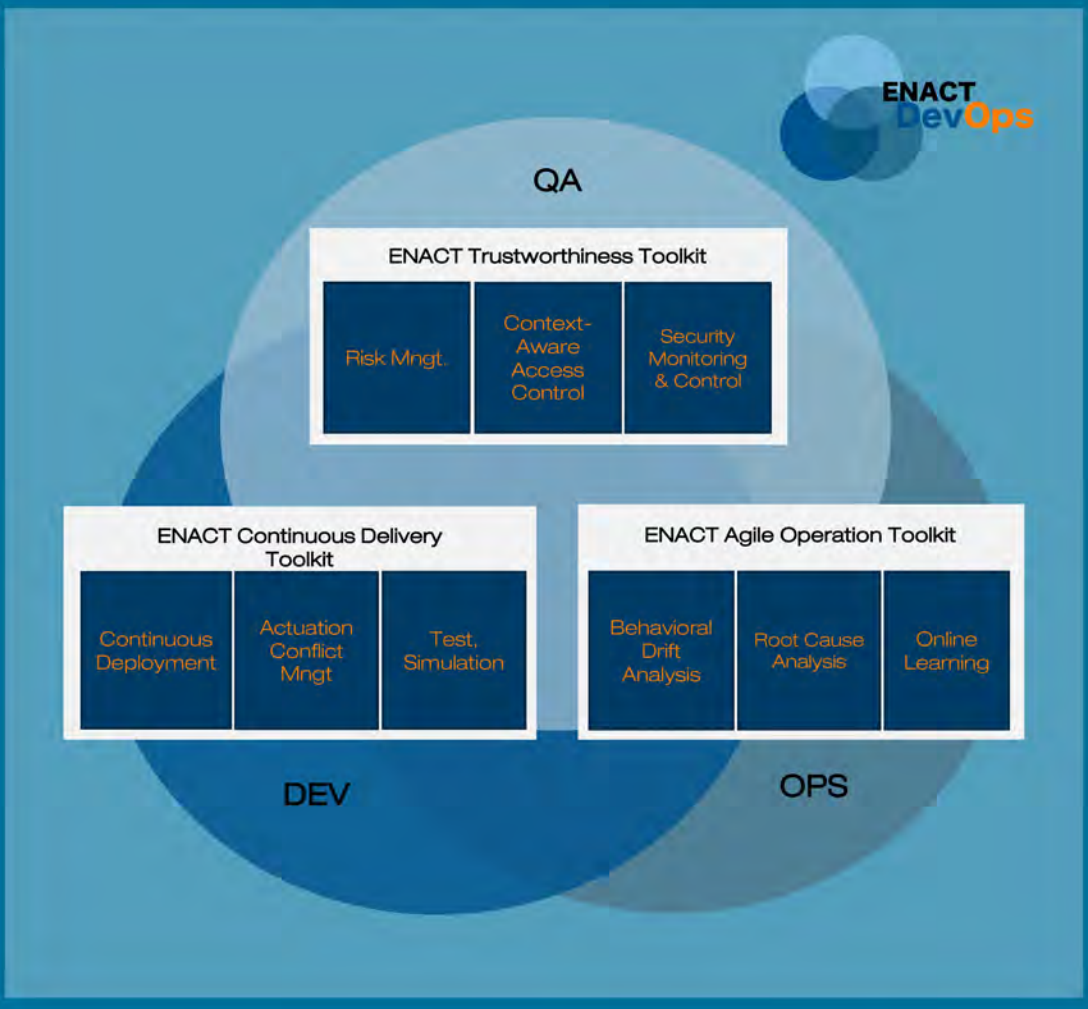
nowopen in Technology

DevOps for Trustworthy Smart IoT Systems

Nicolas Ferry, Hui Song, Andreas Metzger and Erkuden Rios (Editors)



now

DevOps for Trustworthy Smart IoT Systems

Nicolas Ferry, Hui Song
Andreas Metzger
and Erkuden Rios
(Editors)

now
the essence of knowledge

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

ISBN: 978-1-68083-824-4
E-ISBN: 978-1-68083-825-1
DOI: 10.1561/9781680838251

Suggested citation: Nicolas Ferry, Hui Song, Andreas Metzger and Erkuden Rios (eds.). (2021). *DevOps for Trustworthy Smart IoT Systems*. Boston–Delft: Now Publishers

Funded by the European Union (EU) as part of the Horizon 2020 program

Table of Contents

Foreword

Explosive growth in devices and software connected to the internet has led to current estimates of 46 billion "things" connecting and integrating on the internet. Increasing numbers of applications are being facilitated by the use of technology attached to the internet as part of this Internet of Things (IoT). The scale of IoT has inevitably led to the increasing complexity of any solution. Theoretical solutions must quickly give way to pragmatic technology. Smart IoT Systems (SIS) have a highly distributed infrastructure that relies on a wide range of topologies including, but not restricted to, cloud computing, edge computing, and closed or open networks. It requires tools and technology that are reliable, robust, adaptable and secure. Improvements in software, methodology, AI, data-analysis and decision-making are leading to practical applications that will improve the target systems in which they are implemented. The ENACT project indicates that there are many feature and functionality gaps in both the applications and enablers present in this environment and aims to close some of the significant gaps.

The ENACT project has been funded by the European Commission under its H2020 program. The project consortium consists of twelve member organisations spread across the EU as a whole. The funding has allowed the research and development of three toolkits covering trustworthiness, continuous development and agile operation. The emphasis of the toolkits is to build applications quickly and maintain those applications as the application target's circumstances change. The development of the toolkits has benefitted from the research excellence that has pervaded all stages of the project.

Proof of concept use cases will support the development and testing of the toolkits. Use cases that are significant in scope and demonstrate the challenges to be met by the project researchers have been chosen and are discussed in this book. The three

case study domains are in the domains of eHealth, Smart buildings and Intelligent Transport Systems.

In the context of eHealth both wellness and preventative medicine programs are being seen as the best direction for the ever-growing healthcare sector. Instead of reactive medicine proactive medicine is now the preferred approach supported by the use of technology to move beyond health and disease monitoring to a continuum of support for the healthy to maintain their wellbeing. This is individualised by the use of machine learning and activation of AI systems capable of making diagnostic forecasts based on data collected.

In the post Covid 19 world smart buildings will become more pervasive, providing the facilities to monitor not just temperature and light but air quality, security and general health and wellbeing of the building users. Air quality monitors should be able to detect any airborne viruses, extract them or direct the airflow away from users. Individualised warnings or cautions should ensure that only people who are vulnerable or who are possibly threatened by poor air quality can be warned or relocated.

Intelligent transport systems are slowly being developed that take advantage of IoT devices to monitor and manage areas of the transport environment that were not previously managed. Autonomous cars use a variety of sensing and analytical tools to create a world view in which they can operate. Rail services also need to understand the context in which they move. Failed signals or unhitched rolling stock can pose an immediate danger on the track. The physical infrastructure of a rail system uses resources that take time to be implemented but the human machine interface of equipment/driver/passenger/bystander is adapting all the time. Prediction of component or train integrity failures will ensure a more secure and safe solution. Adaptable systems will enhance delivery and business models.

Each of these case studies needs tools for developing applications quickly, ensure that the optimum infrastructure can be built, managed and maintained. The whole environment must be able to adapt to changes rapidly and required updates or new applications can be delivered quickly and simply integrated into the whole infrastructure. The Dev-Ops philosophy is one of the foundations of the project and will support rapid and agile development and continuous delivery that can be demonstrated in these case studies.

I have been researching and writing about innovation and solutions development for many years and am familiar with the pragmatic research and demonstrations that are part of the European Commission framework programs. I attended the ENACT project kick-off meeting and later as a member of the project advisory board attended project reviews. The project case studies are in domains that I am familiar with from my work on automation and collaborative robotics. The domain

specific problems in the use cases can be resolved with the general toolkits developed by the project and will aid decision making, development, deployment and maintenance. This book contains detailed and well written chapters that cover the most important areas of the project. The progress made in the project is excellent and I have been able to observe their progress over the project's duration. In my experience it is always difficult to manage such a large project, but the consortium has managed this well. This book has many interesting topics but I found that the book chapter on "Looking Ahead" was most interesting as it highlights future issues with integration and customisation of applications, an area that has been at the heart of my own work over many years. The authors clear view of what still remains to be done and the potential for further research is insightful.

Peter Matthews
31st March 2021

Glossary

A

ACM - *Actuation Conflict Management*. 9, 10, 18, 97, 98, 100–103, 111–114, 116–119, 226, 229, 238, 253–255

AGG - *Attributed Graph Grammar*. 97

AI - *Artificial Intelligence*. 152, 154, 170

API - *Application Programming Interface*. 69, 79, 83, 89, 146, 164, 165, 170, 194, 195, 250, 260

AVD - *Automatic Vulnerability Detector*. 26, 49

B

BDA - *Behavioral Drift Analysis*. 11, 18, 119, 137, 226, 229, 238–240, 256

C

C-ITS - *Cooperative Intelligent Transportation System*. 47, 49, 50, 52

CAAC - *Context-Aware Access Control*. 12, 162, 166

CAPEC - *Common Attack Pattern Enumeration and Classification*. 26, 44, 47, 49, 52

CMW - *Communication MiddleWare*. 230–233

CPS - *Cyber Physical System*. 16

CWE - *Common Weakness Enumeration*. 26, 44, 46, 47, 49, 52

D

DBN - *Dynamic Bayesian Networks*. 105

DevOps - *Development and Operation*. 2, 3, 6–20, 59, 60, 64, 67, 77–81, 83, 85, 86, 89, 95–99, 101, 103, 104, 109, 110, 118, 119, 170, 175, 188, 193, 195, 196, 198, 218, 221–223, 225–227, 229, 233–235, 237, 239, 240, 242, 259

DEVS - *Discrete EVent system Specification formalism*. 97

DevSecOps - *Development, Security, and Operation*. 59, 60, 78, 81, 84–86

DFD - *Data Flow Diagram*. 25, 26, 44–46

DMI - *Driver Machine Interface*. 232

DPIA - *Data Protection Impact Assessment*. 28, 29

DPO - *Data Protection Officer*. 31

DS - *Data Subject*. 24–43, 46, 47, 49, 50, 52, 53, 55

E

ECA - *Electronic Clearing Service*. 117

F

FSM - *Finite State Machines*. 101

G

GDPR - *General Data Protection Regulation*. 9, 20, 24–30, 36–44, 46, 49, 53–56

GNSS - *Global Navigation Satellite System*. 230, 231

H

HVAC - *Heating, Ventilation and Air-Conditioning*. 18, 123, 129, 130, 133, 136, 138, 252, 254

I

IaC - *Infrastructure as Code*. 60, 63

IAM - *Identity and Access Management*. 162

ICT - *Information and Communication Technology*. 64, 215, 216, 262

ID - *Identifier*. 49–54, 208

IDE - *Integrated Development Environment*. 79, 253

IEC - *International Electrotechnical Commission*. 1, 152

IOHMM - *Input/output Hidden Markov Model*. 105, 106, 108, 138

IOI - *Items Of Interest*. 38, 40, 42

IoT - *Internet of Things*. 1–4, 6–14, 16–18, 20, 23–25, 47, 54–56, 59–70, 78–84, 86, 89, 94–98, 102, 104, 110, 118, 123, 138, 139, 142–145, 147–162, 164, 165, 170, 171, 173–181, 183–185, 187, 188, 190, 192, 193, 199, 201, 207–209, 211, 212, 218, 221–223, 226, 228, 234–236, 240–244, 247–249, 252–257, 259, 264

ISO - *Information Systems and Organizations*. 152

IT - *Information Technology*. 52, 62, 161, 162, 170, 218

ITS - *Intelligent Transportation System*. 4, 17, 94, 95, 195, 199, 204, 232–237, 239

K

KDT - *Key Digital Technologies*. 259–264

KP - *Knowledge Processor*. 144–151, 157, 158, 160

L

LINDDUN - *Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of Information, Unawareness, and Non-compliance*. 24, 25, 27, 36–38, 42, 44–46, 49, 51, 56

M

MAC - *Media Access Control*. 235

MDE - *Model-Driven Engineering*. 61, 101, 117

MDP - *Markov Decision Process*. 128–130, 137

MFCC - *Mel-Frequency Cepstral Coefficient*. 112, 113

Mon&Con - *Monitoring and Control*. 152, 226, 232, 234

MQTT - *Message Queuing Telemetry Transport*. 110, 154, 155, 181, 183, 184, 188, 199, 204, 222, 248–250

MUSA - *MUlti-cloud Secure Applications*. 24

MVC - *Model – View – Controller*. 68

N

NAS - *Network Attached Storage*. 110

NIST - *National Institute of Standards and Technology*. 15, 31, 151

O

OLE - *Online Learning Enabler*. 11, 126, 257

OWASP - *Open Web Application Security Project*. 24, 25, 27–33, 49, 52–54

P

PDP - *Policy Decision Point*. 163

PDP4E - *Privacy and Data Protection Engineering*. 24, 44, 46, 49, 56

PEP - *Policy Enforcement Point*. 163

PHG - *Personal Health Gateway*. 19, 20, 221, 223

PII - *Personally Identifiable Information*. 16

R

RCA - *Route Cause Analysis*. 12, 13, 96, 175, 198–212, 227, 239, 240

REST - *Representational State Transfer*. 164, 194, 250

RFID - *Radio Frequency Identification*. 230–232

RL - *Reinforcement Learning*. 123–138

S

S&P - *Security and Privacy*. 152, 226, 232–235, 240

SCA - *Software Communications Architecture*. 62, 63

SDR - *Software-Defined Radio*. 62

SIB - *Semantic Information Broker*. 144

SIS - *Smart IoT System*. 1–4, 6–12, 14–20, 23, 24, 47, 59–61, 65–67, 70, 71, 78–81, 89, 94–97, 99, 101–108, 110, 112–114, 118, 119, 123, 124, 129, 138, 142, 143, 148, 149, 151–156, 159–161, 170, 171, 177–180, 183, 184, 190, 196, 198, 199

SLR - *Systematic Literature Review*. 63

SMT - *Satisfactory Modulo Theories*. 64, 65

SSAP - *Source Service Access Point*. 144, 145

T

TaS - *Test and Simulation*. 10, 18, 174–181, 183–185, 187–197, 211, 212

TBM - *Transferable Belief Model*. 105

TMS - *Train Management Systems*. 226

U

UML - *Unified Modelling Language*. 44

V

VPN - *Virtual Private Network*. 51

W

WAM - *Web Access Manager*. 162, 167–170

WIMAC - *Workflow and Interaction Model for Actuation Conflict management*. 97

WSAN - *Wireless Sensor Actuator Network*. 229

WSN - *Wireless Sensor Network*. 195, 226, 230–232

WTI - *Wireless Train Integrity*. 225, 227, 229

X

XACML - *eXtensible Access Control Markup Language*. 162–164

Z

ZCR - *Zero Crossing Rate*. 113

DOI: 10.1561/9781680838251.ch1

Chapter 1

Introduction

By Erkuden Rios, Nicolas Ferry, Hui Song and Andreas Metzger

Internet of Things (IoT) systems are evolving towards what we denote as Smart IoT Systems (SIS) – *i.e.*, systems involving not only sensors but also actuators with control loops distributed all across the IoT, Edge and Cloud infrastructure.

However, the capacity in building novel and innovative SIS faces specific challenges, that entail (i) how to efficiently build and operate new value-added software across IoT, edge and cloud infrastructures, (ii) how to close the loop of sensing and actuation, and (iii) how to establish trustworthiness in these systems. Whilst point (iv) is already critical in classical IoT systems: according to the IEC report on smart and secure IoT platforms [5], security, trust, privacy and identity management are major challenges in today's IoT systems, this is all the more exacerbated when actuators are involved.

One of the fundamental research questions concerning these issues is: "how can we tame the complexity of developing and operating smart IoT systems, which

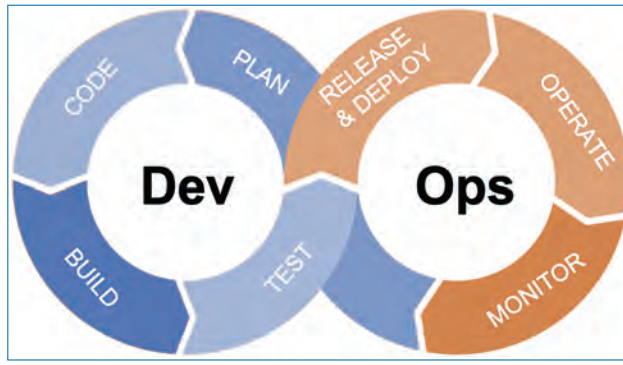


Figure 1.1. The generic DevOps life-cycle model.

(i) consist of software running on all types of resources along the IoT-edge-cloud continuum, (ii) involve sensors and actuators and (iii) need to be trustworthy?". The answer to these questions has formed the core of the Horizon 2020 project ENACT [1, 3]. The overall ambition of ENACT was to expand current DevOps methods and solutions to support the development and operation of trustworthy Smart IoT Systems.

DevOps has established itself as a software development life-cycle model that encourages developers to continuously patch, update, or bring new features to the system under operation without sacrificing quality [8]. By enabling DevOps in the realm of SIS, ENACT not only facilitates the development and operation of SIS but also enables the continuous and agile evolution of SIS, which is necessary to adapt the system to changes in its environment, including such as newly appearing trustworthiness threats. ENACT supports DevOps practices during the development and operation of trustworthy smart IoT systems by offering software tools, called "enablers", for each of the seven stages of the DevOps life-cycle model as depicted in Figure 1.1.

- **Plan:** ENACT supports privacy and security risk assessment enabling the risk-driven planning of IoT systems development cycles as well as the smooth transition towards the code stage.
- **Code:** First, ENACT evolves recent advances of the ThingML language and generators to support modelling of system behaviours and generation of code executable across the whole IoT, edge and cloud continuum. Second, ENACT provides a model-based solution to automatically identify and solve conflicts when multiple applications manage actuators.
- **Test:** Targeting the constraints related to the distribution and infrastructure of IoT systems, ENACT enables continuous testing of SIS in an environment by emulating and simulating IoT and Edge infrastructures.

- **Release and Deploy:** ENACT provides novel deployment modelling languages and the corresponding execution engines to support the continuous and automatic fleet deployment, by assigning multiple deployments to many devices in the fleet, without human interaction. It enables deployment from the IoT to the cloud ends with security as a first-class concern.
- **Operate:** ENACT provides enablers for the automatic adaptation of IoT systems based on their run-time context, including smart preventive security mechanisms such as access control. In addition, ENACT offers machine learning capabilities at runtime in order to deliver self-adaptive SIS. Such automatic self-adaptation addresses the issue that the management complexity of open-context IoT systems exceeds the capacity of human operation teams, and by this, improve the trustworthiness of the smart IoT system execution.
- **Monitor:** ENACT has delivered innovative mechanisms to observe and analysis (i) the status of a SIS including security and privacy aspects at all the network, system, and application levels, (ii) failures, (iii) the overall effectiveness of the SIS in reaching its goals.

ENACT was part of a cluster of related H2020 projects all contributing to IoT security [2]. Among the eight projects that formed the cluster, two are most notably related to ENACT and share common objectives. The Semiotics project[1] also considers SIS with a specific focus on the management of actuators. The project proposes a pattern-driven framework, built upon existing IoT platforms, to enable and guarantee secure and dependable actuation and semi-autonomic behaviour in IoT applications. While not specifically focusing on DevOps, one of the technical objectives of the Brain-IoT project[2] was to facilitate the rapid model-based development, integration, and deployment of interoperable IoT solutions that support smart cooperative behaviour involving actuation in IoT scenarios.

This book describes the ENACT project outcomes (cf. Figure 1.2) and how they solve major challenges in the DevOps of trustworthy SIS. The overall approach pursued in the ENACT project is introduced in Chapter 2, and the chapters following afterwards detail the outcomes of ENACT. In Chapter 3 the privacy and security risk assessment and management in SIS is discussed, and the ENACT enabler dealing with risks is presented. Chapter 4 is focused on deployment support offered by ENACT and the deployment and diversification methods and enablers are detailed therein. Chapter 5 deals with the issues of actuation conflict resolution in SIS that

1. https://www.semiotics-project.eu
2. http://www.brain-iot.eu

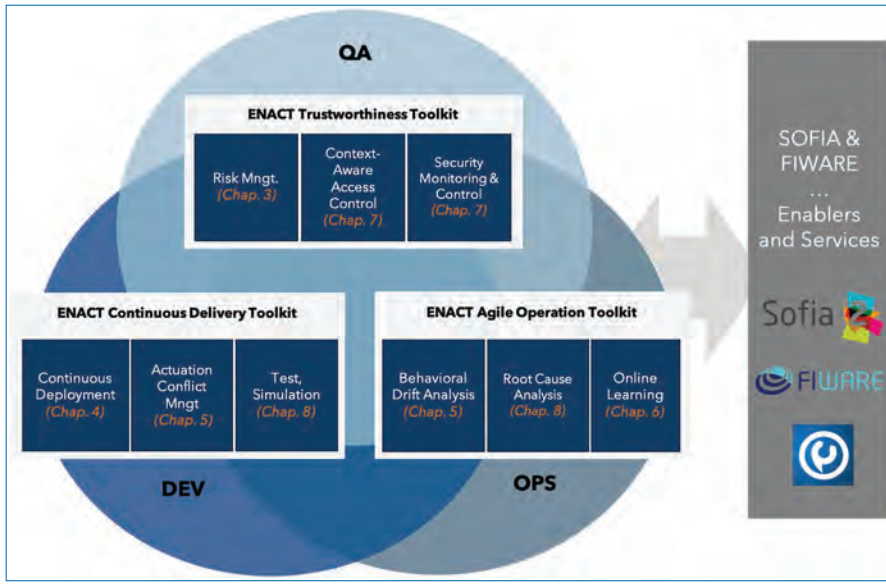


Figure 1.2. ENACT project results.

include actuators, and how to detect and analyse behaviour deviations at SIS operation from those designed when building the SIS. In Chapter 6 reinforcement learning techniques are studied as the ENACT approach for continuously ensuring and improving the quality of SIS during operations. Chapter 7 explains all the details of the ENACT support to security aspects of SIS, including context-aware access control enabler, security monitoring enabler, as well as security control through capabilities embedded in IoT platforms. Chapter 8 describes the ENACT enablers dedicated to the SIS verification and validation activities, including the support to testing, simulation and root cause analysis. Chapters 9, 10 and 11 explain the real IoT system use cases where the ENACT enablers were validated, dedicated to eHealth, Intelligent Transport Systems (ITS) and Smart Buildings domains, respectively. Chapter 12 concludes the book with an outlook on future research challenges and opportunities.

References

[1] ENACT Consortium. *ENACT: Development, Operation, and Quality Assurance of Trustworthy Smart IoT Systems*, 2018. URL: https://cordis.europa.eu/project/id/780351.

[2] Enrico Ferrera *et al.* "IoT European security and privacy projects: Integration, architectures and interoperability. In: *Next Generation Internet of Things. Distributed Intelligence at the Edge and Human Machine-to-Machine Cooperation* (2018).

[3] Nicolas Ferry *et al.* "ENACT: Development, operation, and quality assurance of trustworthy Smart IoT Systems". In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment.* Springer. 2018, pp. 112–127.

[4] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010. ISBN: 860-1401501176.

[5] IEC. *IEC White Paper: IoT 2020: Smart and secure IoT platform*. IEC report, 2019. URL: https://basecamp.iec.ch/download/iec-white-paper-iot-2020-smart-and-secure-iot-platform/.

DOI: 10.1561/9781680838251.ch2

Chapter 2

The ENACT Approach

By Nicolas Ferry, Hui Song, Erkuden Rios and Andreas Metzger

Smart IoT Systems (SIS) are the next generation of IoT systems that span across the complete computing continuum, from IoT via Edge/Fog to the Cloud, with local data analytics, decision making, and actuators involved. Software plays a key role in such systems. The systems' increased complexity, the unpredictability of their environment, as well as the changes in their requirements and infrastructure are many factors that can result in new threats hindering their trustworthiness. The proper functioning and correctness of such systems is critical especially when they control actuators that can have a direct impact on the physical world. The ability of these systems to continuously evolve and adapt to these changes is decisive to ensure and increase their trustworthiness, quality and user experience. Currently, DevOps is the mainstream practice in the software and Cloud industry to foster continuous

evolution of software systems. DevOps promotes a rapid and efficient value delivery to the market, through a tight collaboration between the developers and the teams that deploy and operate the software systems. DevOps seeks to decrease the gap between product design and its operation by introducing software design and development practices and approaches to the operation domain and vice versa [8].

When the ENACT project was created in 2017, there was no DevOps support for trustworthy Smart IoT Systems [12, 34]. Even if DevOps is not bound to any application domain, many challenges appear when the IoT intersects with DevOps, in particular, due to the lack of key enabling tools. ENACT focused specifically on the following three challenges [3].

The first key challenge, as opposed to Cloud environments which are relatively reliable and homogeneous, is the wide diversity that characterizes SIS, not only in terms of hardware but also in terms of their software stack. There is typically a lack of coherent languages, abstractions, security and privacy solutions that can be used to support development and the orchestration of software and their deployment across heterogeneous devices.

Second, SIS are by nature massively distributed on top of a highly heterogeneous and geographically-distributed infrastructure, which means that software is more complex to apprehend, develop, operate, and maintain than on top of Cloud infrastructures. Each device has a unique operational context, in terms of hardware capacity, end-user preference, exposure to security risks, role in the whole data flow, connection to sensors and actuators, etc. This context is dynamic and often unpredictable, *e.g.*, the volume of data may change, the network connectivity among devices can be unstable. Therefore, the management and operation of each software module, *e.g.*, where to place it, when to deploy it, how to configure it, and how to monitor it, etc, needs to be handled individually and continuously to fit its unique and evolving context. For large scale SIS which can include thousands of devices, handling each device individually inevitably leads to enormous operational effort and cost, which, as identified by Gartner, "can easily exceed the project's financial benefits."[1]

Third, SIS can have an impact on the physical world through actuators. There is a need to properly manage these actuators and to ensure that such systems and in particular the software deployed on these systems always work within safe operational boundaries. Only a few approaches exist in the literature focusing on the management of actuation conflicts, and none are meant to be used in a DevOps context.

1. https://www.gartner.com/en/newsroom/press-releases/2018-12-03-gartner-says-the-future-of-it-infrastructure-is-always-on-always-available-everywhere

These key challenges had to be addressed to enable DevOps for trustworthy Smart IoT Systems. In this chapter we present how the overall approach followed in the ENACT project proposes to evolve existing DevOps methods and techniques to support the development and operation of Smart IoT Systems, which (i) are distributed, (ii) involve sensors and actuators and (iii) need to be trustworthy (*i.e.*, trustworthiness refers to the preservation of security, privacy, reliability, resilience, and safety [13]).

The remainder of the chapter is organized as follows. Section 2.1 introduces the overall ENACT approach and lists the enablers that will form the core contribution of ENACT supporting the DevOps of SIS. Section 2.1.4 details how these enablers can be organized together to form a comprehensive and continuous DevOps Framework. Section 2.2 summarizes how the developed solutions facilitate the development and operation of SIS that are trustworthy. Finally, Section 2.3 reports on the three use cases of the project and how they supported validation of the ENACT enablers.

2.1 ENACT Enablers to Deliver DevOps for SIS

To foster the adoption of DevOps practices in the realm of SIS, ENACT's approach is to deliver a set of enablers (*i.e.*, tools and services) that support the continuous development, evolution and operation of SIS. These enablers are designed to integrate with DevOps, Cloud, and Edge services and are loosely coupled, providing SIS providers with the ability to pick the enablers that best fit their needs. In other words, it is not necessary to gather the whole ENACT framework to benefit from one or more of these enablers, and these can be integrated as part of existing DevOps pipelines. As depicted in Figure 2.1, the different enablers contribute to different stages within the DevOps life-cycle and, overall, the ENACT Framework contributes to all the DevOps stages. In the following we provide an overview for each of the enablers, which are detailed in the remaining chapters of this book.

2.1.1 Enablers for the Development Phase

The following three enablers provide specific support for the development of trustworthy SIS.

Risk Management enabler: This enabler provides concepts and tools for the agile, context-aware, and risk-driven decision support and mechanisms for application developers and operators to support the continuous delivery of trustworthy SIS [12]. By leveraging the evidences collectors provided by the enabler,

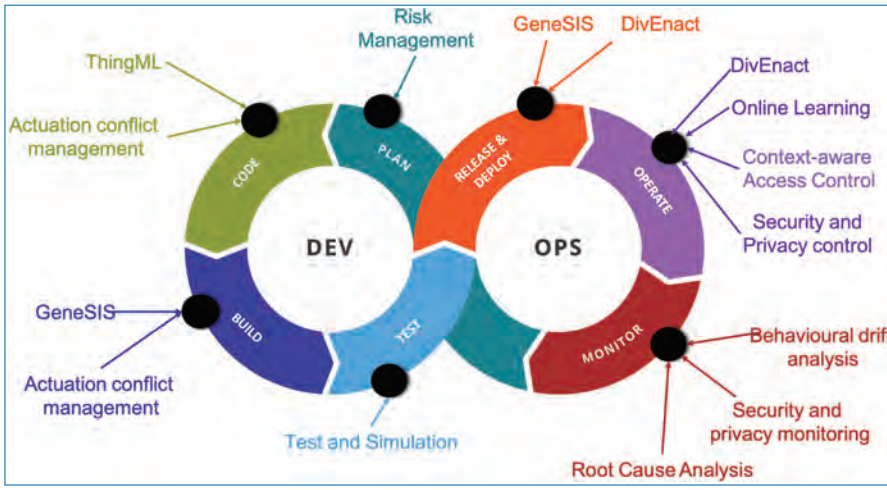


Figure 2.1. Contribution of the ENACT tools to the DevOps lifecycle.

organizations use it not only to assess risks but also to monitor and control treatment implementation and effectiveness during the development and operation of SIS, enabling the treatment of security and privacy risks together and making them actionable for software engineers. This makes this enabler the first DevOps-enabled continuous risk control solution, improving software development and operation organizations' awareness on risks. In addition, it facilitates compliance with standards such as ISO 27001 and regulations such as GDPR, in near real-time. Further details about this enabler can be found in Chapter 3.

ThingML: ThingML [4] is an open source IoT framework that includes a language and a set of generators to support the modelling of system behaviours and their automatic derivation across heterogeneous and distributed devices at the IoT and edge end. The ThingML code generation framework has been used to generate code in different languages, targeting around 10 different target platforms (ranging from tiny 8 bit microcontrollers to servers). A challenge for approaches such as ThingML is how to properly log, monitor and debug the generated programs. Indeed, to fully benefit from the approach, such logging should be performed by relating to the concepts of the original abstraction level. To address this challenge, ThingML has been extended with an automated,platform-independent and easy to use logging mechanism to ThingML developers. This logging approach aims at providing log information about the execution of their ThingML programs, in terms of ThingML concepts being executed.

Actuation Conflict Management enabler (ACM): Actuation conflicts can occur when concurrent applications have a shared access to an actuator and when actuators produce actions within a common physical and local environment, whose

effects are contradictory. This enabler supports the identification and resolution of direct and indirect actuation conflicts as part of a DevOps pipeline in a platform independent and technology agnostic way [6]. DevOps team can integrate the ACM solution as part of their DevOps pipeline to detect automatically direct and indirect actuation conflicts in a complex SIS. Off-the-shelf actuation conflict managers are automatically injected into the SIS. New actuation managers can be designed using a tool-supported domain-specific modelling language and checked against logical and temporal properties. While traditionally, the management of actuation conflicts is handled at the code level, the ACM enabler applies over and abstract representation of the SIS that is decoupled from its detailed code enabling the detection, analysis and resolution of actuation conflicts as part of a typical DevOps process. Verification mechanisms ensure the conflict management solution injected into the SIS satisfies temporal and logical properties making DevOps teams confident to place it in the system. Further details about this enabler can be found in Chapter 5.

Test and Simulation enabler (TaS): Software testing is a crucial step of any software development process, especially in DevOps. Having access to a production-like environment that reproduces the same conditions where a piece of software would run is usually tricky or close to being an impossible task. This is exacerbated in IoT environments where (i) developers need to test their applications to ensure trustworthiness requirements, including scalability, are met, and (ii) building a large-scale testbed that includes a realistic physical infrastructure of devices and sensors can quickly be expensive. The test and simulation enabler provides a light-weight, user-friendly approach for simulating large number of IoT devices and cyber-attacks, in order to set up the testing environment and test SIS in a cost-effective way. The enabler goes beyond the state of the art on sensor and actuation simulation solutions that typically reproduce how the devices behave according to the physical environment. Instead, it focuses on the pure software simulation, by reproducing how devices interact with software in the IoT system.

2.1.2 Enablers for the Deployment Phase

Within the DevOps life-cycle, deployment is typically the activity that bridges development and operation activities. The following two enablers provide specific support for the deployment of trustworthy SIS.

Orchestration and Continuous Deployment enabler (GeneSIS): This enabler, also known as GeneSIS, supports the automatic deployment of software, together with the attached security mechanisms, across the computing continuum from IoT, Edge to Cloud [20]. Developers use a declarative modelling language to specify

what software components and security mechanisms they want to deploy, and the engine automatically deploys them into the resources in the computing continuum, continuously monitoring the deployment status. The GeneSIS modelling language is independent of the underlying technologies, *i.e.*, GeneSIS can deploy components anywhere in the IoT-Edge-Cloud continuum: from microcontrollers without direct Internet access to virtual machines running in the Cloud. It also includes security mechanisms as first-class modeling elements thus promoting security-by-design. Further details about this enabler can be found in Chapter 4.

Fleet Management and Diversity enabler (DivEnact): This enabler, also known as DivEnact, supports automatic software deployment for IoT applications that comprise a large fleet of devices, and maintains software diversity among the fleet [44]. It provides DevOps teams with a mean to deploy a new software version into the abstract fleet, without worrying about what exact devices are in the fleet, their contexts, and whether they are online or not. DivEnact maintains the devices and their contexts in the fleet, the software variants, and assign the variants to the appropriate devices depending on their contexts. Further details about this enabler can be found in Chapter 4.

2.1.3 Enablers for Operation Phase

Finally, the following four enablers focus on supporting the operation and monitoring of SIS.

Behavioral Drift Analysis enabler (BDA): The complex nature of the cyber-physical environment in which a SIS operate makes it impossible for DevOps teams to predict if, once under operation, the system will behave as expected during development. For instance, many unanticipated surrounding physical processes may disrupt and hamper the SIS from achieving its goal. The Behavioral Drift Analysis enabler provides a novel way to overcome this issue by shifting the monitoring and analysis from the internal of the system to its context by observing and analysing the effects of the commands sent to the actuators on the cyber-physical context of the SIS [29]. This makes the approach generic an applicable to any SIS independently of its implementation and it makes it non-invasive in the sense that it does not require any modification of the applications. DevOps teams can use this enabler during operation as a monitoring solution to detect symptoms indicating that the effects of the system on its environment are no longer as expected and to understand this loss of effectiveness. Further details about this enabler can be found in Chapter 5.

Online Learning enabler (OLE): To develop a self-adaptive SIS, software engineers have to create self-adaptation logic encoding when the SIS should execute

which adaptation actions. However, developing self-adaptation logic may be difficult due to design time uncertainty; *e.g.*, anticipating all potential environment changes at design time is, in most cases, infeasible. In addition, due to simplified design assumptions, the precise effect of an adaptation action may not be known, making it difficult to accurately determine how the SIS should adapt itself. The Online Learning Enabler addresses these challenges by leveraging modern machine learning algorithms during the operation phase [16]. In particular, the enabler uses reinforcement learning to address design time uncertainty by learning suitable adaptation actions through interactions with the environment at run time.

Security and Privacy Control and Monitoring enabler (S&P): This enabler is a one-stop solution for the near real-time monitoring and control of security- and privacy-related anomalies across multiple layers of Smart IoT Systems, from things, devices, Edge to Cloud. DevOps teams use the S&P enabler for controlling data protection and secure communications all along the lifecycle of the SIS, through continuous monitoring of security metrics, and automatic detection and feedback for subsequent DevOps loops. The enabler uses machine learning to correlate data captured by multiple probes or monitoring agents deployed in different layers, in order to offer a holistic view of the SIS and enable the detection of sophisticated attacks. The tool has a flexible architecture to adapt to the different information availability and the specific types of anomalies, and is fully elastic for the rapid scaling of the target systems. Further details about this enabler can be found in Chapter 6.

Context-Aware-Access Control (CAAC): Context-Aware-Access Control provides a unified access control of all the IoT actors from administrators, end-users, and services, to devices, and dynamically adapts the authorization according to the changing context [23]. It is a SaaS solution that can be integrated into the IoT applications, and provides a user-friendly authentication interface. The solution ensures the data is only exposed to authorized users and devices. It supports the applications in adapting the authorizations according to context changes, without requiring developers to modify the code. This is done by adding dynamicity to the OAuth 2.0 standard protocol to make the provided authorizations responsive to the context, injecting contextual risk levels as dynamic attributes in the authorization mechanisms. Further details about this enabler can be found in Chapter 6.

Root Cause Analysis enabler (RCA): Understanding the origin of a failure in a SIS is a complex and time consuming task. This is in particular due to the fact that these systems are large, vastly heterogeneous as well as widely distributed. This enabler observes the symptoms of the IoT systems, such as loss of messages, delay

of response, etc. and automatically diagnoses the root cause, such as device failures or broken networks. DevOps teams can thus use the RCA enabler during the operation of their IoT application in order to receive alarms when there are incidents. The alarms will include details about the origin and possibly the reason of the accident as well as targeted instruction about how to fix the incidents. Instead of relying on human experts to exhaust all the causal connections between incidents and symptoms, the Root Cause Analysis tool builds this knowledge itself, by recording the typical incidents and their symptoms. During runtime, it compares the similarity between the observed symptoms with the recorded ones in the library to identify the possible incidents. Further details about this enabler can be found in Chapter 6.

2.1.4 Focus of the Different Enablers

In real cases, a large scale IoT system usually comprises many duplicates of the same or similar sub-systems, which contain a relatively smaller number of nodes. A typical example is the eHealth use case (see Chapter 9). In the eHealth use case, a remote patient monitoring system aims at supporting thousands of patients, and each patient is provided with a sub-system that includes one gateway and several sensors. These sub-systems are similar to each other, in terms of architecture, software and configurations. Under such setups, the DevOps of Smart IoT Systems usually includes two complementary activities: (i) the development, testing and optimization of the functionality within one sample sub-system, and (ii) the operation of the system of systems, with many duplicates of the sample sub-system.

The ENACT enablers naturally have different focuses. While the Risk Management, test and simulation, security and privacy control enablers and DivEnact are solutions that can be used at the system-of-systems level, the other enablers are aimed for the sub-system level. Yet, it is worth noting that the tools that system-of-systems enablers can also be applied at the scale of one sub-system.

2.2 Architecture of the ENACT Framework

The set of ENACT enablers introduced above form the ENACT DevOps Framework. Below we detail the architecture of this framework as well as the relationships between the enablers within this framework.

Figure 2.2 depicts the overall architecture of the ENACT Framework. It is a multi-layer architecture composed of 4 layers hierarchically organized plus one crosscutting layer. In the following we detail each of these layers. It is worth noting that Figures 2.1 and 2.2 are complementary in explaining the relationships

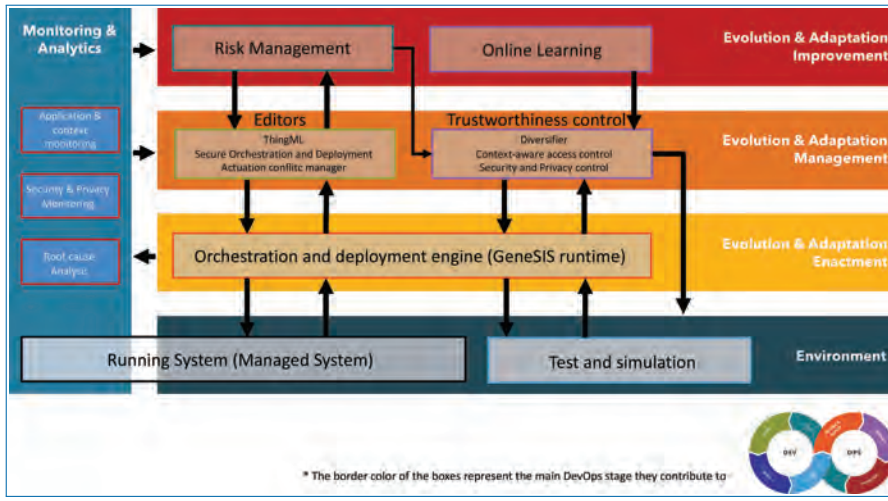


Figure 2.2. The ENACT architecture.

and complementarity of the enablers. The former details the contribution of the enablers within a DevOps pipeline while the later details how they can be integrated within a comprehensive Framework for the development and operation of SIS.

From the most abstract to the most concrete (*i.e.*, from the farthest to the closest to the running system), the layers are described as follows:

1. **Evolution & Adaptation Improvement Layer:** This layer provides the mechanisms to continuously improve and manage the development and operation processes of trustworthy SIS. On the one hand, the Risk Management enabler helps organizations to analyze the architecture of their Smart IoT Systems and detecting potential vulnerabilities and the associated risk (in particular related to security and privacy aspects) and propose related mitigation actions. On the other hand, the Online Learning enabler focuses on improving the behaviour of the adaptation engine that will support the operation of trustworthy SIS. This tool typically relates to the Operate stage of the DevOps process. In general, the improvement layer provides feedback and knowledge to all the other DevOps stages with the aim to improve the development and operation of trustworthy SIS. Thus, in this architecture, information from this layer are provided to the evolution and adaptation management layer with the aim to improve it.
2. **Evolution & Adaptation Management Layer:** This layer first embeds a set of editors to specify the behaviours as well as the orchestration and deployment of SIS across IoT, Edge and Cloud infrastructure. These editors integrate with mechanisms to maximize and control the trustworthiness of the

system. All together, these components cover activities in both the Dev and Ops parts of a DevOps process and in particular to the code, build and operate stages. The activities performed at this layer are strongly affected by the inputs from the improvement layer.

3. **Evolution & Adaptation Enactment Layer:** This layer bridges the gap between development and operation as its goal is to enact the deployment and adaptation actions decided at the Evolution & Adaptation Management Layer. The mechanisms of this layer monitor and manage the deployment of the running system.
4. **Environment Layer:** This layer consists of the running system together with the environment and infrastructure in which it executes. This includes both production and testing environments.
5. **Monitoring and Analytics Layer:** This layer is orthogonal and feeds the other four. The enablers at this layer are supporting the monitoring stage of the DevOps process and typically aim at providing feedback from Ops to Dev. More precisely, this layer provides mechanisms to monitor the status of the system and of its environment. This includes mechanisms to monitor the security and privacy of a SIS. In addition, it performs analytic tasks providing: (i) high level notifications with insights on ongoing security issues, (ii) diagnostics and recommendations on system's failures, and (iii) feedback on the behavioural drift of SIS (*i.e.*, system is functioning but not delivering the expected behaviour).

2.3 Improving SIS Trustworthiness

In this section we first summarize the contributions of the enablers in terms of supporting the development and operation of trustworthy SIS.

Based on the NIST definition of trustworthiness for Cyber Physical Systems [13], within ENACT, we adopt the following definition of trustworthiness and its different properties: "Trustworthiness refers to the preservation of security, privacy, safety, reliability, and resilience of SIS".

We adopt the following definitions of the different properties:

- **Security** refers to the preservation of confidentiality, integrity and availability of information [9].
  - **Integrity** is the property of protecting the accuracy and completeness of information [1].
  - **Confidentiality** is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [1].

  - **Availability** is the property of information being accessible and usable upon demand by an authorized entity [1].
- **Privacy** refers to the protection of personally identifiable information (PII) [10]. PII refers to any information that (a) can be used to identify the PII principal to whom such information relates, or (b) is or might be directly or indirectly linked to a PII principal.
- **Safety** refers to the ability of the cyber-physical system (CPS) to ensure the absence of catastrophic consequences on the life, health, property, or data of CPS stakeholders and the physical environment [13].
- **Reliability** refers to the ability of the CPS to deliver stable and predictable performance in expected conditions [13].
- **Resilience** refers to the ability of the CPS to withstand instability, unexpected conditions, and gracefully return to predictable, but possibly degraded, performance [13].

Figure 2.3 summarizes how each individual ENACT enabler contributes to the development and operation of trustworthiness of SIS. It is also worth noting that the support offered by the ENACT enablers to the DevOps of SIS is, by itself, a major contribution for supporting the trustworthiness aspect. Indeed, the adoption of the DevOps principles and practices in the field of the IoT is decisive to enable the continuous and agile evolution of SIS, which is necessary to adapt the system to newly appearing trustworthiness threats and to ensure its overall quality.

Some enablers are marked as indirectly contributing to the privacy property. This is because the support for security provided by these enablers also contributes

Enabler	Security	Privacy	Reliability	Resilience	Safety
ACM	No.	No.	Yes. To ensure the reliability of the system embedding actuation conflict managers, they are checked against logical and temporal properties. In addition, it helps solving the case when a SIS does not behave as expected due to indirect actuation conflict.	No.	Yes. The management of actuation conflicts can have a direct impact on the safety of the system (i.e., unmanaged conflicts can lead to safety issues)
BDA	No.	No.	No. Because drift should appear only under unexpected conditions, does not focus on performances	Yes. By monitoring and analyzing drifts to trigger a new development cycle.	No.
CAAC	Yes. Confidentiality: ensure proper access control to the data.	Yes. Ensure privacy in the data managed by SIS, since the control over the personal data is kept by their owner	Yes. Information is made available or disclosed only to authorized individuals, entities, or processes, in a controlled way.	No.	No.
DivEnact	Yes. By diversifying software at the Edge, we implement a "moving target" defence strategy.	No.	Yes. Diversifying software can lead to an increase in their reliability.	Yes. Support failure recovery in a DevOps process	No.
GemSIS	Yes. Support specification of security requirements and the deployment of security mechanisms.	Yes. Indirectly via security. Can be used to deploy privacy mechanisms	Yes. Via mechanisms such as blue/green deployment.	No. Not directly, only by supporting adaptation	No.
Online Learning (OLE)	No.	No.	Partially. OLE focuses at system operations under aspects unexpected conditions. However, the explainable AI element of OLE facilitates that online learning itself happens in a reliable fashion.	Yes. Adaptation and improvement of adaptation logic to keep working in unseen environments	No.
RCA	Yes. To detect the cause of security flaws	Yes. Indirectly via security	Yes. Root cause of previously known performance problems	Yes. Root cause of previously unknown performance problems supported by ML techniques	No.
Risk Management Enabler	Yes. Confidentiality, Integrity and Availability can be provided depending on the vulnerabilities defined, risks identified, and mitigation actions proposed by the user or our knowledge bases.	Yes. Predictability, Manageability, Disassociability can be provided depending on the vulnerabilities defined, risks identified, and mitigation actions proposed by the user or our knowledge bases.	Yes. It can be provided depending on the vulnerabilities defined, risks identified, and mitigation actions proposed by the user or our knowledge bases	Yes. It can be provided depending on the vulnerabilities defined, risks identified, and mitigation actions proposed by the user or our knowledge bases	Yes. It will be taken into consideration through the analysis of the impact of risks.
S&P Monitoring and control	Yes. Monitors and controls confidentiality, integrity and availability of data	Yes. Monitors and controls confidentiality, integrity and availability of personal data	Yes. Monitors availability	No.	No.
Test & Simulation	Yes. Test & simulate the system under cyber-attacks simulator to verify the security of the system	No.	Yes. Test & simulate the system in expected situations to verify the reliability of the system.	Yes. Test & simulate the system in unexpected situations to verify the resilience of the system	No.

Figure 2.3. ENACT contribution to SIS trustworthiness.

preserving the privacy of a SIS. The same applies to the safety property, the contributions of the enablers on security, privacy, reliability and resilience properties are important to help ensuring the safety of a SIS.

The enablers from the monitoring and analytics layer of the ENACT DevOps Framework (*i.e.*, security and privacy monitoring, behavioural drift analysis, and root cause analysis) are considering security, privacy, reliability and resilience aspects. It is worth noting that these tools are complementary: On the one hand, the security and privacy monitoring enabler focuses on observing symptoms of security and privacy issues, and the behavioural drift analysis enabler focus on symptoms of reliability and resiliency issues. On the other hand, the root cause analysis focuses on understanding the causes of these symptoms.

2.4 Evaluation and Validation: the ENACT use Cases

The general applicability of the ENACT enablers was validated and demonstrated in the context of three use cases: Smart Building, Intelligent Transport System (ITS), and eHealth. Each of these use cases represent different application domains, all facing specific trustworthiness challenges as depicted in Figure 2.4.

2.4.1 The Smart Building use Case

The first use case explored and validated ENACT in the domain of Smart Buildings, *i.e.*, Smart IoT Systems that make use of Smart Building sensors, actuators and services. The use case leveraged the Kubik test facility,[2] which is a three floors smart building owned by Tecnalia and designed for testing and research. Kubik

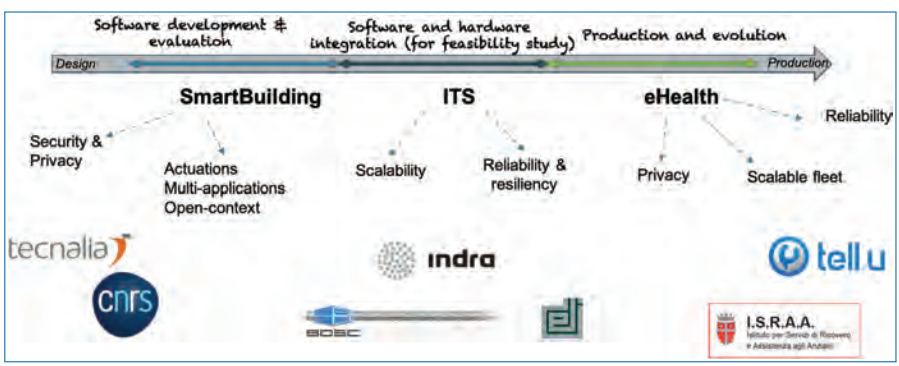


Figure 2.4. ENACT use cases and project partners in charge of the use cases.

2. https://www.tecnalia.com/images/stories/Catalogos/CAT_KUBIK_EN_dobles.pdf

offers SIS providers with a flexible framework not only to explore the opportunities offered by a rich ecosystem of sensors and actuators when designing novel IoT solutions but also to test and make experiments with the SIS resulting from this design in a real infrastructure. Thus, the smart building use case helped us validating our ENACT enablers in the early design stage of a SIS. During the project, several applications dedicated to aspects such as energy efficiency or user comfort were designed, developed and tested in Kubik. This context introduced specific DevOps and trustworthiness requirements that motivated the use of the some of the ENACT enablers.

For SIS providers it is especially important in this early design phase being able to quickly deploy and test the different applications and services that will compose or extend the existing SIS and thus run on IoT, Edge and Cloud infrastructure. The GeneSIS and TaS enablers aim at supporting DevOps teams in such activities.

Smart Building systems are typically composed of several applications controlling different actuation devices within the building (*e.g.*, HVAC, roller shutters, lights, TVs). In such a setting, it is of paramount importance to make sure the actuators are properly managed as to control their effects on the environment (*i.e.*, applications are behaving as expected). On the one hand, while it can be assumed that one application in isolation has a proper control over the actuators it applies, from the SIS perspective this assumption does not sustain as several applications may concurrently control shared actuators. Without proper mechanisms to handle such situation, the behavior of the actuator can quickly become unpredictable and possibly harmful. The ACM enabler aim at support the design of such actuation conflict handling mechanisms. On the other hand, indirectly, one actuator, possibly managed by an application, may hinder the effectiveness of another, managed by another application. Avoiding such loss of effectiveness is a complex task, which, without proper support, requires a deep analysis of the applications under operations. The BDA enabler propose to relieve developers from such a task, whilst the ACM enabler help mitigating the problem.

As in many other domains, smart buildings typically expose a broad attack surface and their security must not be an afterthought. The S&P Monitoring and control enabler provide a means to observe the security of the SIS and support security by design.

More details about the use case can be found in Chapter 11.

2.4.2 The Intelligent Transport System use Case

The second use case explored and validated ENACT in the domain of Intelligent Transport Systems, in particular exploring how SIS could be used for train integrity control. INDRA, as the system integrator, needs to continuously evaluate

the subsystems with both software and hardware from their suppliers (*i.e.*, EDI and BOSC in this project), and adjust the design and implementation of their main services accordingly in order to maximize the integration of the subsystems. DevOps guarantees the effectiveness of the integration process, and provides real-time feedback to both the integrator and the suppliers as reference for subsequent development activities. The ENACT enablers were thus exploited in the use case at a stage where the focus was on understanding how best the hardware and the software can integrate, and if the integrated solution fits requirements for the solution to be scaled in production.

In particular, key challenges included to understand (i) how the software performs on the gateway and handles failures as well as (ii) how the overall SIS scales as, in the long term, the number of gateways and sensors is aimed to grow up to thousands of nodes.

To understand how the software performs on the gateway the first step was to actually deploy it. The use case exploited GeneSIS for this. The later was integrated as part of the Indra delivery pipeline, making sure that, when a new version of the software is ready, it is only deployed if the train is in a state where such maintenance activity is authorized. The second step was to monitor the system under operation and to report and analyse any failure. Because there can be many reason from which a problem in the software may originate, the use case leveraged the Root Cause Analysis enabler to guide DevOps engineers through a faster understanding of the problem.

For the testing of the solution at scale, building a testbed consisting of real devices was not an option as each individual gateway is already expensive. Instead, the approach selected was to build a hybrid testing environment combining a few real devices with simulated sensors and gateways. In such context, to make the tests as relevant and realistic as possible, the simulated devices must be able to replay real data from real scenarios as well as to inject erroneous data providing a means to evaluate how the system performs when operating properly and when error occurs. The Test and Simulation enablers perfectly fits these requirements and was a natural choice for Indra to evaluate their solution. The enabler was used to record real data from the system, simulate large infrastructure composed of several hundreds of nodes, and test the system accordingly, sometimes also simulating errors and attacks.

More details about the use case can be found in Chapter 10.

2.4.3 The eHealth use Case

The third use case explored a solution for remote patient monitoring and assistance that leverage a Personal Health Gateways (PHG) installed in patients' homes.

The PHG is at the core of the service as it integrates and controls various types of sensors and medical devices (e.g., blood pressure meter, fall detection sensors, glucose meter, video surveillance, indoor and out-door location tracking, etc.), and ensure that the right data are provided to the various stakeholders and to the integrated systems. The services running on the gateway needs to be customized to patient and family needs and requirements. This eHealth solution was partially developed within ENACT and is now in production with a large set of PHGs in production.

For eHealth systems, security and privacy are of paramount importance and compliance to GDPR and ISO 27001 is mandatory. As a result, risks must be carefully analysed and the necessary security and privacy mechanisms must be implemented on the medical gateway and secure communications with the Tellu Cloud platform need to be ensured. For instance, no data can be stored or processed on the gateways without strong gateway authentication and without enforcing a strong binding between the patient and the gateway. Before ENACT, no solution in the market fitted Tellu needs, preventing the migration of services to their medical gateway and thus hindering the full exploitation of the gateways. The Context-Aware Access Control enabler is the first solution to address this challenge. In complement, the Risk Management enabler provided Tellu with a mean to continuously perform the required risk analysis but also facilitating its reporting.

This eHealth solution was partially developed within ENACT and is now in production with a large set of PHGs in production. Each gateway should be configured to best fit (i) patient and family needs and requirements, and (ii) its operation context, including the set of sensors and medical devices connected to it. When dealing with a large fleet of gateways, the service provider (Tellu) cannot afford to operate and configure each Personal Health Gateway manually as this could easily overwhelm their operation teams, resulting in a service that is not scalable. The DivEnact enabler aim at addressing this challenge.

More details about the use case can be found in Chapter 9.

2.5 Conclusion

This chapter provided an overview of the ENACT approach to help the reader better apprehend the next chapters of the book. The focus was in particular on (i) the different enablers offered by ENACT for supporting DevOps for SIS, (ii) the ENACT Framework showing how the enablers may be combined, as well as (iii) the validation of these results in the context of realistic use cases from different IoT domains. Details about the enablers and use cases, such as scientific basis, implementation, application, and effect, can be found in the following chapters.

References

[1] Matt Bishop. "Computer security: Art and science. 2003". In: *Westford, MA: Addison Wesley Professional* (2003), pp. 4–12.

[2] Nicolas Ferry *et al.* "Continuous Deployment of Trustworthy Smart IoT Systems". In: *The Journal of Object Technology* (2020).

[3] Nicolas Ferry *et al.* "ENACT: Development, operation, and quality assurance of trustworthy Smart IoT Systems". In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Springer. 2018, pp. 112–127.

[4] Franck Fleurey and Brice Morin. "ThingML: A generative approach to engineer heterogeneous and distributed systems". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE. 2017. pp. 185–188.

[5] Anne Gallon *et al.* "Making the Internet of Things More Reliable Thanks to Dynamic Access Control". In: *Security and Privacy in the Internet of Things: Challenges and Solutions* 27 (2020), p. 61.

[6] Thibaut Gonnin *et al.* "Actuation Conflict Management Enabler for DevOps in IoT". In *10th International Conference on the Internet of Things Companion*. 2020, pp. 1–4.

[7] Edward R Griffor *et al.* "Framework for cyber-physical systems: Volume 2, working group reports". In: (2017).

[8] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010. ISBN: 860-1401501176.

[9] J ISO. "ISO/IEC 27000: 2012, information technology-security techniques-information security management systems-overview and vocabulary". In: *International Organization for Standardization* (2012).

[10] J ISO. "ISO/IEC, 29100.2011 Information technology-security techniques-privacy framework". In: *International Organization for Standardization* (2011).

[11] Victor Muntés-Mulero *et al.* Model-driven Evidence-based Privacy Risk Control in Trustworthy Smart IoT Systems". In: (2019).

[12] NESSI. *SOFTWARE CONTINUUM: Recommendations for ICT Work Programme 2018+*. NESSI report. 2016.

[13] Alexander Palm, Andreas Metzger, and Klaus Pohl. "Online reinforcement learning for self-adaptive information systems". In: *International Conference on Advanced Information Systems Engineering*. Springer, 2020, pp. 169–184.

[14] Gérald Rocher *et al.* "An IOHMM-Based Framework to Investigate Drift in Effectiveness of IoT-Based Systems". In: *Sensors* 21.2 (2021), p. 527.

[15] Hui Song *et al.* "Model-based fleet deployment of edge computing applications". In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2020, pp. 132–142.

[16] Antero Taivalsaari and Tommi Mikkonen. "A roadmap to the programmable world: software challenges in the IoT era". In: *IEEE Software* 34.1 (2017), pp. 72–80.

DOI: 10.1561/9781680838251.ch3

Chapter 3

Privacy Issues Control in Continuous Risk Management

By Victor Muntés-Mulero, Jacek Dominiak, Elena González-Vidal, Guillaume Mockly, Yuliya Miadzvetskaya and Tommaso Crepax

3.1 Introduction

In order to fully exploit the potential of IoT, it is crucial to facilitate the creation and operation of trustworthy Smart IoT Systems or, for short, trustworthy SIS. The different dimensions of trust for IoT systems were described by Yan et al. [22] concluding that risk management is an essential piece to guarantee trustworthiness. Markets in the need of trustworthy SIS, such as Smart Vehicles, Smart Grids or eHealth, are just flourishing and businesses will be continuously adapting to new technologies. In this context, poor risk management together with a reactive strategy usually forces companies to continuously re-factor application architectures to improve software quality and security, incurring high re-implementation costs [2]. Besides, there is a lack of solutions to support continuous control of risks. In general, organizations struggle to collect valuable evidence to control on actual effectiveness of the mitigation actions defined during risk management

process. In addition, many organizations use manual procedures based on using spreadsheets, by departments and locally [1]. This approach becomes quickly inefficient as projects or teams grow.

In the ENACT project, the Risk Management enabler is an evolution of the MUSA (H2020 Project No. 644429) Risk Management tool. While the tool created in the MUSA project focused in assessing risks and mitigation actions of Cloud Security based primarily on security related risks, ENACT enabler has evolved towards trustworthy SIS. While MUSA risk management tool explored risks related to the use of cloud services and recommended cloud services to be leverage from the system, ENACT enabler is able to consume the whole architecture of a SIS, expressed in GeneSIS, and find any type of vulnerabilities related to entities, processes, data store or data flows in the system. The basic risk management methodology the enabler is based on is described in [12] and, during the project, the risk management enabler has been effective embedded in the software development lifecycle both from a Dev and Ops perspective. Besides, although automated vulnerability detection is also discussed in [11], the details on how to create a knowledge base to support the detection are not discussed.

In parallel, GDPR discusses data protection by design and by default, remarking that it is essential to consider privacy from the beginning of any software development process to address related issues successfully. This is specially important for trustworthy SIS, since many IoT technologies are still under evolution and mixing legal requirements with a deep technical understanding is challenging. Therefore, including privacy aspects in a continuous risk management process is not straightforward.

Previous work related to this enabler, made in collaboration with the PDP4E project (H2020 Project No. 787034), is focused on showing how we embedded privacy-related risks explicitly through the combined use of models for both the architecture and the data flow implemented on the components of the architecture [12]. We achieve this by enabling the use of the information that is typically collected from the infrastructure to control security, thanks to the link that LINDDUN [21] establishes between privacy and security threats in STRIDE [16]. However, relevant aspects such as risk severity assessment is unclear, since most risk rating methodologies such as OWASP Risk Rating methodology are focused on security rather than privacy. As a consequence, impact assessment is usually focused on protecting the components of the system or the organization rather than data subject (DS) rights or other privacy-related principles.

Besides, even when it is possible to detect risks related to privacy from an engineering perspective (for instance based on LINDDUN) there is not a clear link between these and the main assets to be protected described in GDPR, including GDPR principles and DS rights. This makes it difficult for an organization

to declare how they stand in front of GDPR in terms of risk control. While this issue is not exclusively relevant for IoT systems, it is essential to guarantee trustworthiness, specially when IoT devices are involved and the complexity of the architecture of the system grows, together with the attack surface. Because of this, we found it essential to guarantee an explicit analysis of privacy challenges in this book.

In this chapter, we focus on the missing pieces mentioned above, namely:

1. Extension of the impact assessment methodology to enable a more privacy-friendly assessment.
2. Discussion about the mapping of engineer-related aspects with higher level concepts in GDPR such as principles and rights.
3. Creation of a knowledge base to enable automated vulnerability detection.
4. Evaluation of the enabler in an IoT-based use case scenario related to smart vehicles.

This chapter is organized as follows: Section 3.2 provides a brief description of some previous work used through the chapter to found its main contributions. Section 3.3 proposes an extension to the OWASP Risk Rating methodology. Then, Section 3.4 discusses the relationship between LINDDUN threats and GDPR-friendly vocabulary related to data processing principles and DS rights. Section 3.6 describes an IoT use case related to smart vehicles where trustworthiness is essential and we discuss the concepts presented in the previous sections. Finally, we draw some conclusions related to the contributions of the chapter.

3.2 Previous Work

3.2.1 LINDDUN Methodology

LINDDUN is a threat modelling methodology that encourages risk analysts to address privacy risks affecting end-users of the application or system. This methodology provides some guidance to identify and categorize threats under a set of general risks (Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, and Non-compliance). LINDDUN is sometimes considered the privacy-oriented alternative to the STRIDE framework [16]. In fact, LINDDUN threats are described in the so-called LINDDUN trees which are explicitly connected to STRIDE threats trees. LINDDUN methodology requires to formalize the functionality of the system and its dependencies with respect to personal data. In such sense, LINDDUN proposed the usage of the Data Flow Diagrams (DFD) [4]. The notation of a DFD is based upon 4 distinct element

types: (i) an external entity (i.e., end-users or third-party services that are external to the system), (ii) a data flow (explains data propagation and dependencies between all the functional components), (iii) a data store (i.e., a passive container of information) and (iv) a process (i.e., a computation unit).

3.2.2 Automated Vulnerability Detector

In order to facilitate an effective identification of privacy-related risks, it is important to make it easy for our enabler users to detect the vulnerabilities that expose the system to attacks that may violate DSs' rights. For that, we created an Automatic Vulnerability Detector (AVD) [11]. An AVD starts out from a set of DFDs to describe a software system under development. Based on these DFDs, it is able to detect potential vulnerabilities to kick off the risk analysis process. As explained, the AVD relies on a list of conditions that need to hold for a vulnerability to be effective. These conditions will need to be defined and stored in the Knowledge Base for the correct performance of the AVD.

3.2.3 Common Weaknesses Enumeration (CWE)

According to their website,[1] CWE™ is a community-developed list of software and hardware weakness types. It serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

3.2.4 Common Attack Pattern Enumeration and Classification (CAPEC)

According to their website,[2] CAPEC™ helps by providing a comprehensive dictionary of known patterns of attack employed by adversaries to exploit known weaknesses in cyber-enabled capabilities. It can be used by analysts, developers, testers, and educators to advance community understanding and enhance defences.

3.2.5 GDPR Enforcement Tracker

According to their website,[3] the CMS Law GDPR Enforcement Tracker is an overview of fines and penalties which data protection authorities within the EU

1. CWE – Common Weakness Enumeration (mitre.org): https://cwe.mitre.org/
2. CAPEC – Common Attack Pattern Enumeration and Classification (CAPEC) (mitre.org): https://capec.mitre.org/
3. GDPR Enforcement Tracker – list of GDPR fines: https://www.enforcementtracker.com/

have imposed under the EU General Data Protection Regulation (GDPR). Our aim is to keep this list as up-to-date as possible. The list does not list any fines imposed under national/non-European laws, under non-data protection laws (e.g. competition laws/electronic communication laws) and under "old" pre-GDPR laws.

3.3 Extending Risk Rating Methodology for Privacy

The ENACT Risk Management enabler uses LINDDUN as the baseline for privacy threat modelling. LINDDUN, just like any other modelling system based on STRIDE, has the issue that, once you automate the threat elicitation process, it returns as output an enormous amount of potential threats. Therefore, engineers need to identify in a given system what are, among a pool of many, the threats that actually need mitigation. At this point, we resort to risk assessment to prioritize the risks to mitigate.

Among the many risk assessment methodologies, the ENACT Risk Management enabler is based on the risk rating methodology of OWASP [20], a widely tested and accepted risk rating methodology for security. Unfortunately, the security nature of OWASP implies that the objectives it aims to achieve only partially intersect, but do not fully align, with those of privacy engineering. On the one hand, in 'traditional' security, the risk assessment is mainly carried out on behalf and benefit of the organization. Simply put, if the organization faces economic losses, the impact is deemed negative. Differently, in privacy and data protection, the assessment is made on behalf and interest of the DS, meaning that even if the organization can profit, the impact is negative if the DS suffers from a violation of its rights and freedoms. On the other hand, even though privacy engineering objectives of predictability, manageability and disassociability are in line with GDPR principles, we nonetheless acknowledge the existence of ontological differences between engineering objectives and privacy legal principles.

With all this in mind, the aim is to ensure that the use of OWASP does not undermine the protection of personal data. To do so, it is necessary to check up to which point OWASP's methods address legal requirements and, when needed, to customize them for privacy compliance.

3.3.1 Risk Appraisal and Risk Assessment

From a practical perspective, should a controller wish to process personal data, it is required by article 35, paragraph 1 GDPR to make two assessments. First, it has to assess whether the type of processing to be carried out is "likely to result in a high risk to the rights and freedoms of natural persons", which we call "Risk Appraisal".

Should the outcome of the Risk Appraisal be positive, then a second assessment is in order, this time on the "impact of the envisaged processing operation" – also known as Data Protection Impact Assessment, or DPIA. The Article 29 Working Party released official guidelines on how to conduct both [13]. It shall be noted that, both stages consider the overall risk value from the perspective of risk analysis (i.e. encompassing both what we term as 'likelihood' and as 'impact', regardless the different wording employed by the GDPR), albeit the former does so in a shallower and more abstract way.

3.3.2 A GDPR-friendly, OWASP-Based Privacy Risk Estimation System

DPIAs and Risk Appraisals are functionally dependent on privacy risk assessments. For this reason, we thought it would be twice as useful to propose an effective privacy risk rating system to underpin either of them. For the privacy risk rating system to be GDPR friendly, we look into what the GDPR requires in regards to DPIAs and Risk Appraisals and extrapolate concepts to use as factors.

The law is not clear in determining whether the concepts that are critical to the initial Risk Appraisal and the risk assessments are different. For example, recital 84 GDPR states that aspects to consider for risk evaluation are origin, nature, particularity and severity, but does not clarify whether such aspects only relate to risk assessment or also to Risk Appraisal. In addition, the WP29 is of the opinion that controllers have a constant obligation to implement measures to manage privacy risks:

> 'The mere fact that the conditions triggering the obligation to carry out DPIA have not been met does not, however, diminish controllers' general obligation to implement measures to appropriately manage risks for the rights and freedoms of DSs. In practice, this means that controllers must *continuously* assess the risks created by their processing activities in order to identify when a type of processing is "likely to result in a high risk to the rights and freedoms of natural persons".

The ambiguity of the law on one side, and a more functional approach towards risk assessment on the other, not only seem to allow for, but to encourage that risk management be continuously active in parallel to the data processing activity. In our case, this translates into the chance to use the same tool for Risk Appraisal, risk assessment and even to check whether there are residual risks after the DPIA is conducted. Consequently, since our study provides for a more granular analysis of privacy risks, it can discover issues at earlier stages of the process, and is partially automated, it can be used repetitively by the data controller to track and manage changing privacy risks over time.

The GDPR key in relation to DPIAs is article 35 paragraphs 1, 3 and 4, together with a number of recitals giving insights on what the law considers important to determine the severity of a risk, namely 71, 75, 76, 84, 89, 91, 92, and 116. By a combined reading of article 35 and the recitals, the WP29 extrapolated 9 processing operations as 'likely to result in a high risk' for the DS. If two or more of the following coexist, then the high risk is likely to occur and, thus, a DPIA is in order.

The processing operations are:

1. Personal evaluation or scoring of the DS, including profiling and predicting;
2. Automated decision-making that significantly affects the DS;
3. Systematic monitoring that results in observation, monitoring, or controlling of DSs;
4. Processing of sensitive or highly personal data;
5. Data processed on a large scale, considering number of DSs, volume and range of data, duration of activity and geographical extent;
6. Matching or combining data-sets;
7. Vulnerable DSs, when there is a power imbalance between the controller and the subject who is unable to consent or object to the processing;
8. New technology or innovative use of technology or organizational solutions;
9. Processing prevents a DS to exercise its rights, enter into contracts or make use of services.

Rather than systematizing privacy risk assessments, the GDPR gives a number of rules scattered among articles and recitals on how to understand what to consider while evaluating the severity of privacy risks. Similarly do the Guidelines of the WP29, which only better refine the categories of data processing operations considered 'high risk'. Therefore, one has to resort to the privacy engineering academic scholarship to find attempts to systematize privacy risk assessments that can help quantifying privacy risk factors.

Methodologically, the difficulty that any expert encounters when estimating risk values depends on that their factors, namely likelihood and impact, are impossible to quantify with precision. Only a few scholars have tried to lay the theoretical foundations for such assessment, and it is in fact from the studies of the building blocks of privacy risk metrics by Wagner and Boiten 2018 [19] that we start our exercise of combining the requirements of the GDPR, their interpretations by the WP29, and OWASP risk rating.

Our aim is to model a privacy risk rating system on the basis of the data processing operations considered 'high risk' by the WP29, with the further trust that such system will guarantee a high level of compliance with GDPR requirements.

In the next sections we put forward our solutions to address the issues of estimating risk likelihood and impact.

3.3.3 Likelihood

The calculation of likelihood is one that risk methodologies take at best as rough estimate, mostly because risks may or may not materialize due to a number of unforeseeable circumstances, as well as their probability of occurrence being stretched over an uncertain amount of time. Moreover, it is hard to determine complexity, variation and hiding of multiple root causes and consequences associated to each risk.

The imprecision of likelihood measurements does not put the privacy risk assessment to a halt. In fact, from a functional perspective, risk severity — labelled on a scale "from low to high" — provides enough data to inform risk management decisions in compliance with GDPR requirements. Nevertheless, a more accurate quantification of likelihood is important because the privacy controls that will be used for the mitigation of privacy threats will most likely decrease risks' likelihood, rather than impact [19].

The OWASP likelihood estimation methodology considers two sets of factors, the first being *threat agents* and their characteristics, and the second being *vulnerabilities*. Different threat agents, or attackers, are analysed on the basis of their potential *skills*, *motives*, *opportunities* and *size*. The idea behind such differentiation is that, for instance, attackers coming from the inside of an organization may have more opportunities in terms of access than outside attackers, yet be less skilled in terms of hacking abilities.

Privacy and security risks are different in nature, but the analysis for determining their likelihood seems, at first sight, similar. In fact, the determination of likelihood is only similar for those privacy risks that share analogous characteristics with security risks. Consequently, such privacy risks' likelihood is rated on the basis of how easily can a vulnerability be discovered and exploited by an identified threat agent, how many threat agents of the same type know about the vulnerability (i.e., awareness), and what intrusion detection measures are put in place against exploits by threat agents. Visibly, OWASP's determination of likelihood is fundamentally connected to threat agents, fact that depends on OWASP being designed on security attacks.

Regrettably, what OWASP does not consider is that threats may not be caused by a willing threat agent. In fact, there are privacy risks that lie outside the attacker-type scheme. As far as data protection is concerned, the controller organization itself can be considered as an attacker from which the DS shall be protected. Upon this assumption, many privacy-by-design and minimization concepts are rooted.

Bearing in mind the mentioned difference between 'traditional' security and privacy risk assessments, back to the comparison with OWASP, the threat agents in the privacy case are still the same individuals as in security, that is organization

employees, executives, etc.; however, for a given risk, they will have different motives, such as the exploitation of the DSs' personal data for economic advantage – more a matter of privacy than security [10].

Harms, both for security and privacy, can be caused by a poorly designed policy within an organization, the careless work of a DPO, or even the use of a badly designed tool for risk estimation. All these events increase the likelihood of materialization of adverse effects on the rights and freedoms of the DS, which the NIST defines "problematic data action", an "operation that a system is performing on personally identifiable information, that could cause an adverse effect or a problem for individuals [3].

Accordingly, the likelihood of problematic data actions cannot be quantified just over the characteristics of what may not be an attack. Therefore, the NIST suggests that, within a specific context, controllers take DSs' perceptions of which data actions they consider problematic through customer demographics, focus groups, surveys, etc. Once a list of problematic data actions is created, it should be possible to determine the likelihood of their happening. If, for instance, in one specific area, DSs have indicated "destruction of personal data due to earthquake" as problematic data action, the controller should be able to determine the likelihood of an earthquake happening. Similarly, if the DSs have identified "ambiguity of privacy policy wording", a controller should be able to register how many times did such unwanted events happen in its organization. Such problematic data actions can be monitored and quantified, and with them their likelihood.

A rating can be created to determine whether data actions that are perceived as problematic happen in the real world in a fashion that is rare to unlikely (1 to 3), possible to likely (4 to 6), or almost certain to certain (7 to 9). The mentioned levels mimic those of OWASP, where the likelihood of a security risk happening is rated as low (1 to 3), medium (4 to 6) or high (7 to 9).

3.3.4 Impact

In comparison to security, it is the use of OWASP to quantify the impact of privacy risks on DSs that presents the most substantial differences. Such differences, in turn, imply equivalent adjustments to the privacy risk rating system. Keeping the framework of OWASP as baseline, we combine it with the impact factors, categories and dimensions of Wagner and Boiten, mentioned before. To every impact category of Wagner and Boiten, namely, harm, scale, sensitivity and expectation, we map the key aspects of WP29's processing operations. To appreciate the varying impact of each of the four categories, we will do a simple exercise of analysing one category while keeping the other three constant.

OWASP divides the impacts of an attack into two categories, namely 'technical impact' on application, data, and functions, and 'business impact' on the organization. In regards to technical impacts, OWASP lists the loss of confidentiality, integrity, availability and accountability as factors. Evidently fundamental to security, such factors also have repercussions on privacy so long as the confidential, incorrupted, available and accountable information are personally identifiable. This means that, the four technical factors in OWASP for privacy are similar to, but have a much restricted material scope that excludes all data other than personal.

Harm. In regards to business impacts on the organization, it is crucial to understand that "only individuals — not agencies can *directly* experience a privacy problem" [3]. This means that each individual has a different perception of the harm caused by one problematic data action, and that such perception may also vary depending on the context.

The most important consequence of the personal nature of impacts is that it makes them very challenging to quantify consistently. NISTIR 8062 does not address the problem of quantification of harms directly, but suggests instead that businesses (or organizations) use costs, such as reputational or legal costs incurred for legal compliance, as proxies for the quantification of individuals' impacts. Wagner and Boyten suggest a different solution, that is either using a Likert scale (called 'perceived harm') or, as a proxy, the amount of damage that a court would be likely to grant (called 'damages awarded') [19].

Although non-optimal, the measure of harm from the standpoint of a DS is arguably to average scales similar to Likert's. An organization willing to understand the perceived harm to the DS involved in its systems should answer the following questions: "How much do you think that this problematic privacy action would harm the DSs related to your system? Not at all to moderately (0 to 3), considerably to significantly (4 to 6), highly to irreparably (7 to 9)".

The scales solve the problem of defining and finding a common metrics to harms of different nature, such as reputational harm, financial harm, etc. We suggest organizations to conduct a survey with their DSs to get a better understanding of how much the dreadful event would impact them. The averaging of the Likert scales comes to solve the problem that harm is felt differently among DSs, and it is thus impossible to tailor its measuring to each DS involved in a problematic data action.

It is safe to say that all high risk operations can be mapped to the category of harm. This is due to that, if no harm were to be inflicted to the DS, the related risk would not exist. We can take as examples the following categories: automated decision-making that significantly affects the DS, because it may bring to discrimination and exclusion, which are harms that are personally felt differently from one DS to another; systematic monitoring, because the

knowledge of being constantly monitored is also perceived differently by different DSs, and may affect their behaviour accordingly; vulnerable DS, because the power imbalance between the controller and the DS is greater when the latter is a child, an employee, a mentally ill person, an elderly, an asylum seeker, a patient, or another category of people who are unable to consent or oppose to processing due to relational or personal circumstances; processing prevents a DS to exercise its rights, because, e.g., the inability to enter into an insurance contract has different implications depending on the denied person, who not only may personally perceive the denial differently, but also be objectively awarded different damages by a court depending on the circumstances of the case.

Scale. To Wagner and Boiten, between two problematic data actions that affect (a) the same type of data (e.g., medical data), which belong to (b) equally harmed DSs (that is, DS who would feel the same personal harm as well as would be awarded the same amount of damages by a court) with (c) the same expectation of privacy, the one with the greater impact is that which affects the larger number of people.

Thanks to the processing operation 'data processed on a large scale', we are able to extend the scale category to a second dimension that is, volume of data. As regards to volume, the processing of more data items has a bigger impact than that of fewer data items: considering two data-sets, A and B, which contain exactly the same personal data belonging to the exact same people, the action of replicating multiple times and in different places data-set A would have a bigger impact on the DSs, because the chance for unlawful processing is likewise multiplied, or because more personal data are anyhow more demanding to protect.

Measuring scale is perhaps the easiest quantification among the impacts categories, because the number of DS involved and the volume of data are all objective, ordinal numbers that are either known, or so can be through data analytics. It is possible to use a specific category in OWASP called 'Privacy Violation' that combines the dimensions of volume and number of persons by measuring how much personally identifiable information could be disclosed by one particular processing activity. OWASP lists a number of options, and gives to each option an impact rating (in brackets), from 0 to 10: one individual (3), hundreds of people (5), thousands of people (7), millions of people (9).

Sensitivity. Keeping other impact categories constant, the more sensitive the processed personal data, the higher the impact on the DS. The law gives exceptional attention to data that, because of their nature, are considered special, namely: data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership; genetic data and biometric data used for the purpose of uniquely identifying a natural person; data concerning health; data concerning

a natural person's sex life or sexual orientation; and data related to criminal convictions or offenses.

Additionally, the WP29 lists a number of data types that should be considered sensitive because they increase the risk to rights and freedoms [13] (Sensitive data or data of a highly personal nature): "personal data linked to household and private activities (such as electronic communications whose confidentiality should be protected), or because they impact the exercise of a fundamental right (such as location data whose collection questions the freedom of movement) or because their violation clearly involves serious impacts in the DS's daily life (such as financial data that might be used for payment fraud)".

Processing operations involving all such data are considered 'high risk' but, unfortunately, there is no way to objectively determine which of these special data types have a bigger impact on the DS without considering the context and purposes of use. However, on the one hand, the law gives sensitive data a greater weight compared to non sensitive personal data and, on the other, it is safe to say that, between two processing activities of the same volume about the same person, that which includes the most categories of special data types must have a bigger impact. For these reasons, Wagner and Boiten suggest to use the number of different data types as means to measure sensitivity.

One measuring rating for sensitivity could be created by answering the question: how sensitive is the processed personal data? The options, with related impact rating in brackets (from 1 to 10), could be: not in the list [13] of sensitive data types (2); Not in the list, but could be easily used to predict sensitive data (5), Matches 1 category in the list (7), Matches 2 or more categories in the list (10).

Expectation. DSs have reasonable expectations about how their personal data will be handled by a controller. For instance, when consent is given as legal basis for processing, a DS should be able to predict what will happen to its data; similarly, a DS managing privacy settings to decide what types of cookies is a website allowed to use, or what information can it share with third parties, has an expectation on that only those cookies will be stored, and only those specific information be shared with predetermined third parties. Once the expectation is set, it is possible to determine to what extent has the actual processing deviated from it.

Processing operations involving evaluation or scoring of DSs are generally precursory to profiling, or to some forms of behavioural prediction. They are considered high risk because often leading to one or more of the other high risk processing operations, such as discriminating DSs on the basis of their personal vulnerability, race or other sensitive data, automated decision-making significantly affecting the DS, or preventing DSs to exercise rights or enter contracts. For this reasons, between

two collections of personal data, the one collection based on which a profile is created has a bigger impact on the DS.

Systematic monitoring of DSs with the purposes of observing, monitoring and controlling has a different impact on each DS. People tend to change their behaviour according to whether they know of being constantly monitored (so called 'chilling effect' [18]), and governments as well as private companies exploit more or less obtrusive technologies as means of control. When systematic monitoring is undetectable, personal data may be collected in circumstances where DSs may not be aware of who is collecting their data, how they will be used, and that personal data is being collected in the first place. When technologies for systematic monitoring are purposefully non obtrusive, the expectation of privacy of the DSs are very high and, thus, any type of personal data processing inherently diverges from such expectation.

Matching or combining data-sets of an unaware DS is an intrinsic violation of the principle of purpose limitation. Given a specific set of personal data, the DS should always be able to predict the consequences of a specific type of processing. The combination of multiple data-sets, thanks to data analytics, can reveal personal information that were not deemed to be shared within the principal processing, or even create new personal data [9]; both of the outcomes exceed the DS's expectation of privacy.

DSs have expectations on how a technology or a process will manage their personal data given the information they have on that technology at the time of collection. Therefore, innovative uses, or new technological or organizational solutions for data processing exceed such expectations unless the DS was put in the position to agree on the new means of processing. Given two processing operations on the same data of the same DS, the one using new technologies or solutions has a bigger impact.

To quantify the impact of exceeded expectation it is critical to first set a baseline and, to do so, we welcome Wagner and Boiten's suggestion to use Solove's taxonomy of privacy [17]. Based on the typically American concept of expectation of privacy, Solove's taxonomy is useful to determine what a DS expects from a data processing activity from the moment of collection, to dissemination, through management and storage. The divergence between the expected and actual means of processing, expected and actual types of created and shared data, and expected and actual consequences of processing can be measured by counting the number of exceeded categories of processing (collection, storage, dissemination, etc.) or, more granularly, by referring to the metrics we already used in other impact categories. This means that, for exceeded expectations on the types of processed data, one can refer to the higher sensitivity of the personal data, their bigger volume, the more severe personal or objective harm, and so on.

Another way to conduct such measuring is by considering that, as a general rule, the deviation from expectation gets bigger every time that the personal data, collected for a specific purpose, are reprocessed, re-used, re-analyzed, re-combined, etc. However, an engineer may not be able to count that, as the code may be implemented by several people. Therefore, we follow Solove's taxonomy and focus on expected intrusiveness into DS's life through the following question: "Considering that a potential system may collect, analyse, process and disseminate information, what is, in the eye of a DS, your system expected to do with the information?". Collect, analyse, process and disseminate information (2); Only disseminate information (4); Process, without disseminating information (6); Only collect information (9)". The idea is, the less the user expects the system to do with the information, the farther it will be from their expectation if a breach happens.

3.3.5 Summing up

To assess likelihood, a controller could determine in what fashion do data actions that are perceived as problematic by a DS happen in the real world. To assess impact, there are 4 categories to consider: harm, calculated on the controller's estimation of the damages to its specific categories of DSs; scale, calculated on the basis of how many individuals are involved in the processing activity; sensitivity, measured depending on whether the processed data belong to one or more of the categories identified by the WP29 as "high risk"; and expectations, calculated on the (un)expected intrusiveness of the data processing into a DS's private life, from the perspective of the data controller.

3.4 Connecting Engineer-driven Privacy Practices with GDPR

Despite the fact that the GDPR is a legal text and LINDDUN is an engineering method, an attempt can be made to align LINDDUN and the GDPR in order to bridge the existing gap between legal and technical practices and, thus, address the demands of privacy engineering community of, first, translating legal jargon of rights, values and principles into notions and tools that engineers are more familiar with, such as threat trees, data flow diagrams, etc.; and second, of operationalizing the GDPR, particularly in the the detection of the privacy threats and the elicitation of the associated mitigation strategies.

3.4.1 Initial Considerations

We start by remarking some initial considerations about the relationship between LINDDUN and GDPR.

Linkability (L), identifiability (I), detectability (D), and to some extent non-repudiation (Nr) are all pointing out to the existence of personal data, since the occurrence of one of these threats could lead to the identification of a natural person. According to the European legislation, the anonymous information does not require protection for compliance with the principles of data protection. Anonymous data do not relate to an identified or identifiable natural person and are therefore considered non-personal. However, "in this era of big data, full anonymity is hard, if not impossible, and even more advanced anonymity techniques cannot guarantee full anonymity when data are linkable" [8]. The threat of linkability may necessitate a further analysis since it cannot be established without context whether the linkability of two items of interest would allow the identification of a natural person and, thus, qualify as personal data.

Linkability might lead to identifiability (i.e. linking data to an identity). Once the DS is identified or is identifiable, the information qualifies as personal data and triggers the applicability of GDPR.

Information disclosure links to arguably all principles of GDPR art. 5. In fact, when personal data are disclosed to non-authorised parties they are no longer under the control of the DS nor of the responsibility of the controller/processor, which means that all the procedural and substantial safeguards provided by art. 5 and related rights are exposed to risk of violation. Personal data shall be processed in such a manner that ensures appropriate security, including protection against unauthorised or unlawful processing, alteration or accidental loss.

Unawareness is linked to principles related to information requirements, as well as to the procedural enjoyment of the DS rights. Not only shall the DS be given all the information about data processing activities, but more importantly she has to be made aware that any processing of her personal data is happening. Unawareness links to the principle of lawful processing, insofar as the DS cannot consent to processing she is unaware of; same applies to any other right she is entitled to enjoy by active personal request (e.g., right to information, access, rectification, erasure, etc.).

Non-compliance threat could be associated with data protection by design requirement, accountability obligation under Article 24 GDPR, such as adopting appropriate technical and organisational measures ensuring the GDPR compliance or adopting internal privacy policies. For the most part we can speak about general GDPR non-compliance resulting in a pyramid of sanctions: from warnings to sanctions as a last resort.

In this subsection, we provide the description of each LINDDUN threat category and its relation with the GDPR.

3.4.2 Linkability

Linkability means "being able to sufficiently distinguish whether 2 IOI (items of interest) are linked or not, even without knowing the actual identity of the subject of the linkable IOI". Pfitzmann and Hansen give the following definition: "unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, etc.) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not" [15]. For instance, unlinkablity of a message sender/recipient to a message sent or received or unlinkability of a relationship between a sender and a recipient [15]. Unlinkability is one of prerequisites of anonymity. Nevertheless, failing unlinkability will not necessarily eliminate anonymity, but will decrease its strength [15].

From a legal perspective, linkability means that the failure to hide the link between different actions, identities or pieces of information could potentially result in the unexpected personal data processing. For instance, the Article 29 WP provides for the following example: Titius has these fingerprints, this object has been touched by someone with these fingerprints and these fingerprints correspond to Titius, therefore this object has been touched by Titius [14]. Thus, linkability allowed to establish a link between one piece of information and the individual. The linking of different pieces of information can result in the misuse of the personal data by third parties. Such misuse can be caused by the failure to implement the necessary controls to ensure an appropriate level of protection of personal data (e.g., failed anonymization). If the controller is not aware of the personal data processing operation due to the failed anonymization, it will not be able to comply with the GDPR data processing principles and, thus, will fail to ensure the respect for DSs' rights. Thus, linkability may result in the violation of a number of the personal data processing principles and of DSs' rights listed in the GDPR.

Relationship with personal data processing principles. First, the principle of lawfulness will be violated since there will be no lawful grounds for processing, as provided in Article 6 of the GDPR. Second, the principle of transparency will not be complied with, because DS will not be informed about the processing activities over their data. The DS might not be even aware at all that such personal data have been collected, used, consulted or otherwise processed and what is the extent of this processing.

Third, the purpose limitation principle will be also jeopardized since the controller, unable to establish the existence of personal data, will not be able to ensure

that the data collection is limited to "specified, explicit and legitimate purposes" (Article 5(1)(b) GDPR). Moreover, in this case the controller will be collecting personal data without knowing itself how and when these data will be used, since in its system the data are not identified as personal.

Moreover, the data minimisation and storage limitation principles will be also violated since the unawareness about the personal data mere existence will prevent controllers from establishing whether the same purpose can be achieved with a narrower amount of personal data and for a shorter retention period.

The inability to establish that the personal data exist in the system or that a third party can establish links between different pieces of information and, consequently, guess the existence of such data, will prevent us from ensuring that the data are accurate and kept up to date. As a result, controllers will not be able to ensure accuracy at all stages of collecting and processing of personal data and take every reasonable step to ensure that inaccurate data are erased or rectified without delay. Thus, contrary to the principle of accuracy, controllers will not make sure that outdated data are eliminated, or that data are correctly interpreted.

The compliance with the principle of integrity and confidentiality will be also jeopardized since the processing of the data, deemed as non-personal, will not be as secure as required for the personal data processing, "including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures" (Art. 5(1)(f) GDPR) . This will result in a lack of appropriate controls to prevent unauthorised access to the personal data as well as implement systemic quality controls in order to ensure that an appropriate level of security is reached. Moreover, the personal data will not be validated (e.g. using hashes), which might lead to some negative consequences, such as inability to guarantee its integrity and, consequently, the accuracy of that data.

According to the principle of accountability, the controller shall be responsible for, and be able to demonstrate compliance with, principles relating to processing of personal data and listed in Article 5 of the GDPR. The non-respect for one of these principles will trigger the accountability obligation.

Relationship with DS rights. Since linkability in many cases is undetected because the personal data is not recognized as such and is not traceable in the system, the controller will not comply with information obligation, as substantiated in Articles 13-14. Thus, DSs will be deprived of the right to obtain information about the processing activities over their data, the identity and the contact details of the controller, the purposes of the processing, the categories of the data and their recipients, and how the data are being controlled, monitored or used further. The information obligation is the essential first step setting out the stage towards the exercise of other DSs' rights. Neither right of access, nor right to

rectification or erasure of personal data, nor restriction or objecting to their processing will be possible unless the DS knows the personal data is processed by the controller.

3.4.3 Identifiability

"Identifiability of a subject from an attacker's perspective means that the attacker can sufficiently identify the subject within a set of subjects." [15] Identity can be explained and defined as the opposite of anonymity and the opposite of unlinkability [15]. In a positive wording, identifiability enables both to be identifiable as well as to link IOIs. The less is known about the linking to a subject, the stronger is the anonymity. The anonymity decreases with a growing linking [15].

The definition of identifiability provided in the technical literature seems not to be totally in line with the legal understanding of an identifiable natural person. While both the legal and technical literature recognise pseudonimisation as one of the techniques decreasing the likelihood of identifiability, the GDPR takes a stricter stance on pseudonimised data. For instance, Recital 26 GDPR sets out that "pseudonimised personal data, which could be attributed to a natural person by the use of additional information should be considered to be information on an identifiable natural person". And, thus, such data will be treated as personal under the GDPR, since pseudonym means that it is possible to backtrack to the individual and discover individual's identity. At the same time, the technical literature admits the flawlessness and high linkability potential of pseudonimised data, but still seems to treat pseudonimity as a concept in a slight opposition to identifiability [8]. "Whereas anonymity and identifiability (or accountability) are the extremes with respect to linkability to subjects, pseudonymity is the entire field between and including these extremes" [21].

In addition the concept of identifiability is not that straightforward. For instance, the GDPR provides a non-exhaustive list of identifiers in Article 4, such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person. "The natural person is "identifiable" when, although the person has not been identified yet, it is possible to do it" [14]. But the likelihood of identifiability should be analysed on a case-by-case basis. For instance, a very common name will not necessarily allow to single out one particular person from the whole of a country's population [14], but can achieve the identification of a pupil in the classroom. In addition, the name, combined with some additional information can also allow the identification of someone as a result of this "unique combination" set. Even a very descriptive information can identify someone, i.e. someone wearing a red hat at the bus stop at a particular moment.

The identifiability is a dynamic process and, while it may not be possible to identify someone today with all the available means, it may happen at a later stage due to a technological progress. To determine whether an individual is identifiable, Recital 26 GDPR underlines, "account should be taken of all the means reasonably likely to be used, such as singling out, either by the controller or by another person to identify the natural person directly or indirect". The likelihood of identification must be assessed in light of "objective factors, such as the costs of and the amount of time required for identification, taking into consideration the available technology at the time of the processing and technological developments".

Since identifiability is closely related to linkability, it will affect the same GDPR principles and DSs' rights.

3.4.4 Non-repudiation

Non-repudiation is the opposite of plausible deniability. Plausible deniability from an attacker's perspective means that she cannot prove a user knows, has done or has said something [21]. While the goal of non-repudiation is to provide irrefutable evidence concerning the occurrence or non-occurrence of an event, it must be admitted that some participants may desire that there is no irrefutable evidence concerning a disputed event or action [21]. Wuyts provides for some concrete examples where non-repudiation is a privacy threat. For instance, e-commerce applications, where the vendor can later use the signed receipt by the buyer as evidence that the user received the item. For other applications similarly users may desire plausible deniability in order to ensure that there will be no record to demonstrate the communication event.

In an attempt to single out the most linkable GDPR principles with non-repudiation, we came to the conclusion that non-compliance with integrity and confidentiality requirements might lead to the loss of control over the personal data and increase the probability that unauthorized parties can access it. Logically, the controller will be held accountable for such incidents and for lack of appropriate confidentiality strategies. We consider that right to be forgotten and right to rectification are intrinsically linked with plausible deniability, since they allow for ex ante rectification of the personal data inaccuracies and the possibility to ask for erasure of those data, which are no longer necessary for the purposes for which it was collected or where such purpose ceases to exist, or where the DS withdraws consent on which the processing is based. Thus, right to be forgotten and right to rectification will prevent a priori the third parties from getting access to the information, which the DS considers as inaccurate or compromising. Nevertheless, as provided in Article 17 GDPR some exceptions might apply to the exercise of the right to erasure,

including the situations where there is a need to strike a right balance between public interests, freedom of expression and other competing rights and legitimate interests. In addition, Deng et al. notes with regard to plausible deniability that it ensures that "an instance of communication between computer systems leaves behind no unequivocal evidence of its having taken place" [5]. Thus, in relation to the right to be forgotten and right to rectification, one might ask whether the controller should store requests for personal data erasure or rectification. And would not such storage be detrimental to plausible deniability? Thus, the right balance should be again struck between accountability obligations and DSs' legitimate interests.

In addition, in order to guarantee plausible deniability the data controller may decide to make the data less accurate to "cover user's tracks". While the GDPR requires to keep the personal data up to date and ensure that inaccurate data are erased or rectified without delay, plausible deniability may require a different approach towards accuracy. On one hand, the accuracy of personal data should not be compromised, on the other hand, making personal data less discernible from the outside may be necessary for ensuring plausible deniability.

3.4.5 Detectability

"Undetectability of an item of interest (IOI) from an attacker's perspective means that the attacker cannot sufficiently distinguish whether it exists or not. If we consider messages as IOIs, this means that messages are not sufficiently discernible from, e.g., random noise" [15]. The difference between unlinkability and undetectability is the following: in unlinkability, the IOI itself is not protected, but only its relationship to the subject or other IOIs is protected. For undetectability, the IOIs are protected as such [21]. Undetectability consists in, for instance, hiding the user's activities or location [21]. Undetectability in the past was referred as unobservability. However, since unobservability comprises both anonymity and undetectability, LINDDUN method focuses on undetectability.

Detectability threat is strongly related to the context. It is impossible to establish without further details whether detectability of one particular activity can lead to identifiability of an individual. But if we assume that detectability results in an identifiability of a natural person, the scope of the GDPR will be triggered in a similar way to linkability and identifiability.

3.4.6 Disclosure of Information

Information Disclosure is the exposure of information to individuals who are not supposed to have access to it. Principles of integrity and confidentiality will be the most relevant to guarantee the security of the personal data processing. While Wuyts

considers confidentiality as a security property, she highlights also its importance for preserving privacy properties, such as anonymity and unlinkability [21].

Similarly to linkability, information disclosure will also trigger all personal data processing related principles, since the data could be further collected, stored by third parties without specific purpose and without informing the DS. Thus, data minimisation and storage limitation principles cannot be complied with either. In addition, the accuracy of the personal data can be also jeopardized.

3.4.7 Unawareness

Unawareness occurs when a user is unaware of the information she is supplying to the system, and the consequences of her acts of sharing. In the era of digitalisation users tend to provide excessive information resulting in a loss of control of their personal information. Thus, awareness aims at ensuring that users are aware of their personal data and that only the minimum necessary information should be collected [21].

Unawareness points out to the violation of fairness and transparency requirements, since the DS is not informed of all the risks related to the personal data processing and was not provided all the information required in relation to their personal data processing. Unawareness also leads to the fact that the DS provides more personal information than required, and thus, the principles of data minimisation and purpose limitation are violated [21].

3.4.8 Non-compliance

Non-compliance is related to legislation, policy and consent and implies that the DS should be informed by the controller about the system's privacy policy and allows the DS to specify consent [21]. Wuyts gives some examples of non-compliance, such as incorrect privacy policies provided to the user or when the policy rules are incorrectly managed by the system administrator [21].

Wuyts notes that policy specifies one or more rules with respect to data protection and these are general rules determined by the stakeholders of the system; consent specifies one or more data protection rules and is determined by the user and only relate to the data regarding this specific user [21]. From a legal perspective, while the processing of personal data can be based on DS's consent, lawfulness of the processing is not limited to consent as a lawful ground for data processing activities. The GDPR provides for 5 additional legal grounds where the processing of personal data is not based on consent (Art. 6 GDPR).

When it comes to policy, Wuyts mentions compliance with internal policies of the company. However, compliance with internal policies of the company will not

be enough if those policies are not correct, lack detail or are not user friendly with regard to privacy notices provided. Thus, non-compliance with policies should be related to broader issues covering also some external requirements and legal framework applying to controllers.

Non-compliance threat, as described in LINDDUN, seems to be too generic and lacks in precision. Its current wording suggests that all the data protection related legal frameworks will be triggered. However, eliminating this threat is easier said than done, since the legal compliance is like shooting at a moving target, as it continuously changes while you are working on it.

3.5 Privacy-Related Risk Knowledge Base

In this section we briefly describe the knowledge base created in ENACT and PDP4E projects. This knowledge base is the baseline for the recommendations provided in the risk management enabler, as well as the main database for the Automated Vulnerability Detector described in Section 3.2.

The proposed knowledge base is founded on the LINDDUN methodology to frame privacy-related threats, as well as on STRIDE to cover security issues. We use the threat trees proposed in LINDDUN as the starting point to populate our knowledge base. We connect the content of our knowledge base to public content in the Common Weaknesses Enumeration (CWE) list. Besides, we also leverage information from the Common Attack Pattern Enumeration and Classification (CAPEC) dictionary, through their link from CWE. In addition, our knowledge base contains information to facilitate the link of its content to known fines and penalties which data protection authorities within the EU have imposed under the GDPR.

3.5.1 Definition of Concepts to be Stored in the Knowledge Base

Figure 3.1 presents the class diagram published in [12] to model risk management concepts to allow to control privacy risks using DFDs. In particular, we consider each element of a DFD (Entity, Data Flow, Data Store and Process) a specific asset, according to LINDDUN methodology. This diagram is inspired by the UML diagram presented by Gupta *et al.* [8].

Based on this class diagram, our knowledge base will contain the following information:

- LINDDUN threat category: Our knowledge base will classify all vulnerabilities depending on one of the LINDDUN threat categories, namely,

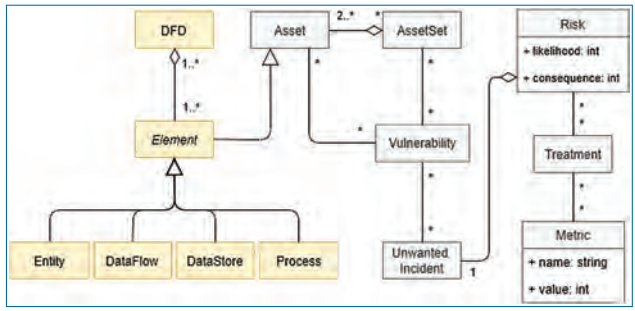


Figure 3.1. Class diagram to model risk management concepts using DFDs to describe system functionality [12].

Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness and Non-compliance. Note that LINDDUN is remarked because of the focus on this chapter on security issues. However, the knowledge base also uses STRIDE threats to classify privacy and security issues.

- Type of DFD component: we allow to express vulnerabilities of each of the four different types of components considered in a DFD, namely, entity, data flow, data store and process.
- Vulnerability information: We store information about vulnerabilities. This will include information such as a unique identifier, a short title, a longer description, etc. As mentioned before we will also need to store the conditions that are considered for a particular vulnerability to be relevant.
- Threat information: In order to simplify the information, we do not explicitly store the information about unwanted incidents, but we directly store the potential threats that may exploit a particular vulnerability.
- Control/Treatment information: we create a collection of controls that can be used as potential treatments to mitigate risks related to specific vulnerabilities and threats.

Based on this, in Figure 3.2, we describe the schema that represents the data schema of the information stored in this knowledge base.

The central concept of the schema is the *Vulnerability*. Vulnerabilities are connected to threats and threats are connected to controls. These link between these three types of elements constitutes the backbone of any risk management tool to provide options related to vulnerabilities and related threats and controls or mitigation actions. As explained in [11], vulnerabilities are also related to conditions. Besides, our knowledge base allows to establish dependencies among conditions. A certain condition may only make sense if another condition holds. For instance, let us assume a condition with id $c1$ that evaluates the following question: "Does

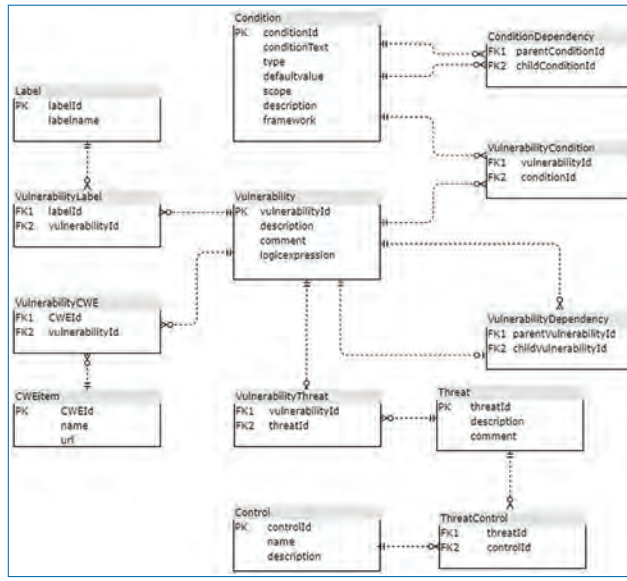


Figure 3.2. Knowledge base schema.

this entity represent a DS or a proxy to DSs?" From a privacy perspective, if $c1$ is true then other conditions may be relevant such as for instance "Does this entity need to be authenticated to execute this DFD?" or "Are credentials of this entity shared with any potentially untrustworthy receiver entity?". So, the relevance of conditions may depend on the evaluation of other conditions.

Besides, vulnerabilities are related to other vulnerabilities. In reality, inspired by STRIDE and LINDDUN, we have also created and extended threat trees in the knowledge base. Therefore, a vulnerability may be decomposed in more detailed vulnerabilities in a structure shaped like a tree. Relation *VulnerabilityDependency* represents the link between vulnerabilities in the tree. *Vulnerability* relation is also related to *Label* relation and *CWEitem* relation. The first link represents keywords related to the vulnerabilities that we will need later on to look for cases in the GDPR Enforcement Tracker.[4] This way, we will be able to show actual cases of GDPR enforcement in cases related to this type of vulnerability. For instance, "Information disclosure of a process" may be labelled with "Confidentiality" label or vulnerability named "Data is not encrypted" may be labelled as "Encryption". The actual mapping between cases and labels has been implemented in the PDP4E project. The second link corresponds to a manual exercise to connect the STRIDE vulnerabilities with CWE items. When the link is established, our enabler uses the url stored

4. www.enforcementtracker.com

in the database for each CWE item and scans online information to automatically detect mitigation actions and add them as controls in the *Control* relation. We also follow links to the CAPEC database related to attack patterns related with that vulnerability and store the information in the *Threat* relation and potential connected mitigation actions in the Control relation.

3.6 IoT Use Cases Description

Risk Management enables building trustworthy systems. Especially when considering SIS because of the way multiple independent devices connect together and exchange data. To illustrate this, the following section will describe the kind of challenges found in an IoT use case in terms of privacy risks for a DS. Privacy is specially challenging in IoT systems, not only because IoT systems rely on technology that is still not mature and in continuous evolution, but also because the higher integration of devices in the physical world makes those devices actual proxies to DS. For instance, the location of a device may easily act as a mechanism to deduce the location of a DS. For this purpose, following we describe a use case which is particularly relevant under this point of view.

3.6.1 Connected Vehicles

In this section we present an IoT-related use case focused on communications between vehicles and other vehicles or the road infrastructure, also called V2X. More specifically, the use case focused on the Cooperative Awareness Messages protocol [6], a part of the Cooperative Intelligent Transportation System (C-ITS) ecosystem which aims to make the different actors of road infrastructure share information on vehicle status, traffic, road works, etc... This protocol specifies how vehicles can share data about their position and status, like speed or heading, with other vehicles around them. This type of communication is of particular interest for the development of advanced driver-assistance systems (ADAS) as well as autonomous vehicle. It allows a vehicle to build a map of its surrounding and track nearby vehicles to anticipate possible incidents.

We chose to consider this system because it highlights a potential loss of control of data related to a DS, which can lead to significant privacy issues. The V2X network on which the CAM messages are transmitted is a local radio network. Because this system aims to inform other participants in the neighborhood about the status of the vehicle, messages are broadcast to every station capable of listening to the network. A consequence of broadcasting is that these messages cannot be encrypted with anything else than a key shared among all participants. As a result, the data

they carry can effectively be read by any station with access to the network without control of the sender.

Because the data received from these messages can be used to trigger warnings to drivers or even collision avoidance systems, it is necessary to ensure that they come from a reliable source. One of the mechanisms deployed to this end is the use of cryptographic signatures to authenticate the messages. When signing a message, a vehicle uses a pair of cryptographic keys provided by an external authority trusted by every participant of the network. This signature shows that they have been authorized to send messages and the receivers will be able to trust the data they send.

If used in a privately-owned vehicle, this system can raise issues about the privacy of the owner. First, to provide the information about the vehicle, a lot of parameters are collected by the equipment which generates those messages. Those parameters show when and where the vehicle has been driven, but also give information about the behaviour of the driver, like the speed of the vehicle. If these parameters were recorded and stored for analysis, they could be used to determine the driving habits of the owner. On top of that, because of the signature they carry, it can be possible to classify the messages by which vehicle has sent them. This could make mass recording of messages a source of personal data leakage if a link between a signature and specific vehicles can be made.

Relevant attack scenarios

The CAM system could be used in different ways which have consequences on the privacy of the owner of the vehicle. For example, it could be used to tail a specific vehicle, or the data collected inside the transmission equipment could be harvested by a malicious entity. For the purpose of this demonstration, we will use an attack scenario that leverages the signatures used to authenticate vehicles to track the behaviour of a specific vehicle. This scenario could be carried out by recording messages thanks to stations deployed across the road infrastructure. Because the vehicle is registered to a specific owner, such a record would allow to get the trip history of the owner and determine his usual destinations. The time at which the messages are emitted can also be used to build his schedule, the additional information on vehicle status like speed, light status, brake indicator can be used to analyse his driving profile. This data could be, for example, used by insurance companies to determine fees, or to show that an individual goes regularly to a political meeting.

3.6.2 Practical Implementation Aspects

All the contributions presented in this chapter have been implemented in the latest version of the ENACT Risk Management enabler in collaboration with the

PDP4E project. In particular, the enabler includes the extended version of the OWASP Risk Rating methodology presented in Section 3.3. Specifically, the questions suggested for impact analysis have been added to the enabler. Besides, the mapping between LINDDUN threat categories and GDPR data processing principles and DS rights have been embedded in the enabler. The enabler has a GDPR-based dashboard that allows to understand the overall level of risk for each principle and DS rights, based on the status of each risk detected in the enabler. Finally, the knowledge base described in 3.5 has been also embedded in the enabler, this enables not only the automated detection of vulnerabilities, but also benefiting from open databases such as CWE or CAPEC.

3.7 Risk Management Enabler Evaluation

This section aims to demonstrate the usage of the enabler on the connected vehicle use case. The enabler is evaluated based on its ability to help the user identify privacy risks in the use case and suggest meaningful treatments to manage them. The results are also compared with the way these issues are currently handled to demonstrate how adequate the enabler with respect to real world constraints.

Figure 3.3 presents the Data Flow Diagram used to support the attack scenario described in the previous section. It features two main processes: Key Provisioning and C-ITS App. The Key provisioning process is in charge of generating the key used by the vehicle to sign the messages it will send. The C-ITS App is collecting information on the vehicle position and status from the GPS and sensors to generate the messages, sign them and send them to the CAM network.

In the attack scenario, we consider the possibility of recording messages sent to the network. This corresponds to the data flow sending signed messages between the C-ITS process and the CAM Network entity.

The Risk Management enabler allows providing information on the role of these elements through the questionnaire described in [11] and used to fit the Automated Vulnerability Detector (AVD), presented in Section 3.2. This questionnaire contains the conditions used to discriminate the relevance of potential vulnerabilities and it is stored in the knowledge base, as described in Subsection 3.5. Table 3.1 gives some examples of questions that the user has to answer.

In particular, the question related to anonymous communication triggers several vulnerabilities based on the threat trees defined by LINDDUN. We now analyze a particular example based on the linkability of a data flow threat tree defined, presented in Figure 3.4. Those vulnerabilities are presented in Table 3.2. The vulnerabilities on linkability based on computer or session ID are both similar to the potential privacy issue of using the vehicle signature to track it. In this case, the

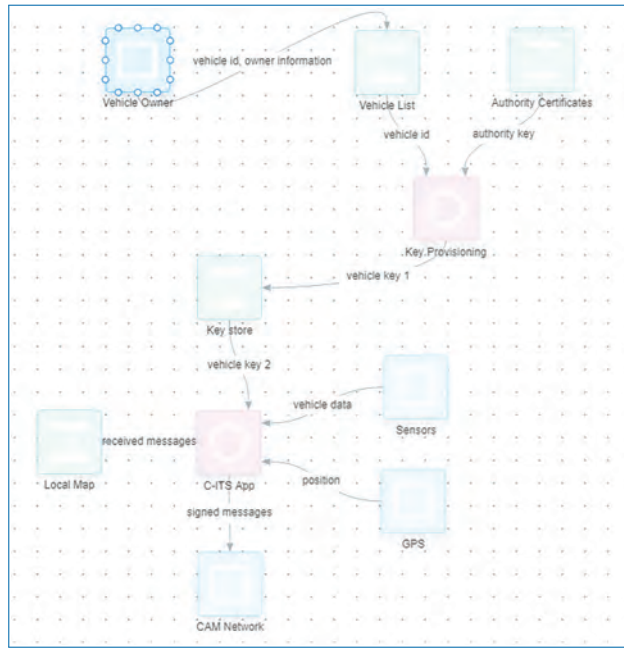


Figure 3.3. Data flow diagram of the C-ITS use case.

Table 3.1. Sample questions.

Type of Element	Question	Answer
Data Flow	Is the communication channel wireless?	Yes
	Are messages sent through this data flow encrypted?	No
	Is anonymous communication used?	No
Entity	Does this entity represent a DS or a proxy to DSs?	No
	Could this entity be or become untrustworthy (now or in the future)?	Yes

public key used for the signature serves as the ID linking together different messages, posing a privacy threat as an attacker may be able to continuously monitor a vehicle, and using other means to finally identify the driver, learn detailed information about her location, movements, habits, etc…

From a pure security point of view these privacy vulnerabilities would probably not be highlighted. On the contrary, they are probably desirable in some way as being able to trace the origin of a communication helps avoiding security issues related to repudiation or authentication, which is the reason why this signature has been added to the messages.

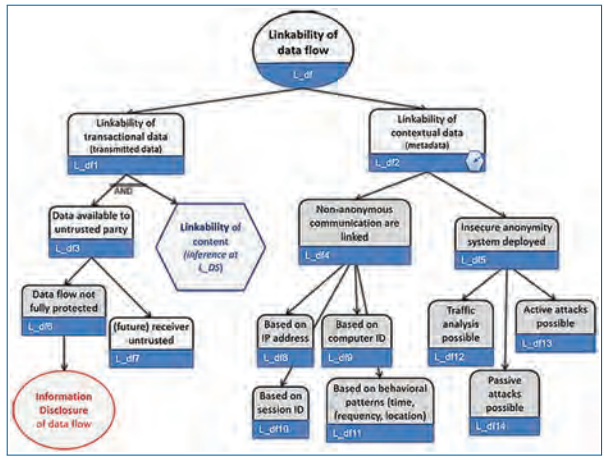


Figure 3.4. LINDDUN threat tree for linkability in data flows from https://www.linddun.org/linkability

Table 3.2. Privacy vulnerabilities detected.

Vulnerability	LINDDUN Property	Category	Associated Risk
Based on computer ID	Linkability	Non-anonymous Communication traced to entity	The data flow can be linked on computer ID and an attacker could link the ID to a person.
Based on session ID	Linkability	Non-anonymous Communication traced to entity	The data flow can be linked on session ID and an attacker could link the ID to a person.
Based on behavioural patterns	Linkability	Non-anonymous Communication traced to entity	Packet Counting Attacks, Timing attacks

If we consider more specifically the risk where data flows share a computer ID that can be linked to a person, the following controls are proposed by the Risk Management enabler:

- **Use tunneling through a Virtual Private Network (VPN).** This control aims at hiding the actual sender by using the exit point of the VPN as a public address. The messages can still be linked together but linking to the sender is harder.
- **Randomizing the computer ID.** This control uses unpredictable identifiers to break the link between messages. It may still be possible to link an identifier to a person but only the messages sent with this identifier will be impacted.
- **Use temporary identifiers which can be reused by different individuals across different periods of time.** This control hides the origin of messages

among the different participants. Both identifying the sender and linking the messages together is harder.

Out of these controls, the solution chosen to address this linkability issue in the described C-ITS use case is to randomly choose identifiers, called pseudonyms, in a pool assigned to a vehicle. This approach is also described in [7] and corresponds to the "Randomizing the computer ID" control described before as it makes it more difficult to link different messages by introducing randomness when choosing the pseudonym. This results in a more complex overall system as these identifiers will also need to be certified by a trusted authority but achieves a compromise between the security and privacy requirements. As an example, the ETSI Technical Report[7] also mentions the possibility of exchanging pseudonyms between vehicles. However, because multiple senders could be associated with the same pseudonym, this exchange would prevent law enforcement as access to the identity of a sender would not be available when required in an investigation. As the pseudonym scheme still allows for limited disclosure of identity, the impact on DSs is kept to a minimum while still addressing security requirements.

With this example, we demonstrated a full iteration of the risk management cycle, going from a guided identification of vulnerabilities to the selection of treatments. As shown the privacy risks identified and treatments selected are consistent with real privacy issues identified and discussed in the context of C-ITS systems. The knowledge base, being based on CWE and CAPEC, is definitely more suited for more classic IT systems than C-ITS systems in the way it describes issues and treatments so issues more specific to the domain considered are likely to be missed. For example being able to send forged messages could lead to traffic disruption. However, despite these shortcomings, it is still able to manage fairly specific issues.

The screenshot from Figure 3.5 shows how the enabler helps its user by offering direct access to potential risks and controls from the selection of vulnerabilities. This presentation allows to have a better view of potential consequences of a vulnerabilities, it also saves the engineer's time by grouping together the relevant information about a vulnerability and its associated risks.

3.7.1 Analysis of the Extended OWASP Risk Rating Methodology

In this section, we will evaluate how the modified OWASP risk appraisal method described before can help having a better view of the impact of a risk on privacy. This analysis will focus on the impact evaluation part because likelihood evaluation has not been modified.

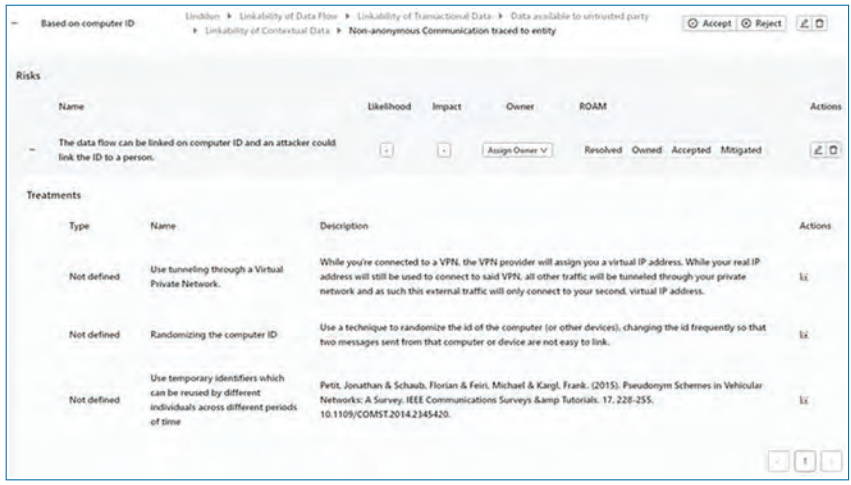


Figure 3.5. Screenshot from the risk management enabler showing the presentation of risks and controls.

Taking the example of the Identification based on machine ID risks that was considered before, with the classic OWASP methodology, the impact would be evaluated as presented in Table 3.3. This results in a technical impact of 3 and a business impact of 4.75. We choose the maximum of both categories as the final impact which gives a 4.75 impact score. This corresponds to a Medium rating according to the scales defined in the OWASP methodology.

Following, Table 3.4 presents the evaluation of the new categories introduced to measure impact on the DS. By reorganizing the previous score to take into account that the *Privacy violation* factor is now in the Privacy impact category and it is renamed as *Scale* according to Section 3.3, we get a technical impact of 3, business impact of 3.33, and privacy impact of 7. The new impact score is 7 which is a High rating.

As expected, the privacy-oriented factors indicate a significant potential impact for the privacy of a person using this system. Considering each impact category separately and using the score of the most important one as the final impact avoids minimising the contribution of a category if the others are rated low. This is an important aspect as security and privacy can be at odds with each other and the apparition of compensation mechanisms where one low-rated aspect could reduce the overall impact and undermine the proper assessment of risks.

3.7.2 Connecting the use Case with GDPR

Besides, the Risk Management enabler will establish a link with respect to GDPR principles and DS rights. This is an important benefit as, to our knowledge, it is

Table 3.3. OWASP evaluation of the identification by machine ID risk.

Factor	Value	Justification
Technical Impact		
Loss of confidentiality	1	The data sent through this data flow is not encrypted therefore it is not confidential
Loss of integrity	1	The integrity of the data is not affected by this risk
Loss of availability	1	The availability of either this communication or the CAM service is affected by this risk
Loss of accountability	9	The threat agent can listen passively to the network
Business Impact		
Financial damage	1	No financial damage will come directly to the maker of the system by exploiting the risk.
Reputation damage	5	Exploiting the risk can lead customers to be suspicious of using the system
Non-compliance	4	The system does not take into account privacy concerns like anonymity so it does not comply with GDPR, but it complies with technical specifications which also do not take into account privacy.
Privacy violation	9	The number of people affected can be measured in the millions

Table 3.4. Extended OWASP impact evaluation of the identification by machine ID risk.

Factor	Value	Justification
Harm	7	The information collected can be used to track the movements of a person. It can also be used to profile its driving style which could be used by insurance companies.
Sensitivity	5	Knowing where a person went can help deduce a political or religious affiliation
Expectation	7	The expectation on the CAM system is that messages are only processed locally by the surrounding vehicles and is not disseminated beyond the neighbourhood

the first tool that allows to map technical risks and controls with GDPR concepts, which are established from a legal perspective.

In the particular IoT use case presented above in this section, linkability issues have been detected related to the particular attack scenario under analysis. However, it is not straightforward to understand and explain, in front of a potential

Figure 3.6. Example of dashboard the risk status categorized by GDPR principles and DS rights.

explicit audit, what are the practical steps that we are considering from a technical perspective, to protect DS rights and GDPR principles, a part from a vague understanding that the mentioned threat may affect confidentiality.

Once the risks are detected and the mitigation actions selected, we can mark risks as mitigated if their severity was considered high enough and controls are deemed sufficient to accept the residual risk.

Thanks to the analysis performed in Section 3.4, we are able to connect linkability threats to different related GDPR principles such as lawfulness, transparency, purpose limitation, data minimisation, storage limitation, accuracy, integrity and confidentiality or accountability. Also, they can be connected to DS rights such as rights to be informed, of access, to data portability, to rectification, to be forgotten, to restriction of processing, to object and not to be subject to a decision based solely on automated processing. Omitting the responsibility for adequately managing these privacy-related risks, as the IoT system architect and developer, goes against GDPR and may involve harm to users and other DSs as well as penalties.

The Risk Management enabler will help users to control the impact from GDPR perspective, showing a GDPR-specific dashboard that may specify the level of mitigation of risks related to each GDPR principle and DS right. An example is depicted in Figure 3.6.

3.8 Conclusions and Future Work

In many cases, security and privacy are conflicting requirements. While security has been largely explored for decades, privacy is a much more immature area. This is

specially true when we try to control privacy in digital ecosystems based on newer and evolving technologies such as in the case of IoT systems.

In this chapter we have contributed to eliminate different roadblocks to achieve a better control of privacy through risk management. One of these is the capacity to improve risk assessment under the scope of privacy, essential for IoT systems, but also in general for any type of digital system. A second one is the capacity to connect the management of technical risks controlled by architects, developers and risks analysts based on privacy-related threat analysis methodologies like LINDDUN, with current legal frameworks to protect privacy such as GDPR. In particular, the level of connectivity between the GDPR concepts and LINDDUN threat categories is very large since they rely on different vocabulary and it is difficult to establish a 1:1 relationship between concepts. This interdisciplinary exercise is one of the first attempts to bridge the existing gap between the legal approach towards privacy risks and engineers approach towards privacy risks.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No. 780351 and No. 787034.

References

[1] Atif Ahmad, Justin Hadgkiss, and Anthonie B Ruighaver. "Incident response teams–challenges in supporting the organisational security function". *Computers & Security* 31.5 (2012), pp. 643–652.

[2] Barry Boehm and Richard Turner. *Balancing agility and discipline: A guide for the perplexed, portable documents*. Addison-Wesley Professional, 2003.

[3] Sean Brooks *et al. An introduction to privacy engineering and risk management in federal systems*. US Department of Commerce, National Institute of Standards and Technology, 2017.

[4] Tom DeMarco. "Structure analysis and system specification". In: *Pioneers and Their Contributions to Software Engineering*. Springer, 1979, pp. 255–288.

[5] Mina Deng Deng *et al.* "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements". In: 16 (2011), pp. 3–32.

[6] *ETSI EN 302 637-2 V1. 3.1-Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative*

Awareness Basic Service, 2014. URL: https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.01_30/en_30263702v010301v.pdf.

[7] *ETSI TR 103 415 V1.1.1-Intelligent Transport Systems (ITS); Security; Pre-standardization study on pseudonym change management*, 2018. URL: https://www.etsi.org/deliver/etsi_tr/103400_103499/103415/01.01.01_60/tr_103415v010101p. pdf.

[8] Smrati Gupta *et al.* "Risk-driven framework for decision support in cloud service selection". In: *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE. 2015, pp. 545–554.

[9] Mireille Hildebrandt. *Smart technologies and the end (s) of law: novel entanglements of law and technology*. Edward Elgar Publishing, 2015.

[10] Susan Eva Landau. *Listening in: Cybersecurity in an insecure age*. Yale University Press, 2017.

[11] Victor Muntés-Mulero *et al.* "Enabling Continuous Privacy Risk Management in IoT Systems". In: *Security Risk Management for the Internet of Things: Technologies and Techniques for IoT Security, Privacy and Data Protection*. Edited by John Soldatos. Now Publishers, 2020.

[12] Victor Muntés-Mulero *et al.* "Model-driven Evidence-based Privacy Risk Control in Trustworthy Smart IoT Systems". In: (2019).

[13] Data Protection Working Party. *Guidelines on Data Protection Impact Assessment (DPIA) and determining whether processing is "likely to result in a high risk" for the purposes of Regulation 2016/679*, 2017.

[14] The Article 29 Working Party. "Opinion 4/2007 on the concept of personal data". 01248/07/EN WP 136.

[15] Andreas Pfitzmann and Marit Hansen. "A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management". In: (2010).

[16] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[17] Daniel J Solove. "A taxonomy of privacy". In: *U. Pa. L. Rev.* 154 (2005), p. 477.

[18] Elizabeth Stoycheff *et al.* "Privacy and the Panopticon: Online mass surveillance's deterrence and chilling effects". In: *New media & society* 21.3 (2019), pp. 602–619.

[19] Isabel Wagner and Eerke Boiten. "Privacy risk assessment: from art to science, by metrics". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 225–241.

[20] Jeff Williams. "Owasp risk rating methodology". In: *Library Catalog: owasp.org. url: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology (besucht am 05. 06. 2020)*, (2020).

[21] Kim Wuyts. “Privacy Threats in Software Architectures”. In: (2015).
[22] Zheng Yan, Peng Zhang, and Athanasios V. Vasilakos. “A survey on trust management for Internet of Things”. In: *Journal of network and computer applications*, 42, (2014), pp. 120–134.

DOI: 10.1561/9781680838251.ch4

Chapter 4

Model-based Continuous Deployment of SIS

By Nicolas Ferry, Hui Song, Rustem Dautov, Phu Nguyen and Franck Chauvel

4.1 Introduction

Smart IoT Systems (SIS) are characterized by the presence of software deployed along the entire IoT-Edge-Cloud continuum. Software defines the behaviour of the SIS, and such behavior keeps evolving during the entire system life cycle following the ever-changing system context. This evolution may be realized as self-adaptation (such as the use of online learning for dynamic adaptation, as elaborated in Chapter 6) or manual reconfiguration of the existing software, but, very often, it requires releasing new versions of software. On the one hand, this evolution characteristic of SIS is typically conflicting with the traditional IoT systems vision consisting of devices with immutable code, once deployed at the factories. There must be new approaches for supporting the evolution of SIS. DevOps, on the other hand, promotes the idea of continuously delivering new software updates. Indeed, the DevOps movement promotes an iterative and incremental approach enabling the continuous evolution of software systems. Embracing DevOps can support the continuous evolution of SIS and improve their trustworthiness (e.g., security). As an evolution of the DevOps movement, DevSecOps [30] promotes security as an aspect that must be carefully considered in all the development and operation phases for the continuous evolution of systems to be secure. However, how

to effectively deploy the software update to the computing continuum is a main obstacle to enable DevOps or DevSecOps for SIS as it requires the capability of continuous deployment of software at all levels.

Continuous and automatic software deployment is still an open question for SIS, especially at the Edge and IoT ends. The state-of-the-art Infrastructure as Code (IaC) solutions are established on a clear specification about which part of the software goes to which types of resources. This is based on the assumption that in the Cloud it is easy to obtain the exact computing resources as required. However, this assumption is not valid on the Edge and IoT levels. A typical SIS in production often contains hundreds or thousands of heterogeneous and distributed devices (also known as a *fleet of IoT/Edge devices*[1]), each of which has a unique context, while their connectivity and quality are not always guaranteed. The major challenges are as follows:

- How to automate the deployment of software on heterogeneous devices possibly with limited or no direct Internet access?
- How to manage variants of the software which fit different types or contexts of Edge or IoT devices in the fleet?
- How to ensure the trustworthiness of the deployed software whilst the quality of the underlying resources are not guaranteed?

In the ENACT project, we focus on the problem of automatic software development for SIS, and our research attempts to address these challenges resulted in two complementary prototype tools for the deployment of SIS at two different layers: (i) GeneSIS targets at the device layer, providing a unified way to deploy software on heterogeneous devices, including those without direct internet connection; (ii) DivEnact targets at the fleet layer, allowing developers to deploy software into the abstract fleet as a whole instead of focusing on concrete individual devices. The tool maintains the software variants and assigns them automatically to the devices according to their contexts. With trustworthiness (e.g., security) being a concern cross-cutting both layers, our tools provide solutions that contribute making the deployment and the SIS trustworthy. At the device layer, we support the specification and deployment of security and privacy mechanisms together with the SIS software in a DevSecOps fashion. Moreover, we provide the novel *rolling deployment* method to guarantee the availability of the deployed software as well as to handle errors during a deployment, i.e., in addition to the main software, we also deploy a

1. Similar to a fleet of vehicles in a transportation company, a fleet of devices are owned by the same application providers and distributed to different places or users. Devices in a fleet conduct relatively independent tasks, whilst coordinated by the application provider from a global perspective.

backup copy which will replace the main one when necessary without delay. At the fleet level, we maintain the software diversity within the fleet for security purposes.

Model-Driven Engineering (MDE) is the scientific basis underlying both GeneSIS and DivEnact. MDE is a branch of software engineering that aims at improving the productivity and cost-effectiveness of software development by shifting the paradigm from code-centric to model-centric. It has shown to be effective in supporting design activities [42]. This approach, which is commonly summarised as "model once, generate anywhere", is particularly relevant to tame the complexity of developing heterogeneous systems such as SIS. Models and modelling languages as the main artefacts of the development process enable developers to work at a high level of abstraction by focusing on deployment concerns rather than implementation details.

This chapter is organized as follows. Section 4.2 provides an overview of the current state of the art and of the practice for the automatic deployment of SIS. Section 4.3 introduces our solutions for the automatic deployment of SIS, first describing how they can be integrated in order to form a coherent deployment bundle and then detailing each our two enablers: GeneSIS and DivENACT. Section 4.4 focus on the support offered by our solutions to ensure the trustworthiness deployment of SIS. Finally, Section 4.5 draws some conclusions.

4.2 The State of the Art

Software deployment has been evolving from deployment of component-based commercial desktop software [39], deployment of component-based distributed applications [25], to deployment on Cloud resources, and more recently deployment for IoT systems along the entire IoT-Edge-Cloud continuum. Even though some core concepts from deployment of component-based applications such as *capability*, *port* in [25] can be inherited for deployment on Cloud or IoT resources, they need to be tailored and customized to fully address the specificities of these environments.

4.2.1 On the Deployment at the Device Layer

For some years now, multiple tools have been available on the market to support the deployment and configuration of software systems, *e.g.*, Puppet,[2] Chef.[3] These tools were first defined as configuration management tools aiming at automating

2. https://puppet.com/

3. https://www.chef.io/chef/

the installation and configuration of software systems on traditional IT infrastructure. Recently, they have been extended to offer specific support for deployment on Cloud resources. Meanwhile, new tools emerged and were designed for deployment of Cloud-based systems or even multi-Cloud systems (*i.e.*, systems deployed across multiple Clouds from different providers) such as CloudMF [19], OpenTOSCA [43], Cloudify,[4] and Brooklyn.[5] Those are tailored to provision and manage virtual machines or PaaS solutions. In addition, similar tools focus on the management and orchestration of containers, *e.g.*, Docker Compose,[6] Kubernetes.[7] As opposed to hypervisor virtual machines, containers leverage lightweight virtualization technology, which executes directly on the operating system of the host. As a result, the container engine shares and exploits a lot of resources offered by the operating system thus reducing containers' footprint. These characteristics make container technologies suitable not only for the Cloud, but also for Edge devices [13].

Besides, a few tools, such as Resin.io (Balena)[8] and ioFog,[9] are specifically designed for the IoT. In particular, Resin.io provides mechanisms for (i) the automated deployment of code on devices, (ii) the management of a fleet of devices, and (iii) the monitoring of the status of these devices. Resin.io supports the following continuous deployment process. Once the code of the software component is pushed to the Git server of the Resin.io Cloud, it is built in an environment that matches the targeted hosting device(s) (*e.g.*, ARM for a Raspberry Pi) and a Docker image is created before being deployed on the hosting device(s). However, Resin.io offers limited support for the deployment and management of software components on tiny devices that cannot host containers.

Regarding the deployment of elements of hardware and software that are to operate in harmony within a networked system, the Software Communications Architecture (SCA) [1] and IoT deployment share some basic concepts. The SCA is an open architecture that specifies a standardized infrastructure for a software-defined radio (SDR). However, the SDR SCA specification requires an SCA-compliant system for elements of hardware and software to operate within. In other words, the SCA is tightly tied to the specific needs for standardizing the development of SDRs, which is much less heterogeneous than the IoT domain in terms of

4. http://cloudify.co/
5. https://brooklyn.apache.org
6. https://docs.docker.com/compose/
7. https://kubernetes.io
8. https://www.balena.io/
9. https://iofog.org/

communication means, systems of systems, which may span all the layers of Cloud, Edge, IoT devices. Moreover, the SCA does not have any concept about supporting the deployment on devices not directly accessible.

In [34], we conducted a systematic literature review (SLR) to systematically study a set of 17 primary studies of orchestration and deployment specifically for the IoT. We found a sharp increase in the number of primary studies published in two-three recent years. We also found that most approaches do not really support the IoT deployment and orchestration at low-level IoT devices. As for the continuous deployment tools mentioned before, these approaches mainly focus on the deployment of software systems over edge and Cloud infrastructures whilst little support is offered for the IoT space. When this feature is available, it is often assumed that a specific bootstrap is installed and running on the IoT device. A bootstrap is a basic executable program on a device, or a run-time environment, which the system in charge of the deployment rely on (*e.g.*, Docker engine). Approaches such as Calvin run-time [28], WComp [27], or D-LITE [10], D-NR [24] all rely on their specific run-time environment where mechanisms such as dynamic component loading or class loading are typically used. There is a lack of addressing the trustworthy aspects and advanced support in the deployment and orchestration of the IoT.

To the best of our knowledge, none of the approaches and tools aforementioned have specifically been designed for supporting deployment over the whole IoT, Edge, and Cloud infrastructure. In particular, they do not provide support for deploying software components on IoT devices with no direct or limited access to internet. In addition, we also identified they do not offer support for including security concerns as core concepts in the tool and/or language.

4.2.2 On the Deployment at the Fleet Layer

While all these solutions discussed above are focusing on the deployment of a software system, they typically do not offer specific support for the management of a fleet of devices or a fleet of systems, which basically consists in managing large set of deployments with those solutions.

To the best of our knowledge, there is no effective solution to this *fleet deployment* problem. The start-of-the-art Infrastructure as Code (IaC) tools automate the deployment of one application on one device, or a predefined set of devices, but lack the support for distributing multiple variants across a large fleet. They also do not provide sufficient automated support for updating devices with constrained resources and limited (or none) Internet connectivity [34]. Such embedded and microcontroller-enabled devices traditionally have been flashed with 'one-off' firmware not intended to be updated in the future, but they are not often seen as active contributors to the common pool of shared computing resources, which

can be iteratively assigned and deployed with updated firmware. This has also led to the so-called concept of IoT-edge-cloud computing continuum, where computing and storage tasks are distributed across all three levels. On the other hand, the mainstream IoT/Edge fleet management platforms offer tools to maintain multiple deployments, the fleet of devices, and their contexts, but developers still need to manually designate which deployment goes to which device.

Another relevant reference architecture for deploying component-based applications into heterogeneous distributed target systems is described in [37]. In particular, the proposed architecture includes the concept of *Planner* – a component responsible for matching software requirements to available platform resources and deciding whether a component is compatible with a device. These existing specifications remain implementation-agnostic and only describe the high-level concepts. Software diversity is a new dimension of architecture-level properties, which is both a result of the hardware heterogeneity and a method toward more secure system. The fleet deployment approach provides a implementation-level support to our theoretical approaches towards a more diverse software [29, 45].

The assignment problem (such as assigning software components to the devices in an IoT fleet) frequently appears in ICT scenarios, where some resources need to be allocated to available nodes, often taking into consideration various context-specific characteristics [38, 40]. The research community has come up with multiple algorithms, ranging in their computational complexity, completeness, preciseness, etc. Many of these approaches treat assignment as a collection of constraints, which need to be satisfied in order to find an optimal solution in the given circumstances [2, 7]. The approaches based on Satisfiability Modulo Theories (SMT) are specifically popular and efficient due to their expressively and rich modelling language [8]. In this respect, a relevant approach that also makes use of SMT and Z3 Solver is described by Pradhan *et al.* [41]. The authors introduce orchestration middleware, which continuously evaluates available resources on Edge nodes and re-deploys software accordingly. Similar goal is pursued by Vogler *et al.* in [46], where authors report on a workload balancer for distributing software components at the Edge. Multiple approaches specifically focus on the autonomic and wireless nature of IoT devices and contribute to energy-efficient resource allocation, where the primary criterion for software deployment is energy efficiency [47]. A main obstacle for using SMT in practice is the gap between real platforms and the mathematical model.

Model-based techniques are often used to support DevOps. Combemale *et al.* [12] present an approach to use a continuum of models from design to runtime to accelerate the DevOps processes in the context of cyber-physical systems. Artavc *et al.* [3] uses deployment models on multiple Cloud environments, which is a promising way to support the smooth transition of software from testing to

production environments. Looking at approaches targeted at particular application domains, Bucchiarone *et al.* [9] use multi-level modelling to automate the deployment of gaming systems. In [16, 17], the authors apply model-driven design space exploration techniques to the automotive domain and demonstrate how different variants of embedded software are identified as more beneficial in different contexts, depending on the optimisation objective and subject to multiple constraints in place. To solve this optimisation problem, the authors also employ the SMT techniques and the Z3 solver implementation.

4.3 Overview of the ENACT Deployment Bundle

The ENACT approach to automatic software deployment is implemented as a prototype deployment bundle with two enablers, i.e., GeneSIS and DivEnact, supporting automatic deployment at the device and fleet layers, respectively.

Figure 4.1 illustrates how the ENACT deployment bundle is used in a typical SIS. The illustrative SIS has six subsystems, each of which is in charge of a particular business task, such as serving a user, monitoring a room, etc. Such a subsystem is usually composed by at least one edge device and several IoT devices such as sensors and actuators. For the sake of simplicity, we do not show all the IoT devices. These subsystems form the fleet of this SIS. Since each subsystem contains one main edge device as the main contact point, or gateway with the back-end service, we also refer to such fleet as an *edge fleet*. A fleet is normally distributed, with the edge devices (together with its IoT devices) serving different customers or tenants, and deployed in different locations. The developers often maintain one or several edge devices at their own premises for testing or trial purposes.

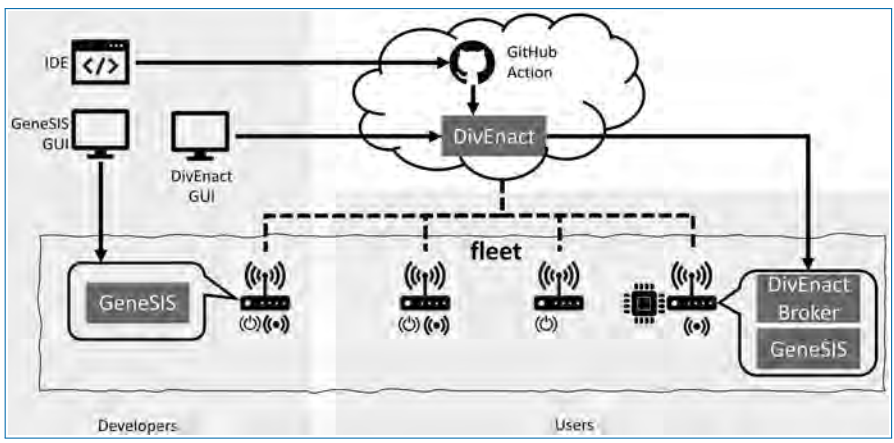


Figure 4.1. The ENACT deployment bundle.

GeneSIS supports the automatic deployment within a local subsystem, for example the deployment on the devices located on the developers' side. In such case, the developers can directly interact with the GeneSIS engine hosted on the local edge device, and use it as the bridge to further deploy required code to the associated IoT devices. In the development phase, developers define a *deployment model* in the GeneSIS modelling language specifying which software artefacts should be deployed onto which devices. Once the development phase completed, in the deployment phase, the same deployment model, or a slightly modified one, will be provided to the GeneSIS deployment engine, running either on a local machine or the edge device. The engine will install or update the software artefacts according to the deployment model.

DivEnact handles a different automatic deployment problem at the fleet level. When the developers want to release the new version of their application to production, they need to deploy software artefacts to all the devices on the users' sites. They cannot extend the deployment model to include every device in the fleet, because such a huge model is not maintainable, especially when the devices keep joining and exiting the fleet. Instead, since each user has a subsystem similar to the one at the developers' side, the developers can provide the deployment models they developed in the previous phase for the local subsystem to DivEnact. The latter maintains the list of all subsystems, and sends the deployment model to the devices before invoking the GeneSIS engine running on the edge device of the subsystem, to eventually deploy the software artefacts according to the deployment model. Within a fleet, the subsystems have different contexts, such as the device capacity, the connectivity, the user preferences, etc., and developers need multiple variants of their software to fit different contexts. DivEnact accepts multiple deployment models representing different software variants and configurations, coming for a series of releases, and automatically assign them to the proper subsystems. For the sake of availability, we recommend running the main service of DivEnact in the Cloud, with a light-weight DivEnact broker running on edge devices of each subsystem.

Next, we present GeneSIS and DivEnact, detailing their main innovations as well as how they contribute ensuring the trustworthiness of SIS.

4.3.1 GeneSIS

GeneSIS enables the continuous orchestration and deployment of Smart IoT Systems throughout the IoT-Edge-Cloud continuum. Given a description of a deployment topology, GeneSIS deploys and configures the needed software components, by connecting to the hardware (or software) nodes. This topology, the so-called deployment model, only prescribes what components must be deployed, how a single component can be deployed, and how they connect to each other. GeneSIS

automatically derives how to deploy them. Therefore, GeneSIS is composed of two key components: (i) a domain-specific modelling language for specifying deployment models, and (ii) an execution engine to enact the provisioning, deployment and adaptation of a SIS. We refer the reader to [20] for more details about the GeneSIS modelling language.

The target user groups of GeneSIS are mainly DevOps engineers, software developers, and software architects. The GeneSIS modelling language has been conceived so the deployment model can act as a touch point between development and operation activities. DevOps teams can use it to deploy either in development, staging or production environments. It is also worth noting a deployment model written using the GeneSIS modelling language is independent of the underlying technologies, i.e., GeneSIS can deploy components anywhere in the IoT-Edge-Cloud continuum: from microcontrollers without direct Internet access to virtual machines running in the Cloud.

The main task of the GeneSIS deployment engine is to reconcile two views of the system: the deployment model given by the user, and the current state of the running infrastructure, assuming that software components may already be running on the infrastructure, for example, as a result of a system upgrade. To reconcile these two views, the GeneSIS deployment engine adheres to the "models@runtime" architectural pattern [6]. It compares these two views and deduces what changes the adaptation engine must carry out on the running infrastructure to align it with the prescription, i.e., the deployment model given by the user. After the deployment, the engine synchronizes the current GeneSIS model with the actual deployment result. Such synchronization will ensure that all the tools in future DevOps cycles will leverage an up-to-date deployment model.

The GeneSIS deployment engine is non-invasive, meaning it does not require any GeneSIS bootstrap or agent running on a target device to deploy software on it. However, when decided by the DevOps engineer, the GeneSIS deployment engine can deploy on a target device a monitoring agent. This agent is an instance of netdata[10] and provides information about the performance and health status of a device, including data about software components it hosts.

Finally, the deployment engine can delegate parts of its activities to deployment agents running in the field. It is not always possible for the GeneSIS deployment engine to directly deploy software on all hosts. For instances, tiny devices do not always have direct access to the Internet or even the necessary facilities for remote access (in such case, the access to the Internet is typically granted via a gateway) or for specific reasons (*e.g.*, security) the deployment of software components can

10. https://github.com/netdata/netdata

only be performed via a local connection (*e.g.*, a physical connection via a serial port). In such case, the actual deployment of the software on the device has to be delegated to the gateway locally connected to the device. The GeneSIS deployment agent aims at addressing this issue. It is generated dynamically by GeneSIS based on the artefact to be deployed and its target host, and is implemented as a Node-RED application. We refer the reader to [20, 21] for more details.

GeneSIS comes with a set of predefined component types that can be seamlessly instantiated in deployment models. In addition, GeneSIS embeds a plugin mechanism that enables the dynamic loading of new component types. A components type repository is scanned by the GeneSIS execution engine before each deployment ensuring all available types are loaded before a deployment model is analyzed and deployed.

4.3.2 DivEnact

While GeneSIS focus on the deployment of a single system, DivEnact, the diversity-oriented fleet deployment enabler, is an implementation of our concept of *fleet deployment*. Fleet deployment is an automatic software deployment support for IoT/Edge applications, which allows developers to deploy software artefacts onto a fleet of devices as an abstract whole, without concerning about the concrete devices and their contexts in the fleet. The automatic fleet deployment tool, such as DivEnact, will maintain the devices and their contexts in the fleet, the software variants, and assign the variants to the appropriate devices depending on their contexts.

DivEnact utilizes Azure IoT Edge to maintain a list of edge devices, together with their contexts and run-time status. Developers provide DivEnact with a set of deployment models (typically GeneSIS models), each of which specifies a particular software artefact, together with the specification about how to configure and deploy it on an Edge device. In order to facilitate the definition of similar deployment models, we also introduce the concept of deployment *templates* and *variants*. A template defines the common parts among a number of deployment models, and a variant further instantiates the template as a deployment model. A common use case for this is to define a deployment model for a particular software, and then use variants to represent the different versions of this software. After receiving all the deployment models, DivEnact automatically assigns them to the list of edge devices, and enacts the deployment model on each edge devices to finalize the local deployment.

Figure 4.2 illustrates the technical architecture of the DivEnact tool. The DivEnact tool is designed and implemented following the established Model-View-Controller (MVC) design pattern for client-server application systems.

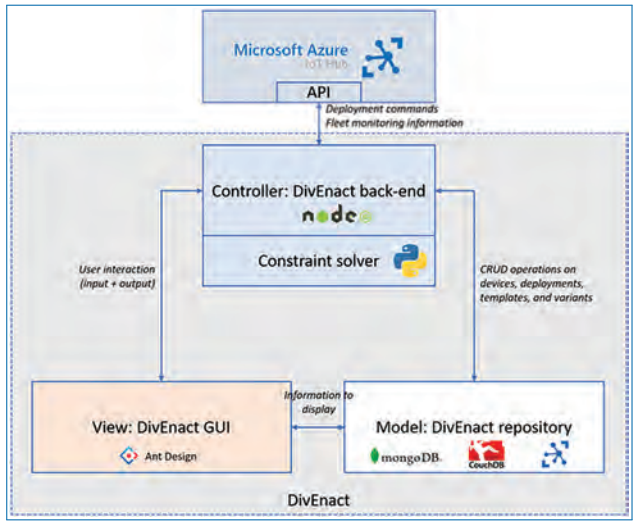


Figure 4.2. The architecture of DivEnact.

The DivEnact knowledge base stores various deployment- and fleet-related artefacts and is modified by the DivEnact back-end upon the user input received through the graphical user interface. The modification actions (CRUD – create, read, update, delete) are implemented on top of standard APIs and libraries. The model itself is spread across the following three repositories. MongoDB database is installed locally, along with a DivEnact instance, and serves to store information about templates and variants unique to each application system. There is a centralised repository in CouchDB for storing various ENACT artefacts, including deployment models used by DivEnact. In particular, the CouchDB database stores previously designed GeneSIS deployment models that are to be enacted on low-level IoT devices as part of the "last mile deployment". Azure IoT Hub Cloud portal keeps track of registered devices in the fleet and existing deployments. The information obtained from the hub reflects the current state of all the devices through continuously updated digital twins, as well as deployments applied to these devices (i.e., software modules currently deployed and running on each device).

The DivEnact graphical user interface remains the main point of interaction with the user. The main functionality is structured across several functions, i.e., the editing and maintenance of templates, variants, deployment models, devices and the assignment.

The back end of the DivEnact tool is implemented in Node.js. It receives RESTful requests originating from the user's graphical interface and manipulates the data model accordingly. It also interacts with the Azure IoT Hub API to update some information about the devices in the fleet and trigger deployments. The back

end also implements the actual diversification functionality (described in the next subsections) by receiving the input model from the user and passing it to the underlying Python script. Upon execution, the calculated solution is passed back to the user for the final approval.

The main function of the back end is the automatic assignment of deployment models into the list of devices, considering the constraints, the deployment preferences, the resource optimization, etc. We have implemented two experimental assignment approaches, using constraint solving and resource assignment theories as the back end mechanisms. The details of these two approaches can be found in our recent publications [15, 44].

4.4 Trustworthy Deployment

As explained before, ensuring the trustworthiness of the deployment and of the SIS is critical and challenging. It is a concern that crosscuts both the device or at fleet layers. In the following we detail how GeneSIS and DivEnact help addressing this challenge. Our effort is concentrated on three complementary directions, i.e., how to increase the availability of deployed system; how to automatically deploy the required security mechanisms together with the application; and how to maintain the software diversity across the whole fleet.

4.4.1 Deploying Availability Mechanisms

Availability refers to "the ability of the system to mask or repair faults such as the cumulative service outage period does not exceed a required value over a specified time interval" [4, p. 174]. Availability is a primary concern for business stakeholders because service interruptions often translate into money loss. The failure of an electricity meter for instance may affect the capacity of the electricity company to properly bill its customers.

Availability, as any extra-functional requirements, does not affect the system function, but rather affects its architecture. Building high-availability systems requires additional components to detect, repair, or even prevent faults, such as monitors, watchdogs, replicas, or voting mechanisms to name a few. Availability tactics are now well documented, so we refer the reader to [4, Chap. 5] for an introduction.

Many things can go wrong in Smart IoT Systems, including incorrect algorithms, network failure and delays, hardware failure, etc. In the following, we focus on scheduled outages, which are interruptions of service needed because of software upgrade, and internal faults, which are faults that occur because of defects

in the source code of the components. In other words, the availability support we present hereafter contributes (i) improving trustworthiness of a SIS by maximising its availability (by minimizing downtime during upgrades); and (ii) improving deployment trustworthiness by modifying a system and its deployment only if the deployment process is successful (i.e., old version of a software is removed only if the new version is up and running).

We extended GeneSIS with the ability to deploy mechanisms that cope with these two kinds of fault. To mask internal faults, GeneSIS deploys multiple instances of the same service/component (so called replicas) behind a proxy. When one replica fails, the proxy can query another replica. To mask scheduled outages, GeneSIS provides zero-downtime upgrades. We leverage the same architecture and deploys the new version (behind the proxy) before to decommission the older one. That way, there is always a replica available and upgrades do not affect availability.

Modern execution platforms such as Docker or AWS already implement various availability mechanisms, and, they have strategies for both scheduled outages and fault-tolerance. Docker Swarm for instance performs zero-downtime upgrades by deploying new services instances before to decommission the older ones. The challenge is that, from a deployment perspective, the availability tactics are tightly coupled to the underlying execution platform. Changing platform requires changing the deployment configuration.

To decouple availability from deployment platform, GeneSIS captures these deployment tactics independently of the underlying platform. If the platform already provides mechanisms (such as Docker Swarm), GeneSIS uses those, otherwise it deploys built-in components to implement the selected tactics. In the following, we illustrate three scenarios that show how GeneSIS copes with scheduled outages and internal faults.

1. Initial Deployment: GeneSIS deploys the system following the availability tactics selected by the user.
2. Internal Fault: A fault occurs in the system and we explain how the mechanisms that GeneSIS has deployed deal with that fault, so that it is not visible to the end-user.
3. Zero-downtime Upgrades: The user requests the deployment of a new version of the system and we illustrate how GeneSIS leverage the underlying mechanisms to minimize service disruption.

4.4.1.1 Using built-in components on top of docker

By default, GeneSIS does not make any assumption of the capability of the platform where it should install a component. It could be a very fully featured platform such as Docker Swarm (see Section 4.4.1.2) or simply an operating system offering

remote access (through SSH, Telnet, etc.). We detail here the later case, that is when the host is a bare OS. Recall that GeneSIS uses two strategies to improve availability: Replication to deal with internal faults, and zero-downtime deployment to deal with scheduled outages. To implement these two strategies, we need three capabilities that are provided by additional components:

- *Routing*, that is, the capability of redirecting incoming traffic to a selected replica. Network proxies provide this and in GeneSIS, we selected Nginx.
- *Error detection*, that is, the capability to proactively detect replicas that have failed (for whatever reasons). We used a watchdog, that is a component that periodically connects to the replicas and runs a so-called "health check". The health check is an application specific behaviour that confirms that the replicas is up and running. It could be requesting a predefined resource using HTTP, checking the status or OS-level services, or any other "quick-check". In GeneSIS, we have implemented simple watchdogs using Shell scripts and CRON tasks.
- *Spatial isolation*, that is, the capability to deploy multiple instances of the same application with guarantees that they can access external resources (network port, files on disks, etc.) without stepping on each other. GeneSIS uses containers (*i.e.* Docker in the current implementation) to ensure spatial isolation of replicas, but other container technologies such as LXC apply.

Scenario 1: Initial Deployment

The first step is for GeneSIS to ensure that the underlying host offers "spatial isolation" guarantees. To do so, GeneSIS first installs Docker as container offer such guarantees. Figure 4.3 below illustrates how GeneSIS interacts with the host to install docker and to create a "replicable image" of the software stack.

Given a component to deploy, GeneSIS first connects to the host through SSH and installs Docker (Step 1). Then GeneSIS configures Docker in remote mode so that other components (including itself) can access it through the network. Then, GeneSIS creates a new temporary container (by default, using the image "debian:10-slim") and installs the underlying software stack. To do this, GeneSIS traverses the underlying software stack and installs all underlying components by triggering the associated SSH commands into the container (Step 6, 7 and 8). Once the stack is installed and configured, GeneSIS converts it to a separate Docker image, that it later uses to install multiple replicas (Step 9). Finally, GeneSIS destroy the temporary container. At this stage, GeneSIS has enforced spatial isolation, and can then proceeds with replication and zero-downtime upgrades.

Once Docker is operational and the component to install is available as a Docker image, GeneSIS proceeds with the two remaining capabilities, namely

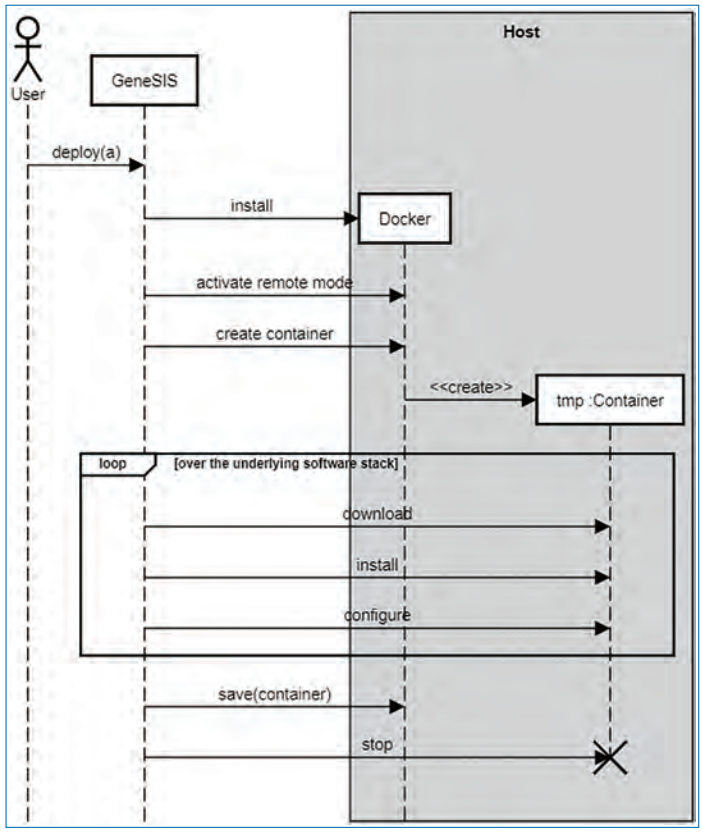


Figure 4.3. Automatically converting SSH resources into a Docker image. GeneSIS connects to the host, and execute all SSH commands into a new container, which it then saves as a new "ready to use" image.

detecting errors and routing as shown on Figure 4.4. First GeneSIS installs the proxy component through Docker (Step 1 and 2). Then, it installs the watchdog and configures it with the endpoints of the Docker host, the proxy and with the number of replicas to maintain (Step 2 and 3). For each missing replica, the watchdog requests Docker to provision a new instance of the image built in Scenario 1 and then the watchdog start checking the health of each replica periodically (Step 6 and 7). As soon as a replica is detected as healthy, the watchdog registers it to the proxy (Step 8), which uses it process user requests (Step 9 and 10).

Scenario 2: Fault tolerance

We now turn to the second scenario where one replica fails and we explain on Figure 4.5 how the watchdog detects and reacts to such a failure. The main mechanism to detect failure is the health check. Since a health check is an application-dependent behaviour, the user must provide it as a script to be executed periodically

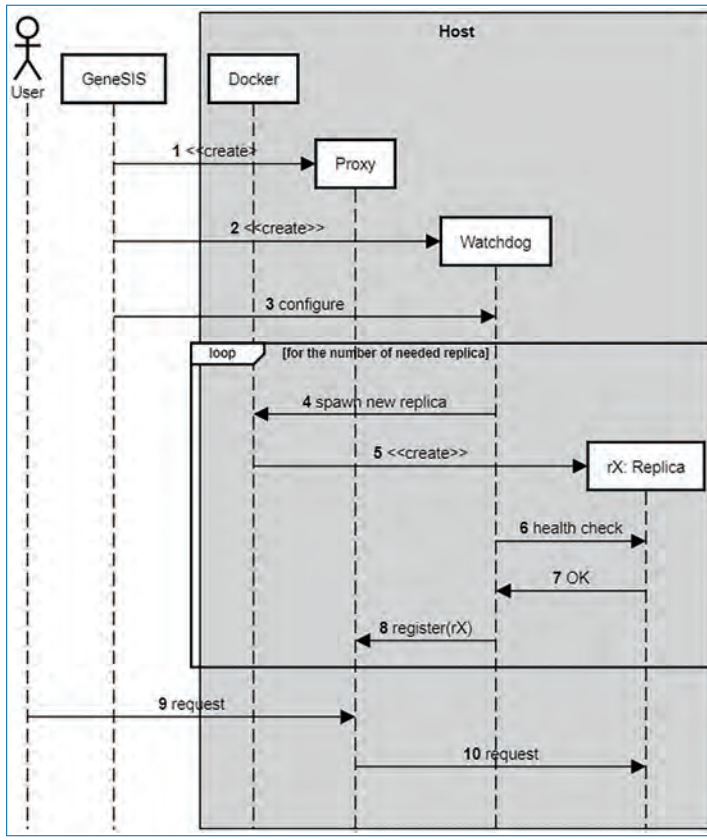


Figure 4.4. Configuring watchdogs and proxies to improve availability.

by the watchdog. GeneSIS defines the interface of the health check script as follows. The health check must accept the endpoint of the replica to query as its sole input parameter and must output the replica status in return through its exit code: Zero if the replica is healthy and any other value otherwise. This gives the user the capability to integrate any application-specific health check logic. The listing below shows one such health check script based on the HTTP status code, returned by a service.

```
#!/bin/bash
ENDPOINT="${1}"
response=$(curl --write-out '%{http_code}' --silent --output /dev/null"
    ${ENDPOINT}")
if [ "${response}" != 200 ]
then
    exit 1
fi
```

Listing 4.1. A Sample health-check script.

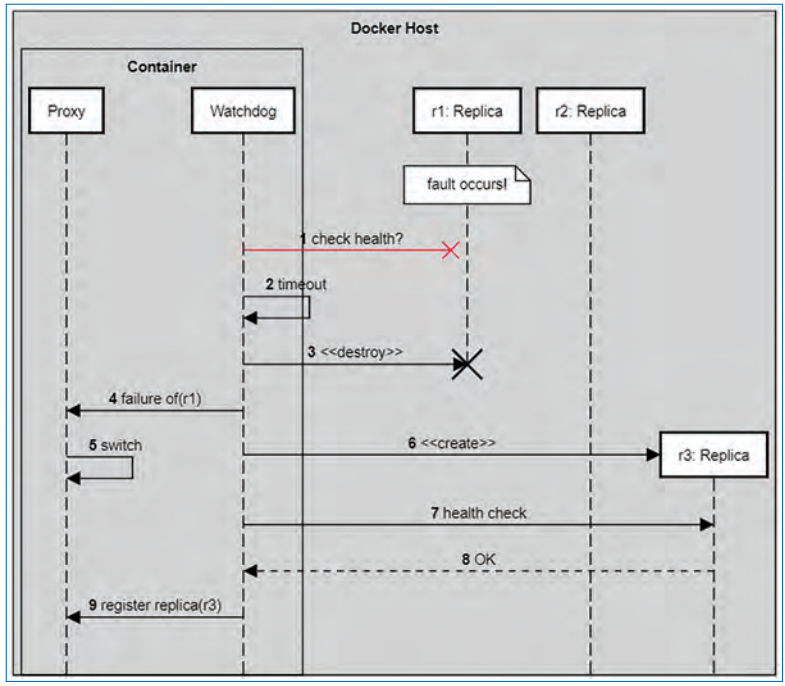


Figure 4.5. Masking internal faults to improve availability.

Note that this architecture can only detect replicas' failure as fast as the watchdog waits between two health checks. Besides, for the failure to be invisible to the user, there must at least two replicas, for the proxy to switch between them as soon as a delegation fails. Finally, transient phenomena such as network delays may be mistaken for replica failures and lead to unnecessary starts and stops of the container.

Scenario 3: Zero-downtime Upgrades

Finally, GeneSIS leverages these proxy and watchdog to guarantee zero-downtime upgrades, as shown on Figure 4.6.

When the user requests an upgrade, that is the deployment of a new version, GeneSIS first builds a new docker image of the software stack, including this new version. We described this process in Figure 15. Once this new image is ready, GeneSIS request the watchdog to perform the upgrade (Step 2). The watchdog thus provisions new instance of the new version (Steps 3 and 4) and, once these new replicas are operational, the watchdog registers them to the proxy (Steps 4, 5, 6 and 7). At that stage, incoming requests from the user are still delegated to the older version (Steps 8 and 9). Only once all replicas of the new version is operational, then the watchdog starts to decommission the older versions (Steps 10 and 11).

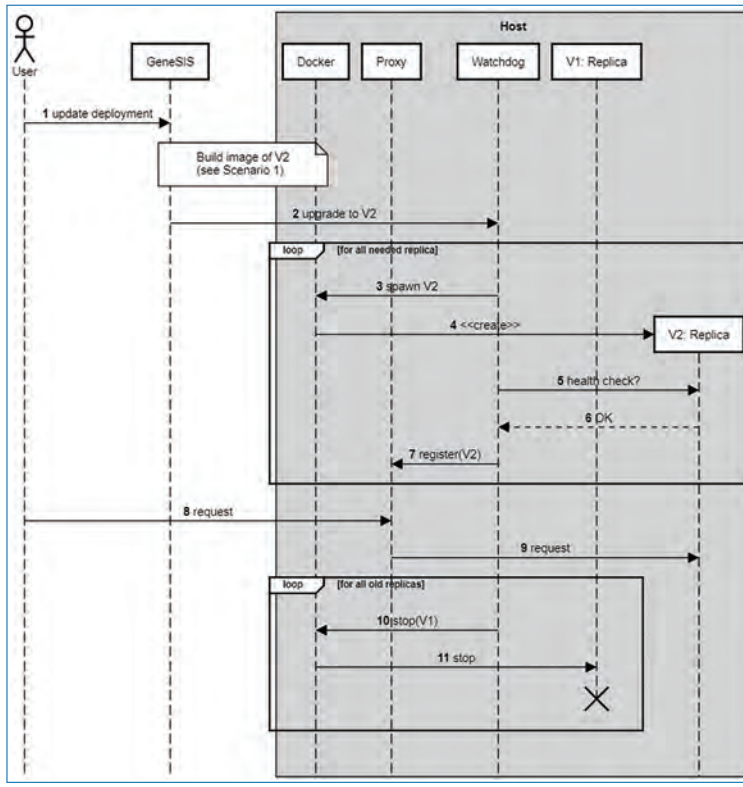


Figure 4.6. Using proxy and watchdog to guarantee zero-downtime upgrades.

4.4.1.2 Using docker swarm

In many cases, developers do not choose the platform on which their software runs: It may result from organizations' policies, customer requirements, etc. Platform such as Kubernetes, Docker Swarm or Rancher for instance all implement availability tactics, including replication and zero-downtime releases. Should the user use such platform, GeneSIS can exploit these native features to ensure fault-tolerance and zero-downtime upgrades. Docker swarm already implements routing among multiple replicas and fault detection. GeneSIS therefore delegates these features to Docker Swarm. We briefly example how GeneSIS handles our three scenarios using Docker Swarm.

Scenario 1: Initial deployment

This step is the simplest as we assume here that the host already runs Docker Swarm and that it therefore already guarantees spatial isolation. Here, GeneSIS simply requests Docker Swarm to deploy the given number of replicas of a given Docker image.

Scenario 2: Fault tolerance

This is also fully transparent from the GeneSIS standpoint. When GeneSIS delegates the deployment to Docker Swarm it specifies the number of replicas and the health check script to be used to detect faults. It is docker swarm that periodically checks the replicas status, provisions new ones if some have failed and route incoming requests accordingly.

Scenario 3: Zero-downtime redeployment

Further Docker Swarm also offers various strategies to upgrade a service (*i.e.*, the set of replicas in Docker Swarm parlance). Among many options, Docker Swarm lets the user specify the "update order". If this order is "stop-first", then Docker Swarm first stops all the replicas of the older version, and only then starts provisioning replica for the new version. By contrast, if the update-order is "start-first", then Docker Swarm provisions all new replicas before to decommissions, as GeneSIS would do without Docker Swarm (see Figure 18). This "start-first" option let Docker Swarm minimize service interruptions.

4.4.1.3 Limitations

The support for availability tactics in GeneSIS is limited to Docker platform, although the general principle applies regardless of the underlying technology and other can extend GeneSIS and support other technologies. In addition, there are other types of faults that the current tactics cannot deal with. Hardware failure for instance would take down the whole host and therefore all the replicas at once. To tackle hardware failure, replication would have encompassed hardware, but this goes beyond GeneSIS whose mission is to provide platform agnostic deployment. Nevertheless, using the ENACT framework, the Root Cause Analysis enabler can be used to monitor and identify such failures and DevOps engineers can use GeneSIS to migrate the software components on a new host, benefiting from its platform independence. Programming faults are also not dealt with. Because GeneSIS is oblivious to the inner working of the components it deploys, all replicas are similar and fail in a similar manner. For instance, if a defect in the code lead to a fault of one replica (say because of invalid user input), then all replicas will exhibit this fault. Only diversification techniques [5] could help having replicas whose behaviours differ from one another, and that exhibit different failure profiles.

Improving availability from a pure deployment perspective, as GeneSIS provides, is bound to stateless components that can be easily replicated. Replicating a component that persists state requires some modification of its code. Either we separate its state from its application logic (using a local database, for instance) and

we ensure that all replica can access this single data source. Alternatively, each replicas also have its own local copy but we must now define a strategy to ensure the correct and timely synchronization of the multiple copies of the state, and possible conflicts. Modern database engines offer such mechanisms and would need to be integrated with GeneSIS in an ad-hoc manner. Edge platforms however often only pass data further on to Cloud services, and are thus likely to be stateless, or can simply leverage a local database as a cache, a strategy that the GeneSIS availability mechanisms handles.

Finally, on an Edge platform there are resources that cannot be replicated and that would require further investigation. A serial link for instance cannot be shared between replicas and, in this case, dedicated, application-specific logic must be in place to ensure consistent behaviour between all the replicas.

4.4.2 GeneSIS for Continuous Deployment Supporting DevSecOps

GeneSIS empowers a DevOps team to cope with security and privacy concerns of SIS as it natively offers support for including, as part of the deployment models, concepts to express security and privacy requirements and for the automatic deployment of the associated security mechanisms [20]. More importantly, GeneSIS enables the continuous enhancement of security controls in a DevSecOps cycle to keep security mechanisms up-to-date and well-aligned with the evolution of SIS, as well as addressing IoT security risks that are always evolving.

In this sub-section, we present the latest development of GeneSIS for better supporting the continuous deployment and enhancement of security controls that can refine or override the associated (default) security and privacy mechanisms of the IoT platforms such as SMOOL [9] or FIWARE [11]. Such security mechanisms are further elaborated in Chapter 7. More specifically, GeneSIS provides a generic way for a DevOps team to extend such existing security mechanisms with other (third-party) security mechanisms to provide enhanced security controls in a DevSecOps fashion.

4.4.2.1 GeneSIS for the specification and deployment of security components

To better support DevSecOps, GeneSIS promotes specifying security mechanisms as explicit elements in the deployment model, instead of hidden (and thus tightly coupled) in the source code, so that developers can see and change the security mechanisms in the deployment model level. This includes specifying security requirements and capabilities, and supporting the deployment of security mechanisms as components reusable in different scenarios. Compared to the

previous GeneSIS version reported in [21], we have built a new library of off-the-shelf security components that can be selected for instantiating in the deployment model. More importantly, we provide DevOps teams with mechanisms to configure security components and inject fine-grained security policies into deployment components (without modifying their business logic), enabling their seamless integration with third party security mechanisms (services, libraries, etc.). These supports can ease the development, integration, and deployment of SIS with continuously enhanced security mechanisms (see Section 4.4.2.2).

GeneSIS supports the deployment of security components as any other software components in the way that their deployment and configuration can be defined via exposed APIs and configuration files. A security component to be deployed together with an IoT application can be declared in GeneSIS with "security capabilities" in a provided port. A required port of a software component that requires a matching security capability can be bound with the provided port of the security component that provides such security capability. Before enacting a deployment, the GeneSIS deployment engine validates the correctness of the provided deployment model. In particular, it ensures that the required "security capabilities" match the provided ones.

GeneSIS allows specifying the deployment of security mechanisms and policies built on top of IoT platforms. We present here its application to the SMOOL IoT platform, which is used in our ENACT project. Similar approach can be applied to other IoT platforms. At the development phase (as well as at the deployment phase presented below), GeneSIS provides the support to relieve developers from manually specifying and maintaining security monitoring and control mechanisms in the code of a SMOOL client. Instead, a developer can define its own SMOOL client, focusing on its business logic. We integrated the SMOOL client wizard with ThingML[11]. As a result, a single Eclipse IDE can be used to generate the code of a SMOOL client, which can then be directly used as part of a ThingML program. The proper Maven manifests are automatically created facilitating the building and release of the desired application. This means that the DevOps team can quickly develop the business logic of the SIS based on the SMOOL platform, including necessary security mechanisms. DevOps teams can define SMOOL clients that leverage built-in security properties to check and enforce security concepts on messages requiring security controls.

The SMOOL's default security enforcement can be done with the SMOOL clients built-in security metadata checker to verify messages exchanged

11. https://github.com/TelluIoT/ThingML

among them. In cases where a deeper control is needed, a specialised security metadata checker can be included in SMOOL clients, with additional privileges to watch and process the security metadata in messages exchanged, in the same way it is done with business logic concepts such as sensed temperature or gas values. This provides a fine-grained control on critical messages that may have a significant security impact in the IoT system such as orders to actuators. More precisely, a client code can conduct security checks based on policies to be fulfilled by ontology concepts by using any of these options: (i) the default security metadata checker (for minimal configuration), (ii) a custom security metadata checker implemented in the development phase (for full control of security), and (iii) a custom security metadata checker for integration with external security services. Whatever security options, GeneSIS provides support for easily configuring the security mechanisms and how they should be integrated and deployed with the SIS. Thanks to ThingML, GeneSIS provides advanced support for the three options.

To support the first option, GeneSIS enables the DevOps team to specify explicitly the default security policy that must be enforced by the Security checker. To support the second option, where the DevOps team can implement its own ad-hoc security checker, GeneSIS provides the means to automatically inject this security checker into the code of the component to be deployed and to rebuild the component automatically. More precisely, when deploying this security component, the GeneSIS deployment engine injects the security policy into the ThingML code of the SMOOL client. This code injection is done before GeneSIS triggers the compilation of this code to generate the actual implementation of the SMOOL client with the corresponding security policy.

To support the third option, GeneSIS not only injects the security checker code that integrates with a third party security solution (*e.g.*, Casbin[12] or the Context-aware Access Control mechanism [23], or a "gatekeeper" in [33]) but it can also deploy the latter. At the deployment phase, a SMOOL client can be deployed by GeneSIS as any other software components. Once the SMOOL client has been developed, the developer can specify how to deploy it together with the security and monitoring mechanisms that should apply to its SMOOL client. GeneSIS will then inject within the SMOOL client the necessary code to perform the security checks before actually deploying it. To do so, we created a generic security component that represents a SMOOL client as a deployable artefact. This client can follow any of the security check options discussed above and is implemented with ThingML code, which integrates (i) the necessary SMOOL libraries, (ii) the SMOOL client business logic, (iii) and the security logic. The main rationale behind this choice is

12. https://casbin.org/

the following. ThingML offers an extra abstraction layer that provides the ability to wrap the code and dependencies that compose a SMOOL client and to inject into it the necessary security code. In addition, it provides GeneSIS with a standard and platform-independent procedure to generate, compile, configure, and deploy the implementation of the security mechanisms. A similar approach could be applied to other IoT platforms. In this way, GeneSIS allows DevOps teams to reconfigure and update security mechanisms by design, in line with the evolution of IoT applications and the development of security and privacy risks. In the next section, we present more details on the DevSecOps support.

4.4.2.2 The DevSecOps support for the continuous enhancement of security mechanisms

SIS typically expose a broad attack surface and their security must not be an afterthought [22]. The ability to continuously evolve and adapt these systems to their dynamic environment is decisive to ensure and increase their trustworthiness, quality, and user experience. This includes security mechanisms, which must evolve along with the SIS, continuously fixing security defects and dealing with new security threats [32, 35]. Following the DevSecOps principles [30], there is an urgent need for supporting the continuous deployment of SIS, including security mechanisms, over IoT, Edge, and Cloud infrastructures [1]. The DevOps movement promotes an iterative and incremental approach enabling the continuous evolution of software systems. As an evolution of the DevOps movement, DevSecOps promotes security as an aspect that must be carefully considered in all the development and operation phases for the continuous evolution of systems to be secure.

In this section, we present how GeneSIS can enable the continuous enhancement of security controls in a DevSecOps cycle: from development to operation. GeneSIS also supports the adaptation of the system having enhanced security mechanisms or updated security policies with minimal impact on the already delivered and under operation. Our approach [18, 20, 21] for the continuous deployment of SIS with enhanced security mechanisms can serve the DevOps team in both adaptation and evolution of the SIS. First, GeneSIS supports for evolving SIS with updated security mechanisms according to a new development cycle. Second, GeneSIS supports for adapting security enforcement to improve how the IoT system operates securely. This DevSecOps support leverages the GeneSIS ' necessary mechanisms, interfaces, and abstractions to dynamically adapt the deployment and configuration of a SIS as presented earlier. We elaborate more on the two kinds of DevSecOps support in the following paragraphs.

First, GeneSIS supports for evolving SIS with updated security mechanisms according to a new development cycle. In this line of adaptation, the SIS in operation is evolving with new business logic components or even new physical devices

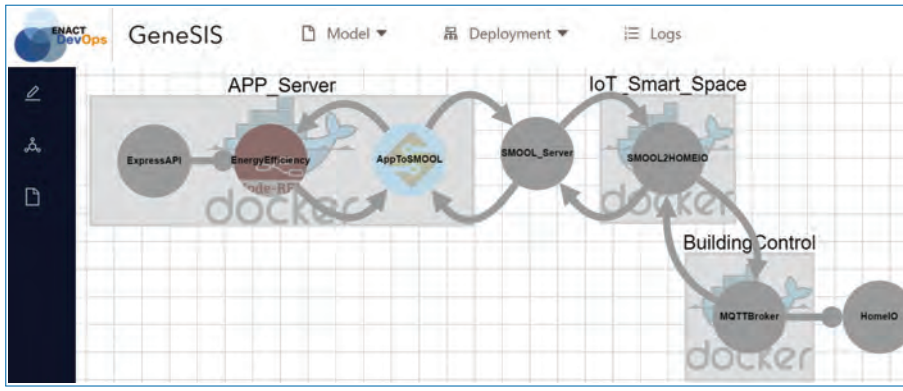


Figure 4.7. An initial version of a smart home system (deployment view).

being added resulting in the need for enhancing security mechanisms accordingly. We demonstrate this support using a smart home simulation called HomeIO.[13] More details on the deployment demo using the HomeIO simulation can be found in this video.[14] In the smart home system, there are IoT applications (*e.g.*, *UserComfortApp*) that get access to sensors' data (*e.g.*, temperature) from the smart home to make decisions and send commands to control the actuators, *e.g.*, window blinds. The applications interact with the smart home devices and services via the SMOOL platform (in the middle of Fig. 4.7). GeneSIS can easily support for the deployment of components that are either built on top of the existing IoT platforms like SMOOL or are independent of any IoT platform because of its generic approach for specifying deployment components. However, to make GeneSIS even more useful in practice, we have developed GeneSIS to ease the integration of IoT platform-specific components (*e.g.*, SMOOL clients) and IoT platform-independent components (*e.g.*, third-party security mechanisms like Casbin presented below) from development to operation.

In the initial version of the smart home system, there is the *EnergyEfficiency* application, which gets access to sensors' data to make decisions for energy efficiency and send commands to control the actuators, *e.g.*, window blinds. In particular, it maximizes the exploitation of daylights and regulates the in-door temperature whilst minimizing the energy consumption. If the room is bright because of daylight, it will switch off the LED-lights, and vice versa. On the other hand, if the room temperature is high, the application may need to close the window blinds to

13. https://realgames.co/home-io/

14. https://youtu.be/yQ9XYWu-EZM

prevent sunlight heating the room. The *EnergyEfficiency* application interacts with the smart home devices via the SMOOL platform. There are two notable security mechanisms associated in this first version of the smart home. The first one is a secure API gateway (Express Gateway[15]) that allows secure remote API access to the *EnergyEfficiency* application. The second one is a *SecurityEnforcer* by default of the SMOOL middleware that enforcing the security check for the data passing through, *e.g.*, only allowing genuine actuation commands to be sent to the actuators of the smart home. The latest version of GeneSIS has provided a built-in support to ease the specification of the Express API Gateway in the deployment model. Adding a new instance of Express API Gateway is easy. The remaining work for the DevOps team is to specify the configuration files of the API gateway, which define how the API of the *EnergyEfficiency* application can be securely accessed.

In IoT platforms like SMOOL, there are often default security enforcements. For example, the actuation orders must be checked before they are actually sent to the actuators. This check (embedded in the *SMOOL2HOMEIO* component, Fig. 4.7) makes sure only genuine actuation commands can be sent to the actuators. In other words, the SMOOL platform allows to check for actuation commands with valid security tokens. All the IoT apps must send actuation commands with valid security tokens.

However, during the evolution of the smart building system, new applications can be added, and new physical devices can also be added. In the subsequent development cycle, another application called *UserComfortApp* has been added to the smart home system. Moreover, the smart home system can also have new IoT devices such as *AirQualitySensor* or *SmartDisplay* as shown in Fig. 4.8.

New security requirements come up because the smart building system must control which apps can access which actuators. This means that more fine-grained security control must be introduced, which may not be available in the IoT platform. GeneSIS should support for seamlessly integrating new (third-party) security mechanisms into the IoT platforms. In this new development cycle, not only that the secure API gateway must be updated with a new configuration file, but also the DevOps team needs to introduce a new security mechanism that can enhance the fine-grained control of how different applications can access to the sensors and actuators of the smart home system. GeneSIS has a generic support for seamlessly integrating and deploying any advanced security mechanism together with the IoT platform in use, *e.g.*, the SMOOL platform. More specifically, in this example, the DevOps team develop an access control mechanism based on an open source

15. https://www.express-gateway.io/

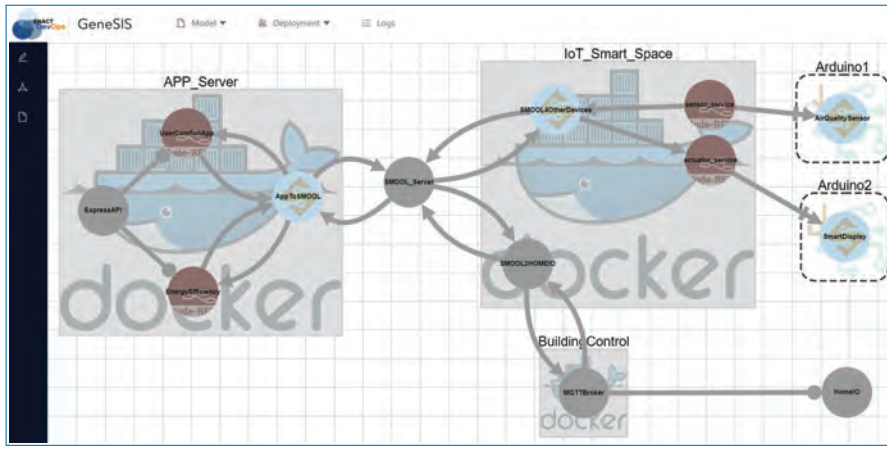


Figure 4.8. New applications and new IoT devices can be added in a development cycle.

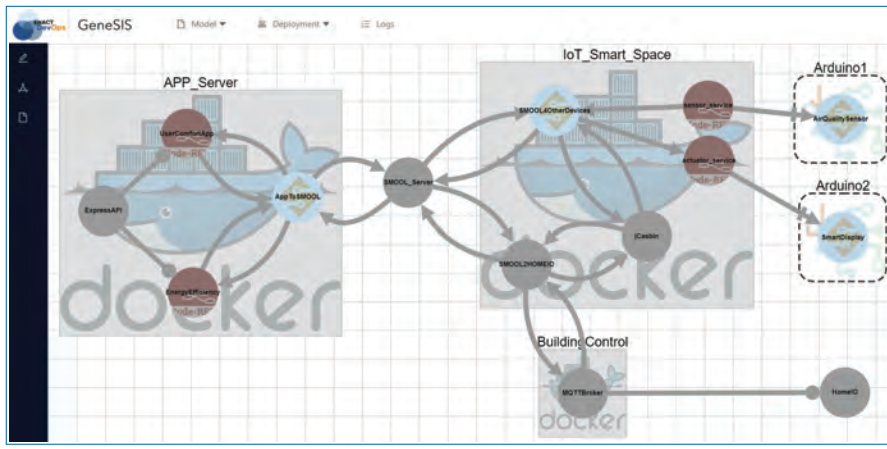


Figure 4.9. An enhanced security control has been added.

framework called jCasbin,[16] and then specify the integration point with the IoT platform in use (with GeneSIS support, see Fig. 4.9). During the deployment process, GeneSIS compiles the integration code before orchestrating the deployment of the integrated components.

To enable such DevSecOps adaptation support, GeneSIS not only provides the modelling language embedded in a web UI for specifying the components of such IoT platforms, but also the reconfiguration and rebuild of these components (for integrating new security mechanisms with the IoT platform) before deployment (for adaptation or for a new development cycle). For example, in the SMOOL

16. https://casbin.org/

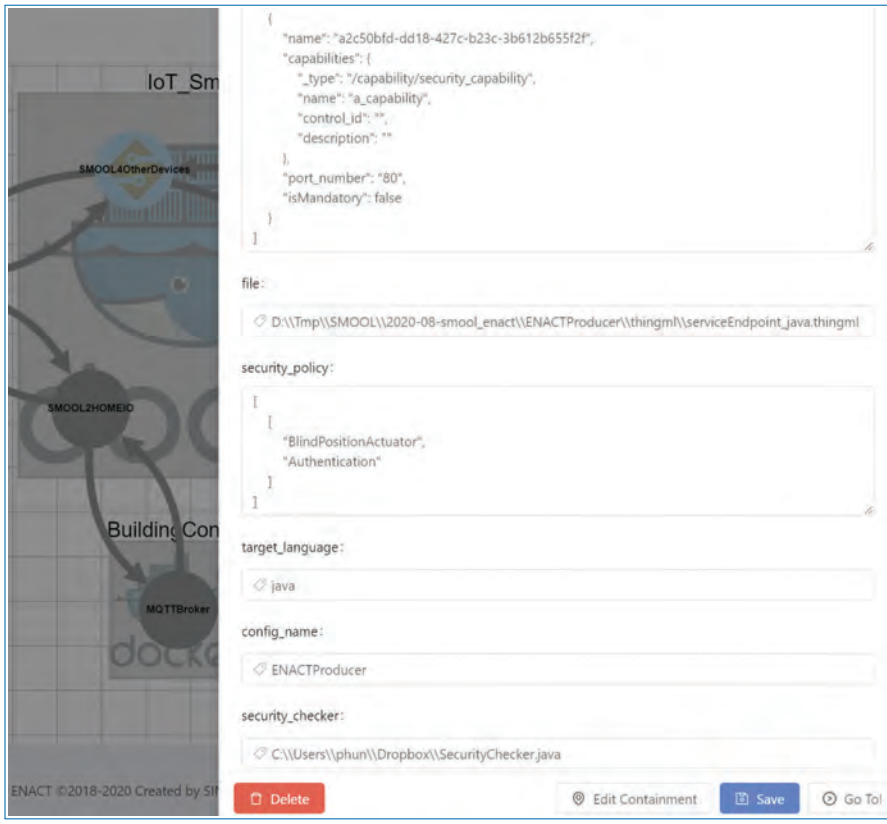


Figure 4.10. An enhanced security control has been added.

platform, each SMOOL producer or consumer is associated with a security checker for checking the security key of sensor data or actuation commands. GeneSIS allows updating the configuration of the security checker (*e.g.*, by injecting new configuration to overwrite the default one), and automatically rebuilding the SMOOL producer or consumer including the reconfigured security checker. By doing so, GeneSIS enables the DevOps team to make reconfiguration or redevelopment and redeployment easily for the evolution of SMOOL producers or consumers including security checkers. Figure 4.10 shows an example of the GeneSIS's UI for extending the *SecurityChecker* (in *SMOOL2HOMEIO*) to become a security enforcement point of the external access control service. Thanks to ThingML support within GeneSIS, the extended *SecurityChecker* is compiled in the *SMOOL2HOMEIO* component for a new version of *SMOOL2HOMEIO* to be deployed that works as a security enforcement point of the external access control service.

This approach is what we call the DevSecOps adaptation support for the co-evolution of business logic components and the security mechanisms. This means

that when new business logic components require security mechanisms to evolve, GeneSIS can support for the adaptation, even including the integration of the IoT platform with other (third-party) security mechanisms.

After the successful deployment, GeneSIS allows dynamic adaptations that can be triggered at any point, manually or automatically for adapting security enforcement to improve how the IoT system operates securely. In this line of adaptation, new security policies or configurations can be updated dynamically for the security mechanisms that are in operation. For example, the role-based access control policy can be easily updated according to new requirements. The trigger of such adaptation can be manually, but also can be automatically from a risk assessment process or after a reasoning process of actuation conflict management.

In summary, with the support from GeneSIS, the DevOps team can develop a new version of the smart home system together with enhanced security mechanisms according to its evolution. The deployment of this new development cycle can be triggered manually from GeneSIS's GUI. After the successful deployment, GeneSIS also allows dynamic adaptations that can be triggered at any point, manually or automatically for adapting security enforcement to improve how the IoT system operates securely. In both ways presented so far, GeneSIS allows DevSecOps teams to reconfigure and update security mechanisms by design, in line with the evolution of IoT applications and the development of security and privacy risks.

It is important to note that in this chapter we have not addressed the security of the build and deployment pipeline itself. The security of this pipeline is critical to protect the integrity of the code and the systems being deployed. For the production environment, GeneSIS must adhere to the secure deployment practice.[17] One of the main principles in secure deployment is to support automatic testing as part of the deployments to gain confidence in the security of the code (see Section 8.2 for Test and Simulation).

4.4.3 Software Diversity Within IoT Fleet

Software diversity in an IoT fleet, i.e., deploying variants of software on different devices, creates a moving target for malicious attacks, and therefore improves the overall security of the system. The DivEnact tool assigns the available variants to the fleet of devices and maintains the balance between the variants. The remaining questions is how to obtain functionally-equivalent variants.

The ENACT IoT diversity-by-design tool takes as input a single deployment or behaviour specification and generates multiple diverse specifications. Within a DevOps context, it is important and necessary to keep the diversity generation fully

17. https://owaspsamm.org/model/implementation/secure-deployment/

automatic, instead of relying on developer's manual effort to diversify systems (such as the traditional N-Version Programming approach). Developers can focus on a single line of code to achieve frequent iteration, and the diversification tool, as part of automatic building step, will generate diversified versions automatically [14].

Automated diversity is a promising means of mitigating the consequences of a security breach. However, current automated diversity techniques operate on individual processes, leveraging mechanisms available at the lower levels of the software stack (in operating systems and compilers), yielding a limited amount of diversity. In this section, we present a novel approach for the automated synthesis of diversified protocols between processes. This approach builds on (i) abstraction, where the original protocol is modelled by a set of communicating state machines, (ii) automated synthesis, applying mutation operators onto those protocols, which produces semantically-equivalent, yet phenotypically-different protocols, and (iii) automated implementation of these protocols through code generation.

The tool is currently in an experimental stage. Automatic diversity of communication protocols is a novel technology, yet without convincing implementation and applications, to the best of our knowledge. Therefore, our focus is currently on the theoretical feasibility of the idea and the experimental evaluation of its effects. In the next step, we will improve the user experience of the tool and its applicability to practical scenarios.

Mass-produced software applications denote clonal applications, with thousands or millions of identical siblings. Think of, for example, a popular mobile application installed on millions of mobile phones, or software embedded into a widely-used connected device. To mitigate the risks of such large mono-cultures, diversity is typically automatically introduced either in a generic way, typically at the OS level, oblivious of the actual logic and semantics of the software, or in some very specific places, typically low-level libraries reused across applications, in order to improve security. This leaves most of the actual business logic unchanged, unaffected by the diversity. In addition, diversity often affects individual processes, but leaves the communication between processes intact.

A more holistic approach to diversity is challenging. Consider a typical client-server application, where multiple clients interact with a server, and where each client has a different implementation, and a different way of communicating with the server. This would significantly hinder a hacker, be it a human being or a machine, when attempting to generalize an attack through all possible protocols. This would make large-scale exploits a time-consuming and costly endeavour for hackers. Yet, the engineering, e.g., the production, maintenance and integration, of such levels of diversity raises several challenges. How to ensure that each implementation still behaves as specified? How to ensure that each client is still able to communicate with the server, without information loss or distortion? How to

ensure that different clients are fundamentally (i.e., sufficiently) different, and not merely cosmetically different? How to keep the development and operation costs of a diversified system significantly lower than the cost of mitigating large scale attacks?

We have seen that abstraction, synthesis and automated implementation can yield a convincing solution to introduce a wide diversity into protocols, for example between a device and a gateway, or a web/mobile app and a server. This approach:

- abstracts protocols into (i) a structural view describing the messages to be exchanged, and (ii) a behavioral view based on state machines describing how those messages are exchanged between the participants, including sequencing and timing.
- combines and applies a number of atomic mutations to this protocol model, yielding a large number of diversified protocols, which operate differently, still with the same semantics.
- automatically implements protocols, diversified or not, by generating fully operational code targeting C, Go, Java and JavaScript, able to run on a wide range of platforms.

Our empirical assessment indicated that this approach implies a reasonable overhead in terms of execution time, memory consumption and bandwidth, fully compatible with the requirements of mass-produced software. We also showed that this approach could generate a significant amount of diversity. Our assumption was that this diversity would contribute to the diversity-stability hypothesis, i.e., this would make the whole ecosystem more robust by making it less likely for an exploit to propagate to the whole population. In other words, if the protocol between a specific client and the server could be observed, analysed and eventually understood, this would not systematically imply that all other diversified protocols could be understood following the very same procedure. In this section, we briefly describe the mechanisms and the corresponding tools we developed to automatically generate the diverse protocols. Technical details and the experiment results can be found in our conference paper [29].

Our approach relies on ThingML [26] for the specification of protocols. ThingML provides a way to formalize the messages involved in protocols, in a comparable way to what Protocol Buffer proposes. In addition, ThingML provides a mean to formalize the behavior of protocols through state machines. ThingML specifications are both human-readable and machine-readable, which makes it possible to analyse protocols at a high-level of abstraction and to fully automate the implementation of those protocols through code generation. In the next-subsection, we present relevant aspects of ThingML on our motivating example.

We model communication protocols as a set of communicating state-machines, encapsulated into components. A protocol typically involves two roles: (i) a client, i.e., a device, a web-browser or a mobile app, and (ii) a server, i.e., a gateway or a Cloud back-end. The clients and the server need to agree on a common API. Since communication is typically asynchronous in a distributed system, the common API is specified as a set of messages. Next, this API is imported by the client component and the server component, and the messages are organized into ports.

The ultimate goal of our approach is to diversify the wire image of protocols. Diversifying the wire image of protocols basically means shuffling the sequence of bytes exchanged over the network e.g., turning the payloads while ensuring the interoperability between the client and the server.

4.5 Conclusions

This chapter summarizes our effort in the ENACT project towards automatic software deployment for Smart IoT Systems. Automatic deployment is a cornerstone of DevOps, as it connects development with operation, and ensures that changes on the software will be placed into the production in a correct and prompt way.

Although there are already mature deployment solutions for Cloud computing in the market, automatic deployment for smart IoT systems is still an open problem. The main challenges are from two fundamental characters of smart IoT systems: First, an IoT application involves software running at all types of resources along the Cloud-Edge-IoT continuum, and it is difficult to provide a consistent way to support the deployment on all those different types of resources. Second, an IoT application in the production stage usually contains many subsystems of Edge and IoT devices, each of which serves a particular user or manages a particular part of the physical world. It is difficult to deploy a new change on the software to all those subsystems regardless of the different contexts and status among them.

During the ENACT project, we conducted research aiming at these two challenges, resulting in two ENACT enablers, namely GeneSIS and DivEnact. We briefly introduced how these enablers work, both as individual tools and as an integrated deployment bundle for the automatic deployment of SIS. More details about the theories, implementations and use cases can be founded in our recent publications [15, 44]. In this chapter, we focused on the mechanisms and practices of using these tools to ensure the trustworthiness of the deployment software, including the availability of software components on unstable resources, the deployment support of security and privacy mechanisms, and the automatic generation and maintenance of software diversity towards a more secure systems.

In the next step, we will extend the concepts and implementation of automatic deployment into the more general edge computing domain, providing an engineering solution for the core problem of edge computing, i.e., the distribution and offloading of computation among the complex and dynamic resources. Currently, the deployment is driven by manually define deployment models which embeds the resource allocation and the constraints about software-device mapping. An important future plan is to introduce intelligence into automatic deployment, which learns from historical deployments and their effects to automatically assign software parts to the proper resources.

References

[1] C. R. Aguayo Gonzalez, C. B. Dietrich, and J. H. Reed. "Understanding the software communications architecture". In: *IEEE Communications Magazine* 47.9 (2009), pp. 50–57.

[2] Carlos Ansótegui *et al.* "Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem". In: *Ninth Symposium of Abstraction, Reformulation, and Approximation*, 2011.

[3] Matej Arta *et al.* "Model-driven continuous deployment for quality devops". In: *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*. 2016, pp. 40–41.

[4] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. 3rd. Addison-Wesley Professional, 2012. ISBN: 0321815734.

[5] Benoit Baudry and Martin Monperrus. "The Multiple Facets of Software Diversity: Recent Developments in Year 2000 and Beyond". In: *ACM Comput. Surv.* 48.1 (Sept. 2015). ISSN: 0360-0300. DOI: 10.1145/2807593. URL: https://doi.org/10.1145/2807593.

[6] Gordon S. Blair, Nelly Bencomo, and Robert B. France. "Models@run.time". In: *IEEE Computer* 42.10 (2009), pp. 22–27.

[7] Miquel Bofill *et al.* "Solving constraint satisfaction problems with SAT modulo theories". In: *Constraints* 17.3 (2012), pp. 273–303.

[8] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. "Satisfiability modulo theories and assignments". In: *International Conference on Automated Deduction*. Springer. 2017, pp. 42–59.

[9] Antonio Bucchiarone, Antonio Cicchetti, and Annapaola Marconi. "Exploiting multi-level modelling for designing and deploying gameful systems". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2019, pp. 34–44.

[10] Sylvain Cherrier *et al.* "D-lite: Distributed logic for internet of things services". In: *2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*. IEEE. 2011, pp. 16–24.

[11] F. Cirillo *et al.* "A Standard-Based Open Source IoT Platform: FIWARE. In: *IEEE Internet of Things Magazine* 2.3 (2019), pp. 12–18. DOI: 10.1109/IOTM.0001.1800022.

[12] Benoit Combemale and Manuel Wimmer. "Towards a Model-Based DevOps for Cyber-Physical Systems". In: *Software Engineering Aspects of Continuous Development*, 2019.

[13] Rustem Dautov and Hui Song. "Towards Agile Management of Containerised Software at the Edge". In: *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*. Vol. 1. IEEE. 2020, pp. 263–268.

[14] Rustem Dautov and Hui Song. "Towards IoT Diversity via Automated Fleet Management". In: *MDE4IoT/ModComp@ MoDELS*. 2019, pp. 47–54.

[15] Rustem Dautov, Hui Song, and Nicolas Ferry. "A Light-Weight Approach to Software Assignment at the Edge". In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2020, pp. 380–385.

[16] Johannes Eder *et al.* "Bringing DSE to life: exploring the design space of an industrial automotive use case". In: *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2017, pp. 270–280.

[17] Johannes Eder *et al.* "From deployment to platform exploration: automatic synthesis of distributed automotive hardware architectures". In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2018, pp. 438–446.

[18] Nicolas Ferry and Phu H. Nguyen. "Towards Model-Based Continuous Deployment of Secure IoT Systems". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 2019, pp. 613–618.

[19] Nicolas Ferry *et al.* "CloudMF: Model-Driven Management of Multi-Cloud Applications". In: *ACM Transactions on Internet Technology (TOIT)* 18.2 (2018), p. 16.

[20] Nicolas Ferry *et al.* "Continuous Deployment of Trustworthy Smart IoT Systems". In: *The Journal of Object Technology* (2020).

[21] Nicolas Ferry *et al.* "Genesis: Continuous orchestration and deployment of smart IoT systems". In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2019, pp. 870–875.

[22] M. Frustaci *et al.* "Evaluating Critical Security Issues of the IoT World: Present and Future Challenges". In: *IEEE Internet of Things Journal* 5.4 (2018), pp. 2483–2495. DOI: 10.1109/JIOT.2017.2767291.

[23] Anne Gallon *et al.* "Making the Internet of Things More Reliable Thanks to Dynamic Access Control". In: *Security and Privacy in the Internet of Things: Challenges and Solutions* 27 (2020), p. 61.

[24] Nam Ky Giang *et al.* "Developing IoT applications in the fog: a distributed dataflow approach". In: *Internet of Things (IoT), 2015 5th International Conference on the*. IEEE. 2015, pp. 155–162.

[25] Object Management Group. "Deployment and Configuration of Component-based Distributed Applications Specification". In: *OMG Available Specification Version 4.0 formal/06-04-02* (2006).

[26] Nicolas Harrand *et al.* "ThingML: A Language and Code Generation Framework for Heterogeneous Targets". In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. MODELS 16. Saint-malo, France: Association for Computing Machinery, 2016, pp. 125–135. ISBN: 9781450343213.

[27] Stéphane Lavirotte *et al.* "A generic service oriented software platform to design ambient intelligent systems". In: *Proceedings of the 2015 ACM International Conference on Pervasive and Ubiquitous Computing*. ACM. 2015, pp. 281–284.

[28] Amardeep Mehta *et al.* "Calvin Constrained-A Framework for IoT Applications in Heterogeneous Environments". In: *37th International Conference on Distributed Computing Systems*. IEEE. 2017, pp. 1063–1073.

[29] Brice Morin *et al.* "Engineering software diversity: A model-based approach to systematically diversify communications". In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2018, pp. 155–165.

[30] Håvard Myrbakken and Ricardo Colomo-Palacios. "DevSecOps: A Multivocal Literature Review". In: *Software Process Improvement and Capability Determination*. Ed. by Antonia Mas *et al.* Cham: Springer International Publishing, 2017, pp. 17–29. ISBN: 978-3-319-67383-7.

[31] NESSI. *Cyber physical systems: Opportunities and challenges for soft- ware, services, cloud and data*. NESSI White paper. 2015.

[32] P. H. Nguyen *et al.* "SoSPa: A system of Security design Patterns for Systematically engineering secure systems". In: *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2015, pp. 246–255. DOI: 10.1109/MODELS.2015.7338255.

[33] Phu H. Nguyen, Phu H. Phung, and Hong-Linh Truong. "A Security Policy Enforcement Framework for Controlling IoT Tenant Applications in the Edge". In: *Proceedings of the 8th International Conference on the Internet of Things*, IOT 18. Santa Barbara, California, USA: ACM, 2018. ISBN: 9781450365642.

[34] Phu H. Nguyen *et al.* "Advances in deployment and orchestration approaches for IoT – A systematic review". In: *2019 IEEE International Congress On Internet of Things (ICIOT)*. Milan, Italy: IEEE, 2019, pp. 53–60.

[35] Phu H. Nguyen *et al.* "An extensive systematic review on the Model-Driven Development of Secure Systems". In: *Information and Software Technology* 68 (2015), pp. 62–81. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2015.08.006. URL: http://www.sciencedirect.com/science/article/pii/S0950584915001482.

[36] Adrian Noguero, Angel Rego, and Stefan Schuster. "Towards a Smart Applications Development Framework". In: *Social Media and Publicity* 27 (2014). URL: https://bitbucket.org/jasonjxm/smool,%202011-2020.

[37] OMG. *Deployment and Configuration of Component-based Distributed Applications Specification, v4.0*. Tech. rep. Object Management Group, Inc., 2006. URL: https://www.omg.org/spec/DEPL/4.0/PDF.

[38] Temel Öncan. "A survey of the generalized assignment problem and its applications". In: *INFOR: Information Systems and Operational Research* 45.3 (2007), pp. 123–141.

[39] Allen Parrish, Brandon Dixon, and David Cordes. "A conceptual foundation for component-based software deployment". In: *Journal of Systems and Software* 57.3 (2001), pp. 193–200. ISSN: 0164-1212.

[40] David W Pentico. "Assignment problems: A golden anniversary survey". In: *European Journal of Operational Research* 176.2 (2007), pp. 774–793.

[41] Subhav Pradhan *et al.* "Chariot: Goal-driven orchestration middleware for resilient IoT systems". In: *ACM Transactions on Cyber-Physical Systems* 2.3 (2018), pp. 1–37.

[42] Davide Di Ruscio, Richard F. Paige, and Alfonso Pierantonio, eds. *Special issue on Success Stories in Model Driven Engineering*. Vol. 89, Part B. Elsevier, 2014.

[43] Ana C Franco da Silva *et al.* "OpenTOSCA for IoT: automating the deployment of IoT applications based on the mosquitto message broker". In: *Proceedings of the 6th International Conference on the Internet of Things*. ACM. 2016, pp. 181–182.

[44] Hui Song *et al.* "Model-based fleet deployment of edge computing applications". In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2020, pp. 132–142.

[45] Hui Song *et al.* "On architectural diversity of dynamic adaptive systems". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. IEEE. 2015, pp. 595–598.

[46] Michael Vögler *et al.* "A scalable framework for provisioning large-scale IoT deployments". In: *ACM Transactions on Internet Technology (TOIT)* 16.2 (2016) pp. 1–20.

[47] Changsheng You *et al.* "Energy-efficient resource allocation for mobile-edge computation offloading". In: *IEEE Transactions on Wireless Communications* 16.3 (2016) 1397–1411.

DOI: 10.1561/9781680838251.ch5

Chapter 5

A DevOps Toolchain for Managing Actuation Conflicts in Smart IoT Systems

By Gérald Rocher, Thibaut Gonnin, Franck Dechavanne, Stéphane Lavirotte and Jean-Yves Tigli

5.1 Introduction

To assist users in their daily lives, Smart IoT Systems (SIS) have long been limited to the exploitation of environmental information; the use of 'smart objects' was mainly motivated by their ability to collect these information from sensors. However, in many areas such as home automation, factory 4.0, Intelligent Transportation Systems (ITS), etc., SIS are no longer limited to collecting sensor data to infer actionable information, they also interact with the physical environment through actuators. This evolution brings new challenges that the scientific community has to meet in collaboration with industrial players, as evidenced by the numerous calls from the European Commission, in which the ENACT project is part of: *"Most of the today's IoT systems are however mainly focused on sensors, whereas in the future actuation and smart behaviour will be the key points. Platforms should provide connectivity and intelligence, actuation and control features"* [9], p. 100.

5.1.1 SIS Actuation Challenges

By physically interacting with their environment through actuators, SIS become critical; in the absence of end-to-end human control (utopian in complex operational contexts such as ITS, etc.), they have immediate impacts on the physical environment with all the social and economical risks that this may entail. At design-time, these applications must be conceived to formally prevent any undesirable effect in the physical environment, whether caused by sending contradictory or simultaneous commands to the actuators. At run-time, it is yet necessary to ensure that the effects produced in the physical environment are in line with the expectations. Indeed, the lack of a perfect model of the physical environment (of a complex nature) prevents designers from fully predicting the effects of the commands sent to the actuators; effects that are therefore likely to be hampered by possibly disruptive surrounding physical processes. These challenges bring with them heightened concerns about *trustworthiness* of SIS which includes, among others, *safety* and *reliability* aspects. As defined in [13], safety concerns are related to the ability of SIS to prevent catastrophic consequences for humans and the physical environment; reliability concerns are related to the ability of SIS to deliver predictable performance in expected conditions. While the DevOps methodology is part of the good practices in software development and is applicable to SIS limited to merely collect sensor data, it requires new tools in the realm of trustworthy SIS, both at Dev and Ops Times.

5.1.2 DevOps Still Lacks the Tools to Meet These Challenges

While the DevOps approach is not specific to a particular field of applications, many challenges arise when it comes to applying it to the field of SIS. DevOps practices are therefore still far from being fully adopted in their development, notably due to a lack of key enabling tools [1, 34]. Among these key tools, those capable of taking into account SIS operating in open and complex environments and requiring continuous testing at run-time (i.e., in-situ) beyond the tests traditionally conducted on emulated and simulated infrastructures, are missing [1]. In general, there is a lack of tools that address the trustworthiness of SIS, which, as far as this chapter is concerned, is about safety and reliability aspects, exacerbated by the ability of SIS to act in the physical environment through actuators. A study of the literature around the actuation problem [19] shows that, without even considering the DevOps approach, this problem is still in its infancy in the IoT field and therefore still open. Moreover, the approaches proposed in the literature for managing actuation in SIS are often (1) monolithic, they do not or hardly meet the best practices advocated by the DevOps approach; (2) applied to controlled operational environments; they have a software vision of the problem by focusing on the commands sent to the actuators more than on the effects they produce [29].

The work carried out as part of the ENACT European project and described throughout this chapter aims to fill this gap by proposing a toolchain meant to be integrated into the DevOps framework and meeting end-to-end SIS actuation challenges.

5.1.3 An End-to-end DevOps Toolchain

The contribution is built around two complementary toolsets, deployed throughout the DevOps life-cycle phases. This combination of tools aims to improve the trustworthiness of SIS within a framework which, as advocated by the DevOps approach, creates synergies and foster communication between development and operations activities.

At design-time (Dev), a complete toolset is developed for identifying and locally resolving actuation conflicts [12], i.e., (a) preventing contradictory or simultaneous commands to be sent to actuators (direct conflicts), (b) preventing, as much as it can be, antagonistic effects to occur in the physical environment (indirect conflicts). At run-time (Ops), a first tool observes specific environmental features and quantitatively assesses the effectiveness of the SIS, i.e., for the extent to which it produces the expected effects. A second tool is meant to analyse drifts in effectiveness. It makes clear the symptoms of the drifts in effectiveness, providing guidance to designers to help them investigate their possible root causes. Within the framework of the ENACT project, other investigation tools may be used to complement this latter tool, such as, for instance, the Root Cause Analysis (RCA) (Section 8.3) toolset. The analysis resulting from these tools then trigger a new design phase, closing the DevOps life-cycle loop.

The rest of the chapter is organized as follows. Section 5.2 describes the SIS actuation management toolset along with the workflow (Section 5.2.2) involved in the identification and resolution of direct and indirect actuation conflicts. Section 5.3 describes the behavioural drift assessment (Section 5.3.2) and analysis (Section 5.3.3) toolset. Section 5.4 demonstrates, throughout a smart-home use-case described in Section 5.4.1, the complementarity and relevancy of both toolset as part of the DevOps life-cycle. This use-case involves two consecutive DevOps cycles to converge towards a SIS with satisfactory behaviour.

5.2 Overview of the SIS Actuation Conflict Management Toolset

IoT devices, at the edge of SIS infrastructures, have long been leveraged for their capacity at gathering environmental data from sensors, paving the way for decision making support systems covering a broad range of application domains from smart-health, smart-city to smart-grid, Factory 4.0, etc. just to name a few. However,

beyond merely gathering data from sensors, new challenges arise as soon as it comes to leverage IoT devices to interact with the physical environment through actuators that turn commands received from SIS into physical effects. One of these challenges concerns the actuation conflict management. Such conflicts are likely to occur when different applications compete for accessing (1) shared actuators at the edge of the IoT infrastructure (direct conflicts), e.g., simultaneously applying ON and OFF to a light bulb, and/or (2) shared physical properties (indirect conflicts), i.e., turning ON both a cooler and a heater in the same room.

The actuation conflict management challenge goes beyond the technological challenge that shared multi-layered IoT infrastructures usually meet by, for instance, providing sensors with access control mechanisms. Indeed, besides this technological challenge, actuation conflict management also poses a semantical challenge; accounting for the locality of the actuators and the physical properties they act on is here essential. In the realm of trustworthy SIS, actuation conflict management is of paramount importance. Designers must prevent SIS from producing any undesirable effects in the physical environment, whether it is caused by sending contradictory or simultaneous commands to the actuators. To this end, designers must be provided with decision support tools that can assist them in identifying and resolving direct and indirect actuation conflicts, and in deploying relevant, yet robust and safe Actuation Conflict Managers (ACM).

In the sequel, a complete toolset for identifying and resolving actuation conflicts is introduced. This toolset consists of three stages throughout the DevOps life-cycle, as described in Fig. 5.1).

Underlying the tools for actuation conflict identification and resolution, a meta-model, denoted Workflow and Interaction Model for Actuation Conflict management (WIMAC), is used to build a structural model of the SIS upon deployment, implementation and physical environment models (phase 1 in Fig. 5.1) [27]. WIMAC provides a modelling language for describing inter-relationships between SIS software components and actuators at the edge of the infrastructure along with their effects on the physical environment.

On the basis of the structural model, potential direct and indirect actuation conflicts are identified (phase 2 in Fig. 5.1). Actuation conflict identification is based on Attributed Graph Grammar (AGG) rules [18] meant to detect conflicting patterns; actuation conflict resolution is based on AGG rewriting rules meant to instantiate local ACMs. DevOps approach aims to provide continuous and rapid software deployment capabilities. In accordance with this approach, a set of pre-configured off-the-shelf and ready-to-use ACMs are offered to designers.

Third, besides off-the-shelf ACMs, a complete formal verification flow is proposed in the Discrete EVent system Specification formalism (DEVS) [39] for

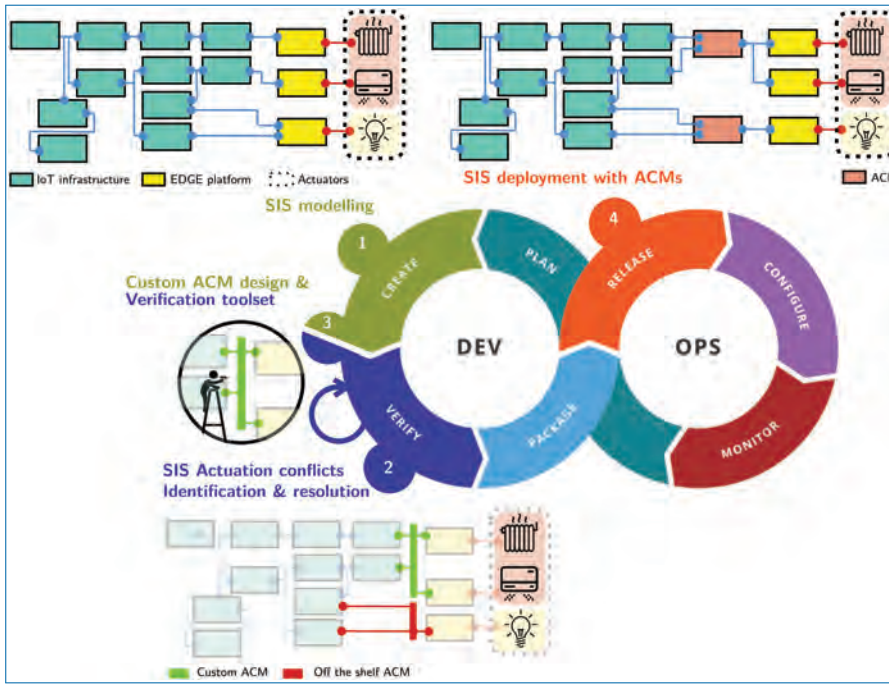


Figure 5.1. Usage of Actuation Conflict Management (ACM) tool-set throughout the DevOps life-cycle.

designing and verifying reusable custom ACMs, covering different implementation strategies at the edge of the infrastructure (phase 3 in Fig. 5.1).

Finally, the WIMAC-based structural model, with ACMs instantiated, is transformed back to deployment and implementation models (phase 4 in Fig. 5.1).

5.2.1 Beyond the State of the Art

The problem of identifying and resolving IoT-based systems actuation conflicts is still in its infancy. While most of the existing solutions focus on the identification and the resolution of direct actuation conflicts (a legacy of the technical challenge mentioned above), few focus on identifying and resolving indirect ones [19, 21]. Moreover, the solutions proposed in the literature raise two main problems when it comes to applying them to the DevOps framework; (1) most of them require an a priori knowledge on the system components and the rules governing their evolution [33]; (2) the solutions proposed are thereby monolithic, they implement global identification and resolution mechanisms not easily reusable (e.g., [2, 21, 22]).

In [24], authors recognize that interactions between IoT devices are an increasing cause of safety and security violations whose detection *"[…] requires a holistic view of installed apps, component devices, their configurations, and more importantly,*

how they interact". This is the approach followed by the SIS actuation conflict management toolset presented in the sequel.

5.2.2 Actuation Conflict Management Workflow

The actuation conflict management workflow is depicted in Fig. 5.2.

DevOps approach provides designers with deployment and implementation models from which it is possible to extract the structural interactions between the SIS software components down to the devices and the actuators they embed. To identify indirect conflicts, however, it is necessary to describe the effects that the actuators produce in the environment in which they operate. Unless each actuator is accompanied by semantic annotations, these descriptions have to be provided by the designers, using a model of the physical environment.

From these models, an holistic description model is generated. This model is built on a metamodel called WIMAC (Workflow and Interaction Model for Actuation Conflict management). This metamodel (Fig. 5.3) provides a language to describe (1) the inter-relationships between software components, down to the

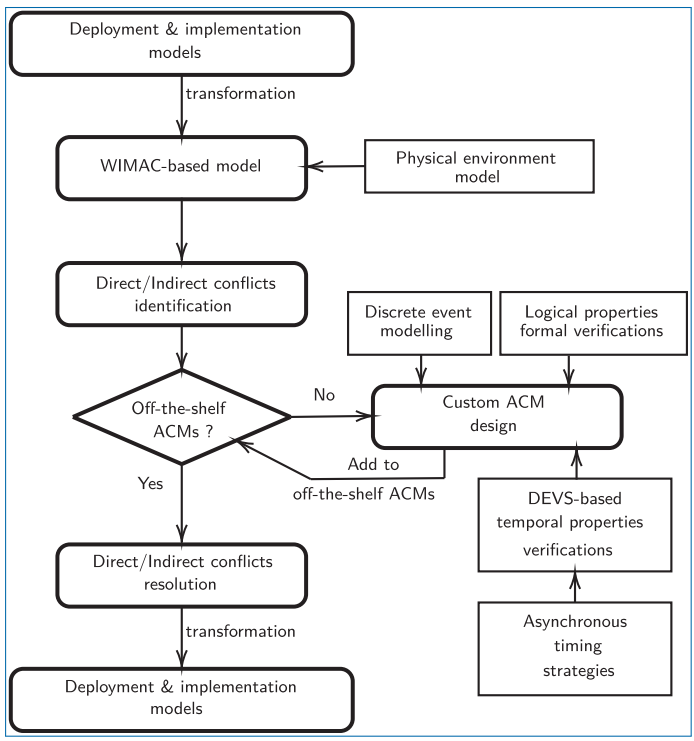


Figure 5.2. Actuation conflicts identification and resolution workflow.

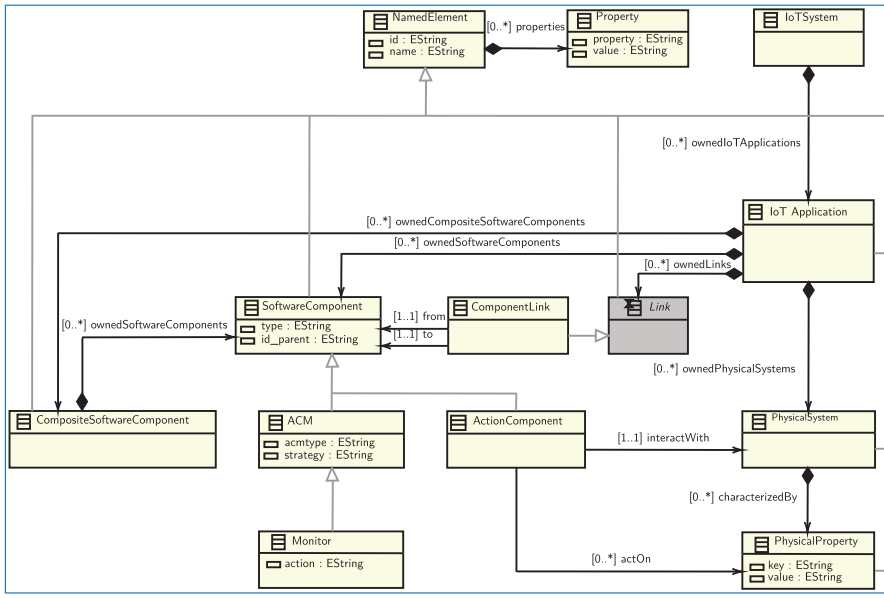


Figure 5.3. WIMAC meta-model.

actuators at the edge of the infrastructure as well as (2), the effects that the actuators produce in the physical environment in which they operate. The WIMAC metamodel main entities are the following:

1. **SoftwareComponents** are black-box components. They conceptualize single applications or composite applications described through implementation models (e.g., Node-RED[1] flow). The actuation conflict management solution proposed in the sequel is based solely on the structural links that exist between software components for inferring their interrelationships, identifying and resolving potential conflicts.
2. **ActionComponents** are SoftwareComponents controlling actuators,
3. **PhysicalSystems** are spatially delimited physical entities whose properties can be changed by ActionComponents (e.g. the temperature in the kitchen).

Relying on the WIMAC-based model, potential direct and indirect actuation conflict points can be identified and monitoring ACMs instantiated locally. This first step is automatically achieved through a set of predefined Attributed Graph Grammar (AGG) rules [18] in the form of attributed graphs. These rules define (1) conflict patterns to be identified in the WIMAC-based model and (2) associated

1. https://nodered.org

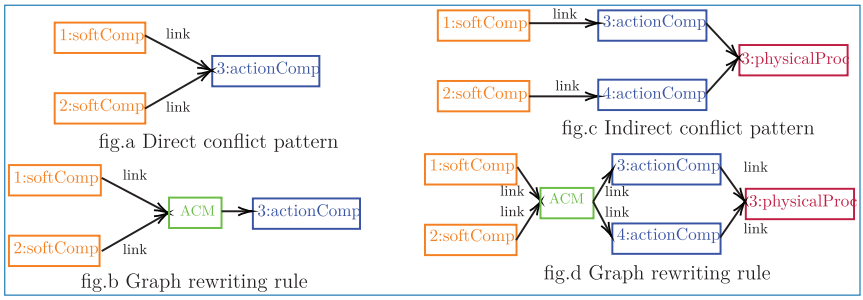


Figure 5.4. Example of AGG identification and rewriting rules for direct (left) and indirect (right) actuation conflicts.

graph transformations to be applied on the model so as to instantiate monitoring ACMs (Fig. 5.4).

DevOps approach aims to provide continuous and rapid deployment capabilities. To support this objective, a set of off-the-shelf generic ACMs are proposed to designers thereby, replacing dummy ACMs with relevant concrete ones. Although this approach requires designers to take responsibility on the ACMs to be instantiated before applications are deployed, it has several advantages. (1) it helps reinforcing SIS trustworthiness, specifically as actuation conflict management implies some semantic background humans are better able to grasp considering SIS complexity globally; (2) this is all the more important as ACMs are instantiated locally, addressing points of potential conflict in the design. The advantage here is that software components are regarded as black boxes, ACMs do not change their logic.

While off-the-shelf reusable ACMs are relevant for resolving common actuation conflicts, they may not be suitable for some particular cases. To address this concern, state-of-the-art Model Driven Engineering (MDE) tools are leveraged, allowing designers to develop custom, yet robust and safe ACMs, throughout a two-level design workflow (Fig. 5.5).

At the first level, logic of the custom ACMs are defined through Finite State Machines (FSM) built on the basis Event-Condition-Action rules. These conceptual models allow logical properties (e.g. completeness, safety, liveness, etc.) to be formally verified using state-of-the-art methodologies [3].

At the second level, implementation models allow temporal properties to be formally verified by applying different asynchronous timing strategies. Implementation models are proposed to be described through the DEVS formalism (Discrete EVent system Specification)[39]. This formalism brings key advantages in the realm of trustworthy SIS and DevOps.

1. DEVS atomic models allow to encapsulate conceptual models into DEVS Atomic models, coupled with synchronizers that can implement different

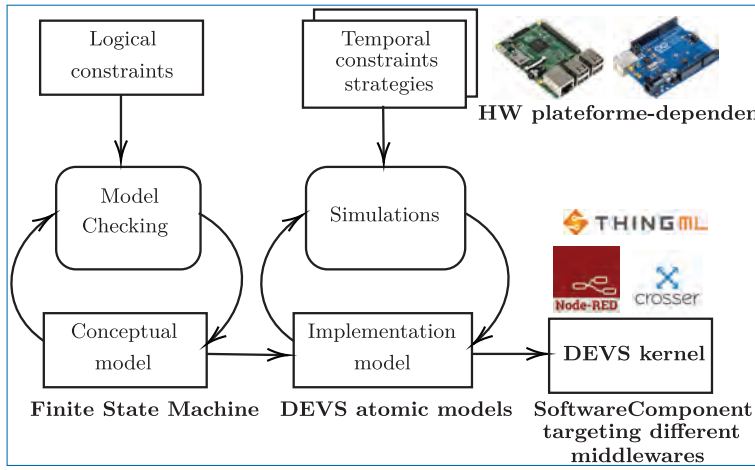


Figure 5.5. Custom ACM design workflow.

timing strategies on the inputs of the FSM and targeting different hardware platforms. This approach is particularly relevant considering that SIS software components are likely to be deployed on resource-constrained platforms at the edge of the IoT infrastructure, governed by asynchronous events relative to wireless communication protocols, computational capabilities, etc. [35]. Several modelling and simulation tools are available to facilitate this process [6, 25, 36],

2. It provides a common representation to different discrete event modelling formalisms (including Petri Nets, FSM, etc.) [41]. Designers are therefore not bounded to a particular modelling framework when designing custom ACMs.
3. DEVS Atomic models, once verified for temporal properties, can be translated into any high-level programming language (e.g., C, C++, C#, etc.) and compiled for further deployment. This allows to build a library of reusable off-the-shelf DEVS-based ACM software components (a.k.a., DEVS kernels) targeting different implementation strategies (i.e., hardware platforms).

5.3 Overview of the SIS Behaviour Monitoring and Analysis Toolset

While, during the development phase, the ACM tool allows for the identification, analysis and resolution of actuation conflicts, it remains, however, based on the global architecture of the SIS and its operational environment, i.e., and is

therefore based on a priori acquired knowledge. However, by interacting with the physical environment through actuators, SIS inherits the *complex* nature of this open environment. Thus, assuming that the knowledge acquired a priori is complete and immutable is illusory, no matter how complicated the associated models may be; an infinite number of unexpected physical processes are likely to disrupt, at any time, operation of the SIS. In the realm of trustworthy SIS, the ACM tool alone thus cannot meet the concerns of reliability and safety. It is therefore necessary to complement this tool with a systemic approach, required to the modelling of complex SIS [30].

Without being able to predict the behaviour of SIS in their design phase, a DevOps tool-set is developed for analysing SIS behavioural drifts at run-time (Fig. 5.6). The aim of the first tool described in the sequel is to quantitatively assess SIS *effectiveness* at run-time, i.e., the extent to which the effects produced in the physical environment are legitimate (phase 2 in Fig. 5.6). It is no longer a question of describing the global architecture of the SIS, but of describing a

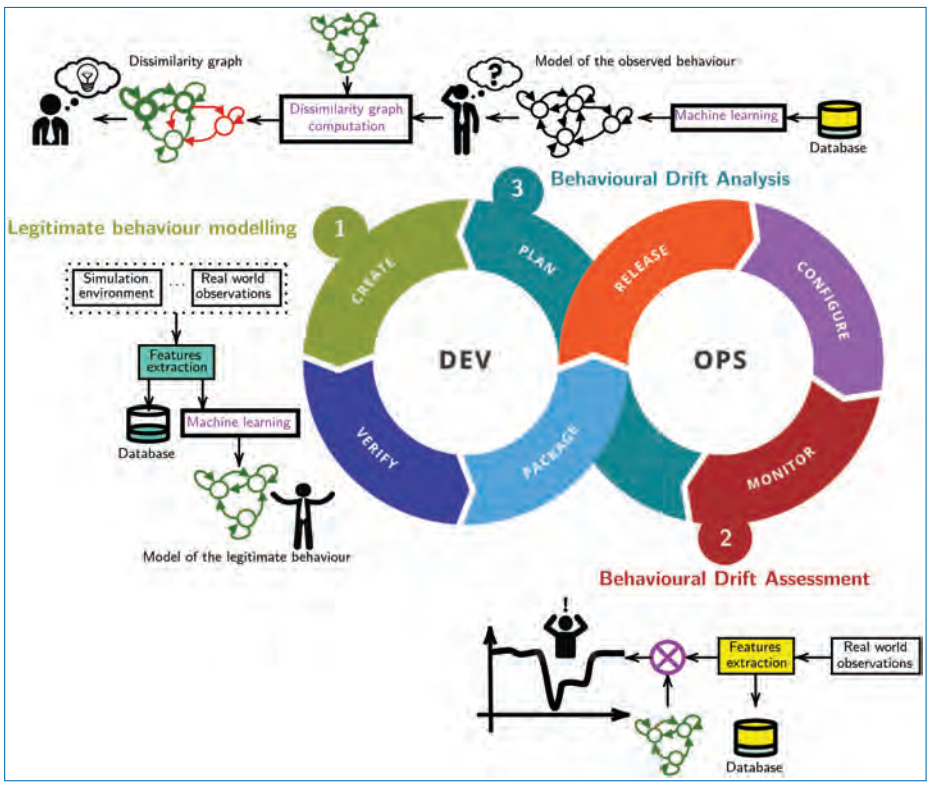


Figure 5.6. Usage of the Behavioural Drift Analysis (BDA) tool-set throughout the DevOps life-cycle.

model of the effects they have to legitimately produce in the physical environment in different contexts (phase 1 in Fig. 5.6). However, being quantitative in nature, the effectiveness assessment is difficult to interpret by humans and does not allow for creating synergies and fostering communication between development and operations activities, as advocated by the DevOps approach. Indeed, from the designers' point of view, while the effectiveness assessment allows the detection of unexpected and/or abnormal behaviour at run-time, it does not allow their causes to be identified. Thereby, this hampers a new DevOps cycle from being initiated rapidly so as to implement the corrective actions required. To address this concern, a second tool is presented in the sequel. This tool (phase 3 in Fig. 5.6), produces a model of the behaviour observed in the field and compares this model with the model of the legitimate behaviour produced in phase 1. This results in a dissimilarity graph between both models which, without identifying the causes of the drifts in effectiveness, sheds light on the symptoms likely to explain them [29].

5.3.1 Beyond the State of the Art

The problem raised in this research, and the solution described in the sequel, concerns the detection, identification and intelligible representation of the symptoms characterizing SIS abnormal behaviours, for the purpose of empirical analysis. A recent systematic literature review carried out on the anomaly detection, analysis and prediction techniques in IoT environments [10], corroborates fairly well the novelty of the approach proposed in this work and the impacts it can have in the SIS community. Indeed, authors found gaps in the visualization of anomalies in IoT-based systems. They conclude that new methods and approaches are needed to represent intuitively IoT-based systems for analytical purposes. This is what the solution described in the sequel is all about.

5.3.2 Behavioural Drift Assessment Tool

Current software engineering approaches, which include formal testing and verification methodologies, claim for predictability. However, *"in practice, analytical modelling is increasingly proving inadequate, whenever it is agreed that one is not sure that something cannot be forgotten (the hypothesis of closing the model), that objective evidence is only evident in a given ideology (…), in other words, whenever one has to make the assumption that the phenomenon modelled is not complicated but complex"* [20] p. 19.

SIS, by their interaction with the open physical environment, are complex systems; At best, can we hope to get as close as possible to the desired behaviour

without reaching it exactly: *"[…] as soon as a system is open there is no optimum and any equilibrium is in interaction with its environment"* [31]. As a result, while the formal testing and verification methods still need to be conducted at design-time, they are not sufficient; the effectiveness of the SIS, i.e., the extent to which SIS behave as expected, must also be continuously evaluated at run-time.

Analytical modelling being inadequate, SIS effectiveness assessment has to rely on a systemic model of the physical effects they are legitimate to produce in the physical environment [30]. In the sequel, it is proposed to build this model in the framework of the Input/Output Hidden Markov Models (IOHMM) [4]. This framework is broadly used in behavioural modelling approaches [14, 40] where it has many advantages [37]; (1) it is an *explainable graphical model* [15] part of the Dynamic Bayesian Networks (DBN) family; (2) it formalizes *conditional dependencies* between the effects and their stimuli (i.e., *contextual input*, *events*); (3) it handles *tolerances on expectations* that may reflect randomness of some expected behaviours (through probability theory) or epistemic gaps in SIS knowledge (through the possibility theory and the Transferable Belief Model (TBM), an extension of the Dempster-Shafer evidence theory).

Formally, an IOHMM is defined by the tuple $< Q, \vec{\pi}, A, \vec{B} >$ where:

- $Q = \{x_1, x_2, \ldots, x_N\}$, $N \in \mathbb{N}$, is the finite set of hidden states where $x_{(k)}$ denotes the hidden state at time $k \in \mathbb{N}$,
- $\vec{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)^T$ is the initial state distribution vector where π_i denotes the likelihood of the state i to be the first state of a state sequence,[2]
- A is the $N \times N$ state transition matrix, where each element a_{ij} of the matrix is an n-dimensional input distribution ($1 \leq i, j \leq N$). Thus, $a_{ij}(\vec{u}) = p(x_{(k+1)} = j | x_{(k)} = i, \vec{u}_{(k)} = \vec{u})$ denotes the likelihood of transitioning to state $x_{(k+1)} = j$ at time $k+1$, given the current state $x_{(k)} = i$ and the contextual input vector $\vec{u}_{(k)} = \vec{u} \in \mathbb{R}^n$ at time k. The function p has here to be understood in the general framework of the uncertainty theory. While most of the hidden Markov-based models are defined in the probabilistic framework (i.e., p is a measure of probability), the model is also compatible with the possibility theory [28], the imprecise probabilities [11], etc.
- $\vec{B} = (b_1, b_2, \ldots, b_N)^T$ is the state emission vector, where each element b_i ($1 \leq i \leq N$) is an m-dimensional output distribution. $b_i(\vec{y}) = p(\vec{y}_{(k)} = \vec{y} | x_{(k)} = i)$ denotes the likelihood of observing the output vector $\vec{y}_{(k)} = \vec{y} \in \mathbb{R}^m$ at time k, i.e., the physical effects produced, while being in the state $x_{(k)} = i$. The output observation $\vec{y}_{(k)}$ at time k only depends on the state $x_{(k)}$ at time k.

2. In this work, we assume that the elements π_i of $\vec{\pi}$ are equally probable, i.e., equal to $\frac{1}{N}$.

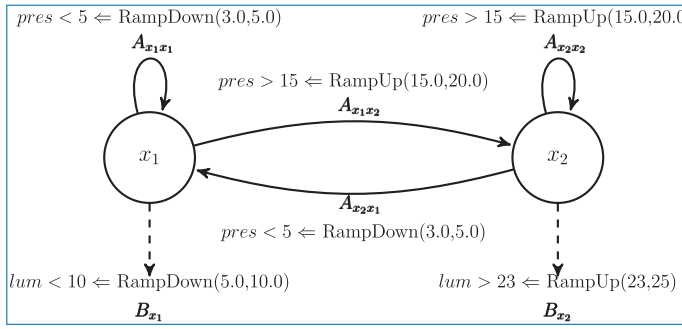


Figure 5.7. Possibilistic IOHMM model describing the legitimate effects that a luminosity control system must produce.

This model serves as a basis for efficient solutions to several inference problems [26]. Among these problems, the problem of inferring the likelihood of an observation sequence (i.e., $p((\vec{u}_{(k)}, \vec{y}_{(k)})_{k=1}^{K})$) to have been generated by the model (filtering), is particularly close to the one of assessing SIS effectiveness whose solution is given by the forward algorithm, here below in its (qualitative) possibilistic form where the function p is a measure of possibility denoted by Π:

1. **Initialization** – $\forall x \in Q$:

$$\alpha_{(1)}(x) = \min\left(\pi_x, \Pi(\vec{y}_{(1)}|x)\right) \tag{5.1}$$

2. **Induction** – $\forall x, x' \in Q, \forall 2 \leq k \leq K$:

$$\alpha_{(k)}(x) = \min\left(\max_{x' \in Q}\left(\min\left[\alpha_{(k-1)}(x'), \Pi(x|x', \vec{u}_{(k-1)})\right]\right), \Pi(\vec{y}_{(k)}|x)\right) \tag{5.2}$$

3. **Termination**

$$\Pi\left((\vec{u}_{(k)}, \vec{y}_{(k)})_{k=1}^{K}\right) = \max_{x \in Q}(\alpha_{(K)}(x)) \tag{5.3}$$

An example is depicted in Fig. 5.7. The model describes the legitimate effects that a luminosity control system must produce where tolerances on expectations are described through distributions of possibility as depicted in Fig. 5.8. Here, $\vec{u}$ corresponds to the value of a presence sensor, $\vec{y}$ corresponds to the value of a luminosity sensor.

5.3.3 Behavioural Drift Analysis Tool

On the basis of the IOHMM model and field observations, the effectiveness assessment allows the detection of behavioural drifts, consequence of unexpected

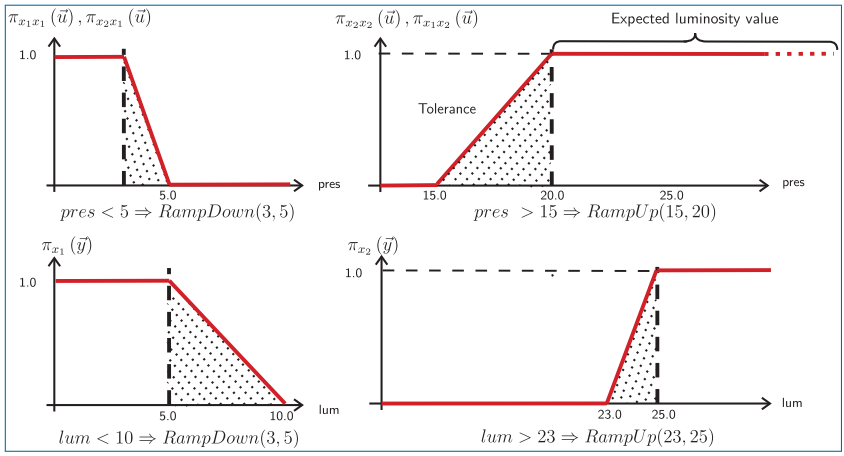


Figure 5.8. Distributions of possibility defining tolerances on expectations of the model depicted in Fig. 5.7.

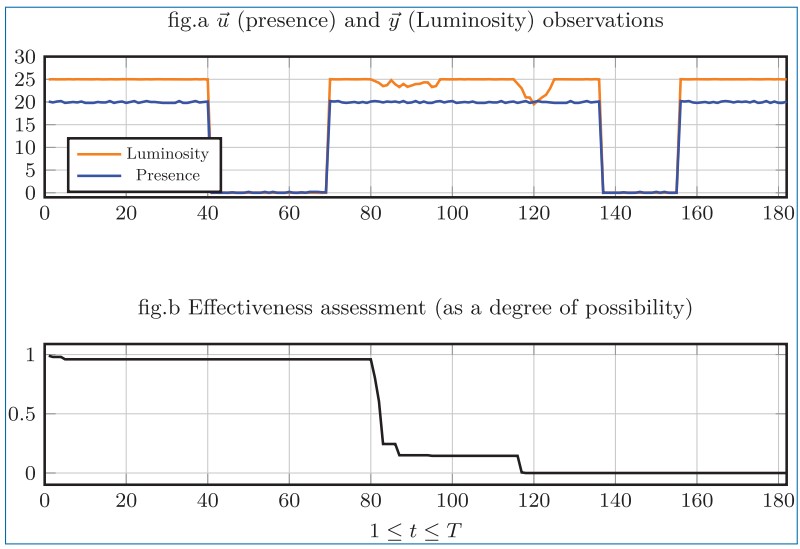


Figure 5.9. Effectiveness assessment results obtained on the model depicted in Fig. 5.7 from synthetic observations applied to the forward algorithm.

behaviours, whether legitimate or not. It complements the formal testing and verification approaches carried out at design-time. However, while the model underlying this evaluation is explainable, the same does not apply to the evaluation itself. Indeed, as a quantitative evaluation, it does not provide designers with information that would support them in understanding the reasons for its drifts, limiting de facto their ability to quickly take corrective actions and mitigate risks. The tool described in the sequel is meant to fill this gap.

This tool is mainly based on a generic clustering-based algorithm meant to learn IOHMM structure (states and state transitions) and parameters (distributions) from field observations. This algorithm consists in segmenting $\vec{u}$ and $\vec{y}$ observation spaces into a finite number of relevant regions such that each region represents respectively an input context and a discrete state, i.e., it is assumed that there exists a bijection between state and observation spaces [32], thereby taking advantage of understanding the structure of the IOHMM from the observation space [16].

Considering the observation sequence $(\vec{u}_{(k)}, \vec{y}_{(k)})_{k=1}^{K}$, the algorithm works as follows (detailed in [29]):

1. Get the set $\mathcal{Y}$ of clusters from $(\vec{y}_{(k)})_{k=1}^{K}$ such that each observation $\vec{y}_{(k)}$ is associated to a cluster $\mathcal{Y}_i$ (i.e., a state), $i \in |\mathcal{Y}|$
2. Get the set $\mathcal{U}$ of clusters from $(\vec{u}_{(k)})_{k=1}^{K}$ such that each observation $\vec{u}_{(k)}$ is associated to a cluster $\mathcal{U}_j$ (i.e., an input context), $j \in |\mathcal{U}|$
3. Get distribution parameters of B_i from $\{\vec{y}\}$ associated to $\mathcal{Y}_i$
4. Get the state-transition matrix A
   a. the sequence of clusters $(\mathcal{Y}_{(k)})_{k=1}^{K}$ obtained from $(\vec{y}_{(k)})_{k=1}^{K}$ defines the valid state-transitions $A_{ii'}(h)$, $i, i' \in |\mathcal{Y}|$, $h \in \mathcal{U}$
   b. $A_{ii'}(h)$ distribution parameters, $h \in \mathcal{U}$, are computed from $\{\vec{u}\} \in h$.

On the basis of this algorithm, a framework is proposed for investigating drifts in effectiveness of SIS. This framework conceptually takes place in two steps as depicted in Figure 5.10.

1. The learning algorithm is fed with observations corresponding to the effects expected to be produced by a SIS in different contexts, leading to the learning of a model of its legitimate behaviour.

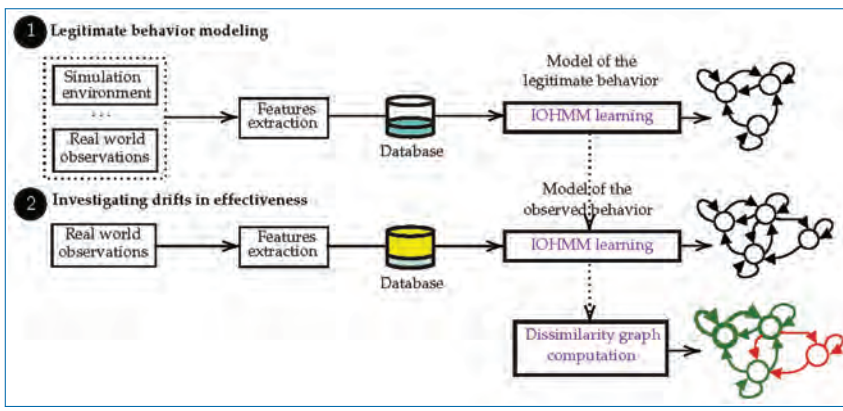


Figure 5.10. Steps to implement the proposed approach.

❷ This model is complemented with real-world observations, leading to the learning of a model of the observed behaviour. Then, on the basis of these models, an algorithm is developed to build a directed dissimilarity graph that makes clear the differences between both models, thereby helping designers direct their research and identify the possible causes of drifts in effectiveness.

Let us consider an observation sequence corresponding to the legitimate behaviour, defined by $(\vec{u}^{\star}_{(k)}, \vec{y}^{\star}_{(k)})_{k=1}^{K}$, followed by an observation sequence corresponding to real-world observations, $(\vec{u}_{(k)}, \vec{y}_{(k)})_{k=K+1}^{G}$. The algorithm works as follows (detailed in [29]):

1. Get the set $\mathfrak{Y}$ of clusters from $(\vec{y}_{(k)})_{k=1}^{G}$
2. Get the set $\mathfrak{U}$ of clusters from $(\vec{u}_{(k)})_{k=1}^{G}$
3. Get the set $\mathcal{Y}^{\star} \subset \mathfrak{Y}$ of clusters associated to $(\vec{y}^{\star}_{(k)})_{k=1}^{K}$
4. Get the set $\mathcal{U}^{\star} \subset \mathfrak{U}$ of clusters associated to $(\vec{u}^{\star}_{(k)})_{(k=1)}^{K}$
   ▷ $\mathcal{Y}_i \notin \mathcal{Y}^{\star}, i \in |\mathfrak{Y}|$ corresponds to unexpected states.
   ▷ $\mathcal{U}_i \notin \mathcal{U}^{\star}, i \in |\mathfrak{U}|$ corresponds to unexpected contextual input.
5. The sequence of clusters obtained from $(\vec{y}_{(k)})_{k=1}^{K}$ defines legitimate state-transitions $A^{\star}_{ii'}(h), i, i' \in |\mathcal{Y}^{\star}|, h \in \mathcal{U}^{\star}$. Also, the sequence of clusters obtained from $(\vec{y}_{(k)})_{k=K+1}^{G}$ defines observed state-transitions $A_{ii'}(h), i, i' \in |\mathfrak{Y}|, h \in \mathfrak{U}$.
   ▷ State-transitions $A_{ii'}(.)$ defined in the state-transition matrix A but not defined in the state-transition matrix $A^{\star}$ correspond to unexpected state-transitions.
   ▷ State-transitions $A_{ii'}(h)$ defined in the state-transition matrix A and in the state-transition matrix $A^{\star}$ with $h \notin \mathcal{U}^{\star}$ correspond to unexpected state-transitions (i.e., are triggered by unexpected contextual input).
6. Plot the dissimilarity graph where expected states (nodes) and state-transitions (edges) are coloured in green while unexpected ones are coloured in red as depicted in Fig. 5.10.

The dissimilarity graph, without identifying the causes of the drifts in effectiveness, sheds light on the symptoms likely to explain them.

5.4 Smart Home Use-Case and Illustration

The purpose of this section is to illustrate the toolset presented in this chapter and its benefits throughout DevOps life cycles. The illustration is carried out on a real

world smart-home scenario, providing the reader with technical details and key results.

5.4.1 Smart Home use Case Description

The experimentation is built upon the widely spread three-layer IoT architecture [17]. The first layer (*Perception Layer*) at the edge of the infrastructure consists of 57 IoT devices providing up to 319 parameters including 249 sensor/actuator whose values/states are updated on a regular basis on a centralized time series database (Synology DS418 Network Attached Storage (NAS) + InfluxDB [23]). IoT devices part of this infrastructure layer are mainly consumer electronics devices (e.g., Netatmo devices, Neato vacuum cleaner, LG TV, etc.), complemented with custom IoT devices based on Arduino Uno/Nano and Raspberry Pi equipped with sensors and actuators targeting specific purposes. The second layer (*Network layer*) enables data transmission and processing, throughout the different IoT wireless communication protocols (Wifi, ZWave [38], etc.). OpenHab [5] and MQTT middlewares provide this layer with the functionalities required for the illustration. Finally, the third layer (*Application layer*) consists in several software components controlling actuators and processing sensor data. These software components (along with middlewares) are encapsulated into docker containers hosted on computational resources (Raspberry Pi) at the edge of the infrastructure.

To illustrate SIS actuation conflict management (Paragraph 5.2) and behavioural assessment/drift analysis (Paragraph 5.3) toolsets, the following devices are used; a smart phone, a TV along with its remote control, a light bulb, an Amazon echo smart speaker and an autonomous smart vacuum cleaner. These devices are all located in the living-room and are implemented as part of a scenario focusing on inhabitants well-being. In this scenario, user-comfort is driven by luminosity and sound physical properties.

A first application set (*UserComfortApps*) is dedicated to give sensor data access to decision making algorithms (*App_Lum* and *App_RC_TV*) that respectively control (1) the luminosity level by acting on the roller shutters and the light bulb and (2) the sound by acting on the TV remote controller and the Amazon echo smart speaker. A second application set (i.e. *CommunicationCenterApps*) is deployed, creating a home-working environment where the focus is put on controlling sound sources so as to prevent home workers to be disturbed during phone calls or video conferences (e.g., *App_Phone_TV* mutes TV while a phone call is in progress). All applications are implemented through Node-RED flows.

To illustrate the toolsets proposed within the DevOps approach, a scenario with two DevOps cycles has been defined, each cycle covering the development and the operational stages.

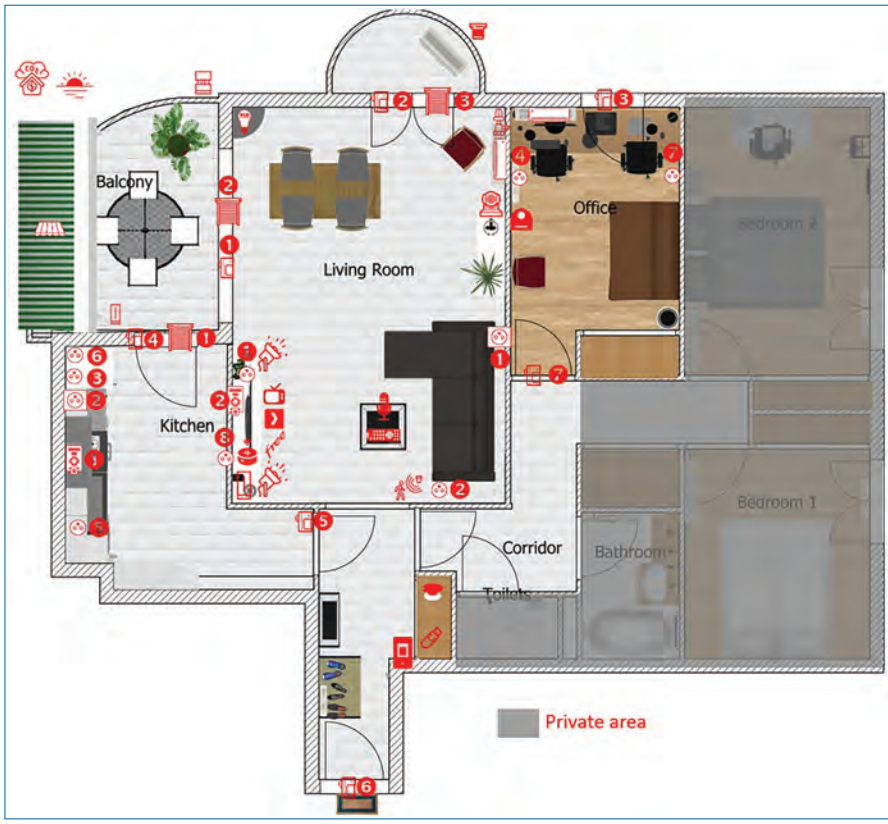


Figure 5.11. Smart Home use case setup.

5.4.2 Software Development (Devs, Cycle 1)

At this stage, the aforementioned software components are designed and developed (Fig. 5.12). Before deploying the whole system, the toolset developed for identifying and resolving actuation conflicts (Paragraph 5.2.2) is applied on the WIMAC model built from deployment, implementation and physical environment models. Designers specify the physical environment model by linking ActionComponents to the PhysicalProperty they act on. At this point, a potential direct actuation conflict is identified on *App_RC_TV* and *App_Phone_TV*; these applications being meant to control the TV sound source (Fig. 5.13).

The management of this direct actuation conflict is straightforward, applications merely send a Boolean value to a shared actuator (ActionComponent). The developer can then select, among the available off-the-shelf ACMs, the one relevant to resolve this generic conflict type (here, a simple OR logic ACM has to be instantiated between both software components (*App_Phone_TV* and *App_RC_TV*) and the actuator they act on (TV)). Once the selected ACM is instantiated into the

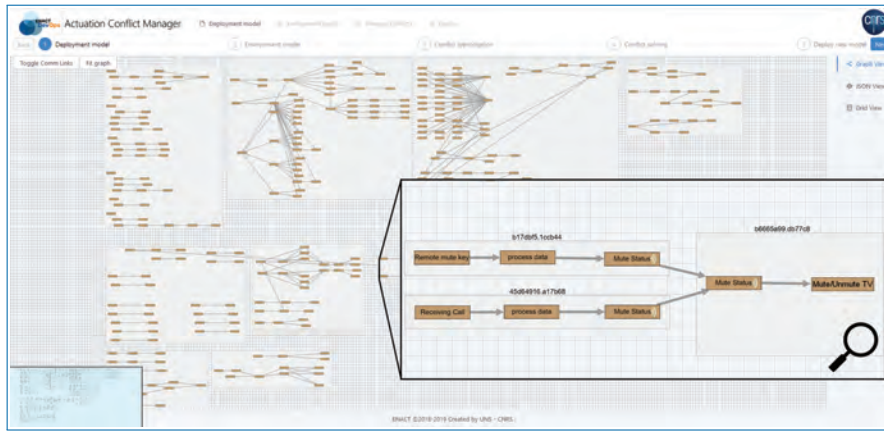


Figure 5.12. Excerpt of *App_Phone_TV* and *App_RC_TV* structural inter-relationships, as described in the WIMAC model, without environment model description.

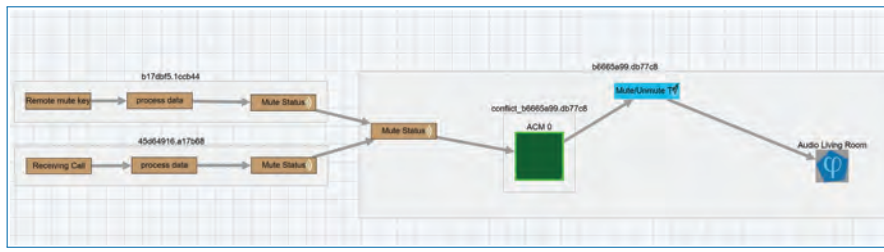


Figure 5.13. Excerpt of *App_Phone_TV* and *App_RC_TV* structural inter-relationships, as described in the WIMAC model with environment model description and the direct ACM instantiated.

WIMAC model, a new deployment model is generated from WIMAC and pushed to GeneSIS which concretizes the deployment.

5.4.3 System Operations (Ops, Cycle 1)

Throughout the execution of the SIS, the effects produced in the physical environment are observed and, thanks to the behavioural drift assessment toolset, the effectiveness of the SIS is measured from a model of the legitimate behaviour it has to comply with (Fig. 5.14).

The model of the legitimate behaviour is built upon a set of sound features computed from a microphone signal. Output observations (expected effects to be produced by the SIS) are then characterized by a Mel-Frequency Cepstral Coefficient[3]

3. https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

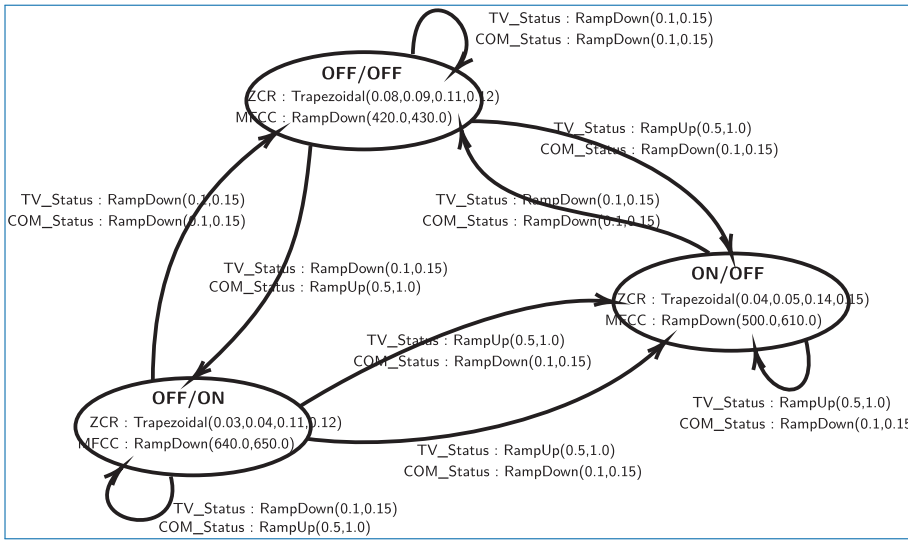


Figure 5.14. Possibilistic model of the legitimate behaviour. States are defined by the operating status of the TV and the communication status (i.e., OFF/ON means TV:ON, COM:OFF). The configuration ON/ON is not legitimate, the direct ACM instantiated is supposed to mute the TV while a communication is in progress.

(MFCCs) and the Zero Crossing Rate[4] (ZCR) sound features. These sound features lead a good segmentation of the observation space thereby, allow identifying device states on the basis of the sound they emit. Contextual inputs are characterized by the operating status of the TV and the communication status.

As depicted in Fig. 5.15, the ACM instantiated during the design phase fulfils its role and prevents the TV to operate while a communication is in progress. No behavioural drift is therefore reported.

While the SIS operates, free of unexpected effects produced in the physical environment, an autonomous smart vacuum cleaner moving around in the house, bursts into the living-room. By operating in the living-room, this device produces unexpected sounds leading behavioural drifts to occur (Fig. 5.16).

From the designers' point of view, the quantitative effectiveness assessment is not informative on the reasons for the drifts in effectiveness observed. At that point, the behavioural drift analysis tool (Paragraph 5.3.2 and 5.3.3) is leveraged to guide designers and help them correlate the symptoms of the drifts in effectiveness to various events (Fig. 5.17).

On the basis of the dissimilarity graph depicted in Fig. 5.17, designers can infer that drifts in effectiveness are the result of the smart vacuum cleaner unexpectedly

4. https://en.wikipedia.org/wiki/Zero-crossing_rate

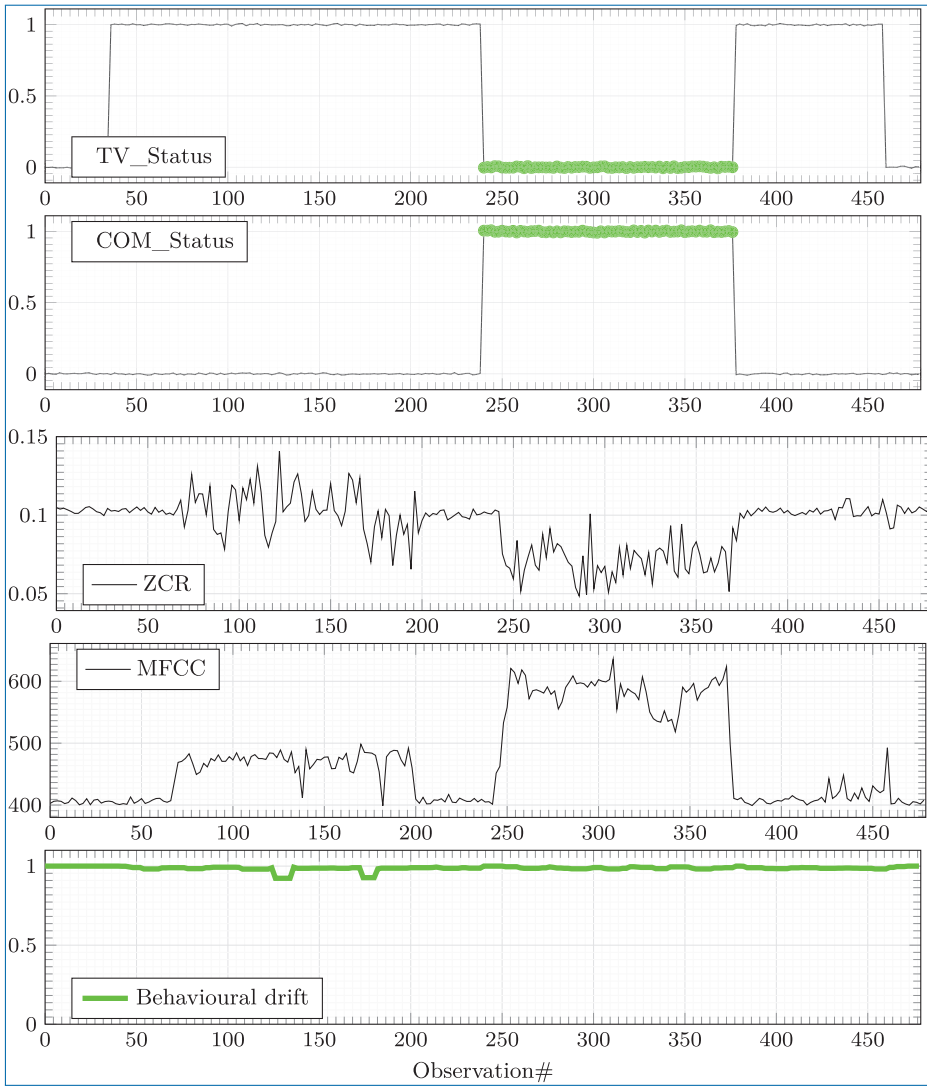


Figure 5.15. Observations from the field corresponding to the legitimate behaviour. The ACM instantiated during the development phase fulfils its role (green part of the graph) and no drift is reported here, the SIS behaves as expected.

(but legitimately) operating in the living-room and producing conflicting noise. This device was unforeseen by designers in the initial model of the legitimate behaviour. Moreover, this device has to be controlled so as to be integrated into the user-well being control strategy and prevent indirect conflicts to occur on the sound physical property.