323, 2008.
- [161] Zheng Yan, Peng Zhang, and Athanasios V Vasilakos. A survey on trust management for internet of things. *Journal of network and computer applications*, 42:120–134, 2014.
- [162] Eric Yu and Lin Liu. Modelling trust for system design using the i* strategic actors framework. In *Trust in Cyber-societies*, pages 175–194. Springer, 2001.
- [163] Eric Siu-Kwong Yu. *MODELLING STRATEGIC RELATIONSHIPS FOR PROCESS REENGINEERING*. PhD thesis, University of Toronto, 1995.
- [164] Pamela Zave. Classification of research efforts in requirements engineering. In *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, pages 214–216. IEEE, 1995.