Title:	*Survey and Requirements of Risk-driven Continuous Delivery of Trustworthy Smart IoT Systems*
Authors:	*Jacek Dominiak (CA), Nicolas Ferry (SINTEF), Alberto Huélamo (CA), Stéphane Lavirotte (CNRS), Phu Nguyen (SINTEF), Jean-Yves Tigli (CNRS)*
Editors:	*Stéphane Lavirotte (CNRS), Jean-Yves Tigli (CNRS)*
Reviewers:	*Adrian Irala Briones (INDRA), Oscar Zanutto (ISRAA)*
Identifier:	*Deliverable # D2.1 v1.4*
Nature:	*Report*
Date:	*31 October 2018*
Status:	*Delivered*
Diss. level:	*Public*

Executive Summary

This deliverable provides an overview of the state-of-the-art mechanisms for the risk-driven continuous delivery of trustworthy SIS. In addition, this deliverable characterizes the requirement, including trustworthiness requirements, to be considered.

The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement n° 780351 (ENACT).

Members of the ENACT consortium:

SINTEF AS	Norway
CA Technologies Development Spain S.A.	Spain
EVIDIAN SA	France
INDRA Sistemas SA	Spain
FundacionTecnalia Research & Innovation	Spain
TellU AS	Norway
Centre National de la Recherche Scientifique	France
Universitaet Duisburg-Essen	Germany
Istituto per Servizi di Ricovero e Assistenza agli Anziani	Italy
Baltic Open Solution Center	Latvia
Elektronikas un Datorzinatnu Instituts	Latvia

Revision history

Date	Version	Author	Comments
01 March	Initial		Outline and summaries
19 April	0.1	Stéphane Lavirotte, Jean-Yves Tigli	Outline and table of content
25 May	0.2	Stéphane Lavirotte, Jean-Yves Tigli	First draft of the introduction
10 July	0.3	Nicolas Ferry, Phu Nguyen	Integration of SMS in Section 3
28 August		Alberto Huélamo	Integration of Section 2 and Section 5
30 August	0.4	Stéphane Lavirotte, Jean-Yves Tigli	Integration of SMS of Section 4
19 September	0.5	Nicolas Ferry, Phu Nguyen, Stéphane Lavirotte, Jean-Yves Tigli	Complete Section 3 with SLR Complete Section 4 with requirement and conceptual design Contribute to introduction
24 September	0.6	Stéphane Lavirotte	Review the whole document and add comments to all sections
28 September	0.7	All authors	Complete each section
02 October	0.8	Nicolas Ferry	Review of the whole document and add comments to authors
04 October	1.0	Stéphane Lavirotte, Jacek Dominiak, Alberto Huélamo	Contributions to conclusion and transitions between sections Modifications in Sections 2 and 5
05 October	1.1	Stéphane Lavirotte, Jean-Yves Tigli, Nicolas Ferry	Revision for internal review
18 October	1.2	Adrian Irala Briones, Hui Song, Oscar Zanutto	Comments and modifications added by reviewers
24 October	1.3	Jacek Dominiak, Nicolas Ferry, Alberto Huélamo, Stéphane Lavirotte, Phu Nguyen, Jean-Yves Tigli	Corrections after internal review
31 October	1.4	Stéphane Lavirotte, Jean-Yves Tigli	Final release of the deliverable

Contents

1 Introduction

1.1 Context and objectives

By 2020, Gartner envisions that 21 billion Internet-of-Things (IoT) endpoints will be in use[1], representing great business opportunities. Despite having enormous potential, the heterogeneous nature of IoT bring up great challenges that must be addressed to realize the full potential of the IoT. In particular, it is important to facilitate the creation and operation of the next generation IoT systems that we denote as *Smart IoT Systems* (SIS) and which need to perform distributed processing and coordinated behaviour across IoT, edge and cloud infrastructures, manage the closed loop from sensing to actuation, and cope with vast heterogeneity, scalability and dynamicity of IoT systems and their environments.

The trustworthiness of such systems that may, for instance, involve actuators and deal with sensitive data, is critical[2], ranging from business critical to safety critical. In addition, SIS typically operate in a changing and often unpredictable environment that is also challenging their trustworthiness. For example, to ensure that they always work within safe operational boundaries[3] (*e.g.*, controlling the impact that actuators have on the physical world) and to manage conflicting actuation requests. The ability to continuously evolve and adapt these systems is thus decisive to ensure and increase their trustworthiness, quality and user experience.

The DevOps movement advocates a set of software engineering best practices and tools, to ensure Quality of Service whilst continuously evolving complex systems and foster agility, rapid innovation cycles, and ease of use. DevOps ideas promote a tight collaboration between the developers (Dev) and the teams that deploy and operate the software systems (Ops). DevOps seeks to decrease the gap between a product design and its operation by introducing software design and development practices and approaches to the operation domain and vice versa. It aims to ensure a rapid and efficient value delivery to market. Therefore, DevOps has been widely adopted in the software industry.

However, there is no complete DevOps support for SIS today. Current DevOps solutions lack mechanisms for continuous quality assurance[1,2], for example, mechanisms to ensure end-to-end security and privacy and mechanisms able to take into consideration open context and actuation conflicts (*e.g.*, allowing continuous testing of IoT systems within emulated and simulated infrastructures). Challenges also remain to perform continuous deployment and evolution of IoT systems across, IoT, edge, and cloud spaces[4]. These are all key features that are necessary to provide DevOps for trustworthy SIS.

The objective of ENACT is to address this issue and to enable DevOps in the realm of trustworthy SIS. More specifically, WP2 will deliver a set of tools that aim at improving the **management and continuous delivery of trustworthy SIS** (see blue part of Figure 1) agile and continuous evolution and (ii) ensuring the proper design of the system before delivery. This includes the following tools: (i) agile and continuous evolution and (ii) ensuring the proper design of the system before delivery. This includes the following tools:

- Risk Management: risk management handled in an agile and continuous and supporting trustworthiness for SIS.
- GeneSIS: a framework to support the continuous orchestration and deployment of IoT systems over IoT, edge and cloud infrastructures, which assimilate ENACT trustworthiness mechanisms.
- Actuation conflict management: sharing resources (devices and physical environment) may create conflicts one needs to identify, analyse and resolve so as to ensure SIS trustworthiness,

[1] http://www.gartner.com/newsroom/id/3598917
[2] https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot
[3] "Software Continuum: NESSI Recommendations for ICT WP18-19", May 2016. http://www.nessi-europe.eu/Files/Private/NESSI_Input_for_WP18_RELEASE2_to_EC.PDF: *"Techniques that ensure that such software will always work within safe operational boundaries (e.g., in safety critical situations, involving the IoT/CPS) are required."*
[4] "Software Continuum: NESSI Recommendations for ICT WP18-19", May 2016. http://www.nessi-europe.eu/Files/Private/NESSI_Input_for_WP18_RELEASE2_to_EC.PDF: *"In addition, new services for dynamic deployment and provisioning in the fog and edge networks (e.g., trusted support of cloud bursting) are required, as are new development paradigms and tools (e.g., DevOps enablement down to the remote devices)."*

– Test, Simulation and Emulation service: two modules to simulate and generate events in order to test SIS in replicable environments.

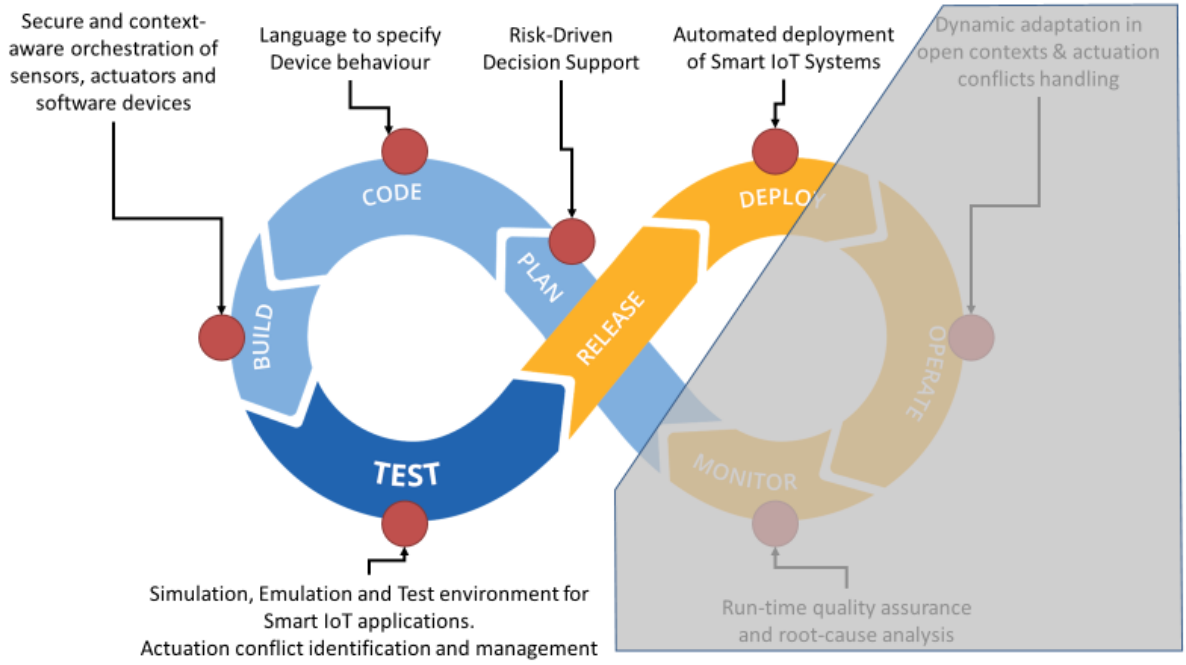

Figure 1: Focus on Dev part of the DevOps cycle

All these tools will not only facilitate the agile creation and management of SIS but will also help enforcing their trustworthiness. Based on the NIST's definition of trustworthiness for Cyber Physical Systems [1], within ENACT, we adopt the following definition of trustworthiness:

"***Trustworthiness*** *refers to the preservation of security, privacy, safety, reliability, and resilience of SIS"*.

We adopt the following definitions of the different properties that make a SIS trustworthy:

- **Security** refers to the preservation of confidentiality, integrity and availability of information [2].
  - **Integrity** is the property of protecting the accuracy and completeness of information [3].
  - **Confidentiality** is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [3].
  - **Availability** is the property of information being accessible and usable upon demand by an authorized entity [3].
- **Privacy** refers to the protection of personally identifiable information (PII) [4]. 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 [1].
- **Reliability** refers to the ability of the CPS to deliver stable and predictable performance in expected conditions [1].
- **Resilience** refers to the ability of the CPS to withstand instability, unexpected conditions, and gracefully return to predictable, but possibly degraded, performance [1].

Deliverable D2.1 focuses on the state-of-the-art analysis for each of the enablers (risk management, orchestration and deployment, actuation conflict management, test and simulation) and derive tools requirements.

Finally, it is worth noting that (i) D2.1 complement Deliverable D3.1, which focuses on tools and mechanisms for operational part of the DevOps process and (ii) the relationship between the different enablers will be further described in deliverables D5.1 and D5.2 on the ENACT architecture.

1.2 Achievements

Objectives	Achievements
State-of-the-art on risk-driven continuous delivery of SIS	We conducted an extensive analysis of the state-of-the-art on approaches for the continuous delivery of SIS. Both industrial and academic approaches were considered. We specifically focused on four topics: 1. Agile continuous risk management 2. Continuous orchestration and deployment of SIS 3. Management of actuation conflicts during development 4. Test and simulation of SIS In particular, two systematic mapping studies were conducted for topics 2 and 3, and a systematic literature review has been done for topic 3.
Requirement elicitation	From the analysis of the state-of-the-art, we derived a set of technical requirements for each enabler. These requirements complement the requirements defined in D1.1 by the use case providers. These requirements will drive the design of the enablers.
Conceptual design	We provided an initial and conceptual design of each of the enablers that will serve as baseline and plan for the first version of the enablers. It is also made clear (i) how each enabler contributes at addressing the challenges identified in the state of the art, (ii) what will serve as baseline technology, and (iii) how each enabler contributes at improving the trustworthiness of SIS.

1.3 Structure of the document

The remainder of the document is structured as follows. We decided to follow the development cycle of the DevOps approach as a thread for this document: plan, code, build and test. Each section details the addressed problematics at each stage.

In Section 2, the agile continuous risk management of SIS is presented. The state-of-the-art (Subsection 2.1) emphasises the risk management in the software development cycle (2.1.1) and the agile continuous risk management (2.1.2). As a result, the challenges (Subsection 2.2) and the requirements (Subsection 2.3.1) are addressed through use cases.

In section 3, the state-of the-art of continuous orchestration and deployment of SIS is provided through a systematic mapping study (Subsection 3.1) and a systematic literature review (Subsection 3.2) of the current approaches. Subsection 3.4 analysis and discusses the state-of-the-art before describing the proposed framework for the deployment and orchestration (Subsection 3.5).

Section 4 covers the identification, analysis and management of actuation conflicts. A state-of-the-art is provided through a systematic mapping study (Subsection 4.1) which is discussed (Subsection 4.2) to define the requirements (Subsection 4.3) and the conceptual design of a framework for actuation conflict management (Subsection 4.4).

The section 5 presents the test, simulation and emulation services for SIS. Subsection 5.1 presents the state-of-the-art and challenges in IoT simulation and testing. As a result, the addressed use cases are presented in Subsection 5.2 and the requirements in Subsection 5.2.1, before describing the proposed conceptual design for testing and simulation (Subsection 5.3).

All the sections described above include a Subsection addressing the contribution to trustworthiness for each proposed enabler.

The conclusion of this deliverable and the next steps are presented in Section 6.

2 Agile and Continuous Risk Management

The first stage of the development cycle of the DevOps approach is about planification. As part of the planification, risk management must be handled in an agile and continuous way.

Selection of correct services is seen as one of the cornerstones of a service-oriented market for technology. In its earliest days this was based on Service Oriented Application, later as Cloud Service selection. Cloud services and services provided by devices connecting to the Internet through Internet of Things are not completely transparent and therefore there is a greater risk to employing cloud services and IoT services in an application. There is a growing body of work investigating and developing solutions and mitigations for the risks in using these services. This section examines the current state of the art of an agile development supporting continuous risk management. It also examines the relevant IoT specific risk and vulnerability management.

2.1 State of the art of the Agile Continuous Risk Management

In this section we explore what it means to analyse the risks in a continues fashion. Section 2.1.1. talks about the history of risk management in a software development world. Section 2.1.2 tries to explore the progression of the software development world to agile and what it means to the risk management world. How the concept of Agile Continuous Risk Management is not a choice but a necessity in this context. Section 2.1.3 explores CA previous research of the topic, from the flat but business-oriented risk management, though security-oriented risk management focused on the cloud environments into the agile, adaptive continues risk management for the IoT we are trying to explore in ENACT.
Section 2.1.4 showcases how the trust and its building factors can be influenced by the risk management approach we are proposing.

2.1.1 Risk Management in the Software Development Cycle

Implementing risk management processes for complex, high-risk projects is essential [5]. A well-known drawback in the application of risk management techniques for software projects managed through waterfall approaches is that requirements tend to change during the project life-time and risk analysis may become obsolete [6]. Waterfall models assume requirements to be clearly defined in advance in the design stage and, in general, to remain fairly immutable, which tends to be unrealistic for industry projects.

Metrics to monitor the level of risks can be quantitative, such as the probability of occurrence or the effort to implement control measures, or even its cost. But they can also be qualitative, *e.g.*, an appraisal of project staff's motivation. There are many quantitative risk methodologies and tools, like RiskWatch[5] or ISRAM [7] and there are many qualitative risk methodologies such as OCTAVE [8], Coras [9] or one of the most commonly known in hosted software arena, STRIDE[6]. A prime example of an open standard trying to define the risk aware software development is OWASP[7] that aims at making software security visible, so that individuals and organizations are able to make informed decisions. DREAD [10] is a successor of STRIDE and provides another approach specialized in multi-stack type of applications. Most risk management tools and methodologies focus on the assessment of risks at a fixed stage in time. **However, current risk management tools and methodologies do not address one of the most prominent challenges in today's adaptive, pivot-oriented world: continuous software evolution and, as a consequence, risk evolution.**

2.1.2 Agile Continuous Risk Management

The first wave of agile adoption in many companies involves structuring software development through a large number of Scrum teams. In general, companies find that aligning these teams against larger

[5] https://www.riskwatch.com
[6] https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx
[7] https://www.owasp.org/index.php/Main_Page

programs is difficult. Specifically, when it comes to application risk control, most companies do not have enough people with sufficient risk management expertise to appoint somebody in each team. In order to partially mitigate this issue, they instead choose to have staff responsible for detecting the most important risks in a centralized way.

The use of framework methodologies to scale agile, such as SAFe, improves this situation only marginally. In this case, organizations structure software development creating teams of teams.

As an example, SAFe delivers solutions through so-called Agile Release Trains[8], which is a self-organizing team of agile teams (typically from 5 to 12 teams including from 50 to 125 individuals in total) that plan, commit and execute together. ART teams plan and synchronize their work through Program Increments (here therefore referred to as PI)[9] and organize the work using a Program Backlog[10].

When planning a new PI (typically every 10 weeks), all the actors participating in the software development process attend the meeting, usually in person. PI planning usually starts with a vision and a prioritized list of features of the product and the group comes up with the team and program PI objectives and the program board (description of feature deliveries, dependencies and milestones). During PI planning, features are decomposed into user stories. This is defined by product owners and created by the different agile teams, which are in charge of drafting plans but also of analysing risks and impediments. In fact, user stories replace traditional functional requirements, and describe intended system behaviour, and risk analysis is done related to expected features and to these user stories. Because of this, non-functional requirements which are usually not represented through these user stories, tend to be unintentionally diminished in terms of importance [11] and commonly ignored during the risk analysis. Based on challenges, risks and impediments, adjustments are made to the plan.

Distributed agile development (DAD) approach is increasingly adopted by more and more software companies. The idea of DAD is to combine the quality and speed benefits of agile with the cost benefits of distributed software development (DSD). However, this combination generates significant risks, considering the contradicting nature of agile and DSD. Work in this area studies risks related to software development life cycle, project management, group awareness, etc. However, this work is focused on analysing risk factors that might represent a threat to the successful completion of a software development project, rather than risk threatening the fulfilment of non-functional requirements.

2.1.3 Previous work

Although the topic of effective Risk Management has been worked on during the last decade, the ever-changing landscape of IT means that we still lack a final solution. The absence of standards and interoperability between solutions providers often makes a selection binding, which imposes new set of risks. Several methods for cloud service comparison are developed. *E.g.*, the Service Measurement Index (SMI)[11]. These approaches only consider multi-cloud environments where a single application may use cloud services from multiple cloud service providers at the same time. MODAClouds FP7 project is one of the first attempts to include new metrics for cloud service evaluation and selection in multi-cloud environments and it provides the mechanisms for risk driven decision support. MUSA H2020 project extends the results of MODAClouds focusing in particular on evolving metrics related to security aspects and addressing the agile risk assessment. However, there is a considerable gap adopting these techniques and/or devising new ones to address the challenges of risk management in the IoT and edge space.

2.1.4 IoT Risk Management and Trust

There have been several IoT specific investigations into Risk analysis that discuss vulnerabilities in this domain. Although it is an enabler for IoT, RFID demonstrates some specific vulnerabilities that potentially impact IoT implementations. Most of the privacy and security issues come from the air

[8] http://www.scaledagileframework.com/agile-release-train
[9] http://www.scaledagileframework.com/program-increment
[10] http://www.scaledagileframework.com/program-and-value-stream-backlogs
[11] https://en.wikipedia.org/wiki/Service_Measurement_Index

interface between a tag and its reader [12]. This becomes an issue when RFID identification is blocked or intercepted at that interface, although this paper mentions the risk of vulnerabilities without offering any mitigation strategies. In contrast to this there are some investigations that provide tools and methodologies that can detect and analyse risks in terms of IoT specific vulnerabilities. If we look specifically at the consumer internet [13], there are some tools, for example Sohdan, to discover IoT devices and Nessus to evaluate vulnerabilities, that are examined and may influence risk management in the wider IoT environment. A number of papers recognise the lack of a framework for risk management in IoT. By creating vulnerability scores and evaluating IoT configurations and attacker capability [14] it may be possible to compute attack likelihood and cost.

Estimating risk is at the heart of some access control methods of securing IoT. In a review of different techniques for estimating risks [15] each potential technique is analysed, but there seems to be little attempt at a comparison of effectiveness. Likewise, dynamic security risk assessment looks at the sense and application layers of an IoT implementation but seems to neglect the networking layer [16]. There is however an interesting proposal that immunology set theory could produce an accurate risk assessment. Security risk assessment and access control is also discussed [17] with a view to protecting IoT installations by creating a policy language that restricts access to only services that are needed and uses a "default-off" approach for protection of the wider network. A wider report [18] on IoT risk analysis took the approach that an analysis of different domains and their issues can give a number of common IoT related risks that introduces UAV/Drones as a category of device. This report also reviews OWASP and the European Research Cluster on the internet of things. It also widens the risk profile to critical national infrastructure in delivering a comprehensive view of common risks. This report and the previous paper have the advantage that they were produced in September 2018 and as such are current with this document.

Looking at the practical implications of risks, as mentioned in the previous report, ENISA has produced a report on the IoT/RFID scenario risk assessment[12]. This is based on the Air travel industry but nonetheless indicates that many risks can be mitigated or even avoided if the way in which the technology is being used is carefully managed. In particular, new business structures and models are considered worthy of consideration. Trust and Risk are often inter-related in discussions, however a literature review on trust management [19] places trust and trustworthiness at a higher premium than risk management because it has a perceptual impact on human reactions. Although trustworthiness is at the centre of ENACT this paper is more a list than a critical review.

In conclusion the combination of risk evaluation techniques and tools allied with a trust -based approach to managing the perception of risk should enable ENACT users to develop trustworthy IoT applications.

2.2 Challenges related to Agile Continuous Risk Management

Devising intelligent recommendation systems to mitigate the lack of expertise of different stakeholders: lack of expertise is an important issue for agile self-managed teams, but also for small or medium enterprises that cannot afford risk experts in smaller development teams. Detecting the most prominent risks and deciding for the best mitigation actions may be a difficult task. Deciding the level of likelihood and impact of a particular risk may also be very subjective and, therefore, difficult to assess and measure. While training is an essential aspect [11], lack of expertise may always be present in large and distributed teams and needs to be mitigated. As the use of artificial intelligence techniques becomes commonplace, an interesting research area emerges to create recommendation systems to help expert and unexperienced employees to define risks and the necessary mitigation actions. Besides, the fact that many tools to control software development are made available to customers through SaaS offerings enables the use of novel crowdsourcing techniques to use the learnings extracted from the activity of past users in the platform.

Enabling continuous risk management for highly dynamic systems that need to be continuously re-factored: There is a need for risk analysis methodologies that are adapted to agile contexts but still achieve the level of analysis and detail provided by traditional risk assessment and mitigation

[12] ENISA. 2018. FAQs to IoT/RFID Scenario Risk Assessment. (April 2018), 1–5.

techniques, in particular related to NFRs. Fitzgerald et al [20] illustrate how Lean Thinking [21] can be applied to continuous software engineering. Authors even go beyond software development and consider issues such as continuous use, continuous trust, etc., coining the term "Continuous" (Continuous Star). However, they do not explicitly tackle challenges related to continuous risk management.

Devising collaboration mechanisms and tools to enable collaborative risk management: One of the five principles for Lean Startups as defined by Ries [22] states that entrepreneurs are everywhere. Ries recognizes that not only start-ups but also large enterprises work under conditions of extreme uncertainty. New ideas and experience may emerge from anyone in a large organization. Therefore, facilitating participation and collaboration becomes essential. In order to improve the control of risk in dynamic agile software development processes, tools that allow better transparency and enable collaboration of all stakeholders involved in the process are crucial. In this sense, creating new tools that can be easily embedded with other common agile tools to manage software development is very important. Common generic tools such as Kanban are increasingly used by software industry for scheduling work, representing user stories, features, etc., or to extend well-established Scrum methods. These tools need to be extended to integrate agile risk control methodologies. The detection of new risks or planning mitigation actions should immediately propagate information and even trigger warning or actions through these software development management tools. In this sense, research on multi-Kanban systems for agile risk management integration is crucial. Besides, the DevOps concept emerged as an attempt to express the need for collaboration between development and operations that arise within software companies. Collaborative risk management should also involve main actors in operations.

Embracing cultural change: Cultural change of any type is difficult and time consuming to achieve. The research challenge here is to develop a cultural change methodology that will place risk management as a central and critical part of an agile methodology. Risk Management is often neglected as part of a backlog or sprint plan which leads to a view that it is not important until the later stages of a development project. Cultural change of this type is often implemented by management mandating making risk part of every sprint. This has the potential to lead to various levels of adoption by the development staff. CA has found in the past that significant changes to development methodologies such as a common development environment are adopted by three distinct groups of people, the innovators, the majority and the laggards. These groups are based on the groups defined as part of the diffusion of innovation theory [23] and adapted for use by CA internally by merging the early/late majority groups into one. The goal of a cultural change strategy is to increase the numbers of members of the early adopters and majority to enable a focus on managing adoption by the laggard groups. More research is needed to develop a more general methodology for adoption of risk management at the early stages of a development life cycle.

2.3 Addressing use cases

Within this section, we will explore the use requirements expressed by the ENACT use cases in respect to the Risk Management Enabler. Although all use cases agree that handing risks throughout the software development process in a necessary step to consider, not all use cases expressed the need to handle it by the means ENACT proposes.

Another part of the chapter addresses the trustworthiness factors which we will be addressed though ENACT Risk Management.

2.3.1 Requirements

The requirements in this chapter are based on the use cases expression of needs described in D1.1 as well as collected though set of interviews with use cases providers. These interviews were set in order to perform in depth analysis of the requirements taking into consideration CA's previous experiences with Risk Management.

There are three main actors identified as a recipient of Risks Management Enabler, these are:

- IoT Application Developer.

- Risk Manager – this can be a person which oversees the application development process, manages the risks directly or indirectly. In the context of Agile and especially SAFe technology this role would involve Release Train Engineer and/or Product Owner.
- Solution Architect.

Requirements which are foreseen to be fulfilled by this tool are listed in Table 1 where the abbreviations of the use cases are defined as: Intelligent Transport System (UC1), Digital Health (UC2), and Smart Building (UC3). Risk Mitigation Enabler will be primarily exploited by UC2 and UC3, however UC1 agreed to evaluate the tool with the risk managers within their company.

Table 1: WP2 requirements for Risks Management

ReqID	Requirement	Description
UC2 **R1**	Risks Overview	The tool is ought to provide means to express and analyse all types of risks. This means functional as well as non-functional types of risks.
UC2 **R2**	Risks Status	The tool is ought to provide means to check the status of the risks and their mitigation at any given time, that being development or operation time. Within the DevOps cycle, risks analysis is believed to be continued so the importance of understanding the status of risks becomes critical.
UC3 **R3**	Active cross actor collaboration	The tool is ought to enable communication and collaboration between the actors of risk management process in order to foster better understanding of risks and the steps required in order to fulfil the mitigation strategy.
UC2 **R4**	Risks prioritization	The tool is ought to provide evaluation means which would enable the actors to understand the impact of mitigation actions on the current development plan and accommodate it within the software development process
UC2 **R5**	Mitigation impact on operations	The tool is ought to provide the set of technically oriented mitigation actions in order to assess if a change of the architecture might be necessary in order to mitigate the risks. This is especially true in IoT, hybrid highly dynamic environments where Enact is planning to make the most impact.
UC3 **R6**	Personalized, architecture crafted mitigation actions	The tool is ought to provide suggestions on the mitigation actions which take into consideration the type of the architecture against the risks analysed as well as types of risks which might occur within whole or subset of the IoT architecture.
UC3 **R7**	"Just enough" level of risks setting	Due to the cost of actions which are necessary in order to mitigate the risks, the tool is ought to provide the necessary means to evaluate what is the "just enough" level of mitigation where the balance of likelihood vs impact is met at any given scenario.
UC2 **R8**	Risks reintroduction	The tool is ought to provide means for the detecting if previously analysed risk has been reintroduced during the evolution of the IoT orient software.
UC2 **R9**	Release schedule impact	The tool is ought to provide means to evaluate the impact of the mitigation actions against the current release planning to that actors can detect potential clashes of mitigation actions vs planning.
UC3 **R10**	Architecture weak points detection	The tool is ought to provide means to evaluate the architectural robustness of the IoT application by automatically matching risks to the architecture described. This would enable the possibility of enhancing the architecture during the design time without exposing the application to potential risks which can be addressed otherwise and become cured.

2.3.2 Towards ENACT Risk Management

Risk Management enabler in ENACT will build on top of the Risk Management tool developed during the project MUSA. The baseline of MUSA Risk Assessment tool is a Kanban-styled board for multi-disciplinary and continuous risk management. The tool eases up the tedious tasks of identifying and evaluating the threats over your cloud-based application.

In order to address the IoT specific risks, it is planned to enhance the tool with the capability to express the risks not only against the application modules, which in the software development paradigm are the

typical building blocks of the application, but also with possibility to express the risks against the physical components of the application. In the IoT space, applications are often highly distributed and carry logic throughout the variety of communication protocols. These types of configurations require extensive risk analysis due to the fact that the software and physical risks might collide and influence each other. Another IoT specific paradigm that requires to go through the risk analysis process is the imposed risk of the device's configuration or even devices vendors.

Taking all of that into consideration Risk Management of ENACT will extend the definition of what an application component is and will allow to express the risks against not only software-based application building blocks but also the hardware ones. This in effect might have a direct impact on the choices which the application developers will make in order to make the application more trustworthiness overall.

Another extension on top of the previously developed Risk Management solution is the fact that the tool will have the capability of discovering the application architecture based on the common model carried out throughout the ENACT framework. This in effect will allow the tool to propose set of risks based on the application overall architecture or its parts prior to the user starting the risk analysis. It is even planned, that with enough learning data the tool should be capable of proposing the likelihood, impact and even the mitigation actions for each of the application components within the architecture without any user actions. This would then serve as a basis to evaluate and adapt for the actors involved in the process rather than building the risks analysis from scratch.

Risk Management in ENACT should be a self-contained application for a complete risk management cycle which can collect and evaluate the business and non-functional risks, functional risks and complete the process of mitigation of identified risks by setting the actionable tasks which would ensure that the risk is at the "acceptable" level.

2.3.3 Addressing trustworthiness though Risk Management

Risk Management Enabler in ENACT will build on top of the Risk Assessment tool introduced in MUSA project where the tool was oriented purely in the cloud security risks. Building on top of existing work, we are planning to address multiple factors which build the trustworthiness.
These are:

- Security – by matching the risks to the existing architectural scenarios we are enabling the users to detect, recognize and action upon the security risks which they might not be aware of. Security in terms of the tool is ought to be based on the OWASP IoT project with its current state of the art analysis from 2018 which are soon to be released.
- Privacy – by trying to match the risks to the GDPR clauses automatically in order to ensure that the design of the system is meeting current legally imposed requirements.
- Reliability & Resiliency – by mapping the characteristics of the nodes of the application to the risks learned from the past analysis the system will be able to detect risks and potentially match less risky application components alternatives with similar parameters.

All above trustworthiness factors take into consideration the application description at design time, with consideration that the same application might be reanalysed in the future as the application evolution progresses.

3 Continuous orchestration and deployment of SIS

After the planification considering the risk management and the development, the implementation of the software components of the whole system must be carried out. A framework shall help developers to deploy and orchestrate the involved software components by addressing the specificities of SIS.

Software deployment typically refers to a post-production activity performed once a piece of software has been developed. This activity consists in making a software available for use [24]. In this study, approaches are considered as supporting deployment when they explicitly offer mechanisms enabling the software deployment process, *which typically* consists of the following stages: (i) release, (ii) installation, (iii) activation, (iv) update, (v) adaptation, and (vi) un-deployment [25].

Deployment approaches typically rely on the concept of software artefacts or components as a modular part of a system that encapsulates its contents [25]. A deployment configuration is thus a connected graph that describes deployment artefacts along with targets and relationships between them from a structural perspective. In the cloud context, the term "topology" is often used as well. A deployment configuration or topology typically includes the description of how its software components are integrated and communicate with each other. This is often referred to as **software composition or orchestration**.

"Continuous deployment is the practice of continuously deploying good software builds automatically to some environment, but no*t necessarily to actual users"* [26].

Because (i) tools supporting the continuous deployment of software systems are typically at the border between development and operation activities (and thus a core element of typically DevOps tooling support) and (ii) there are many industrial and research activities focusing on the development of such tools, we decided to conduct an extensive and systematic literature review of orchestration and deployment approaches for IoT.

The remainder of this section is as follows. First, we present the research landscape of the existing approaches for supporting the orchestration and deployment of IoT systems in Subsection 3.1. Then, we dive into analysing the most complete and significant deployment and orchestration approaches for IoT in Subsection 3.2. We also discuss in Subsection 3.3 the other related approaches for deployment and orchestration of IoT. Based on our extensive literature review presented in Subsections 3.1, 3.2, and 3.3, we analyse the gaps in the current state of the art in Subsection 3.4. Then, Subsection 3.5 presents the specification of our approach and toolchain for supporting continuous orchestration and deployment of trustworthy SIS that can advance the current state of the art. Finally, Subsection 3.5.2 provides the baseline technologies and plan for developing our approaches and tools.

3.1 A systematic mapping study of orchestration and deployment approaches for IoT

Research community and industry have been proposing different approaches for supporting the deployment and orchestration of IoT systems focus on different cloud and edge infrastructures. However, it is not clear what are the existing primary approaches and tools for supporting the deployment AND/OR orchestration of IoT systems (**Depo4IoT** for short), and how advanced they are.

To provide a clear picture of the research landscape of the existing Depo4IoT approaches and tools for supporting the orchestration AND/OR deployment of IoT systems, we have conducted a systematic mapping study (SMS). Section 3.1.1 gives an overview of our SMS such as our research questions and review method. We provide some highlights of the results of our study in Section 3.1.2. All the details of our SMS are in the Appendix A.

3.1.1 The protocol of our SMS

We conducted our SMS by following the latest guidelines for conducting SMS in [27] and other relevant guidelines in [28, 29]. Based on the research questions in Subsection 3.1.1.1, we conducted a rigorous search and selection process presented in Subsection 3.1.1.2. that give us a set of primary studies for data analysis. Subsection 3.1.1.3 presents the taxonomy that we used to extract data and analyse the primary studies.

3.1.1.1 The research questions of the SMS

RQ1: What are the publication statistics of the primary Depo4IoT studies? RQ1 has three sub-questions. RQ1.1 - In which years were the primary studies published and how many publications per year? RQ1.2 - In which targeted venues (Software Engineering-SE, IoT, Cloud/SOA, and Network) and venue types (conference, journal, workshop) were the primary studies published? RQ1.3 – What is the distribution of publications in terms of academic and industrial affiliation?

RQ2: What are the primary Depo4IoT approaches and how advanced are they? RQ2 is broken down into the following sub-questions. RQ2.1: How do the primary Depo4IoT approaches support IoT orchestration? RQ2.2: How do the primary Depo4IoT approaches support IoT deployment? RQ2.3: Are there any primary approaches explicitly addressing trustworthiness aspects? RQ2.4: Are there (run-time) adaptation mechanisms supported in the primary Depo4IoT approaches? RQ2.5: Are there industrial case studies or empirical studies used to evaluate the primary Depo4IoT approaches?

RQ3: What are the open issues to be further investigated in this field? Based on the characteristics of the primary Depo4IoT studies, we want to find out the open issues that would deserve more investigation in the future and some potential directions to tackle these issues. RQ3.1 - What are the open issues of Depo4IoT research? RQ3.2 - What research directions could be recommended for tackling the open issues?

3.1.1.2 Search and selection process of primary studies

From the research questions, we have derived a set of keywords and a search string for searching primary studies in four main publications repositories: IEEE Xplore[13], ACM DL[14], Scopus[15], and Science Direct[16]. Search string: *("Internet of Things" OR IoT OR "Web of things" OR WoT) AND (“Orchestration” OR “Deployment” OR “Choreography” OR “Topology” OR “Composition” OR “Dataflow”) AND ("Tool" OR "Middleware" OR "Service" OR "Framework").* To complement for the automatic search process on repositories, we have also conducted manual searches on the known studies and checked the latest related work of those studies.

To reduce the bias in the selection process, we clearly predefined the inclusion and exclusion criteria for selecting primary studies based on our research questions. The primary studies must meet ALL the following inclusion criteria (IC):

- (IC1) The studies must propose a deployment OR orchestration approach.

- (IC2) The studies must be explicitly for IoT area, either in general or in a specific application domain of IoT.

- (IC3) The studies must have software engineering approaches as software is the main drive of deployment and orchestration for IoT.

We also excluded non-peer-reviewed or unpublished paper, white paper, technical report, thesis, patent, general web page, presentation, book chapter, paper not in English.

From thousands of candidate papers in both automatic and manual search processes, we have systematically reviewed and selected 69 primary studies for answering our research questions. Figure 2 give an overview of our search and selection process of the SMS.

[13] http://ieeexplore.ieee.org
[14] https://dl.acm.org
[15] https://www.scopus.com
[16] https://www.sciencedirect.com

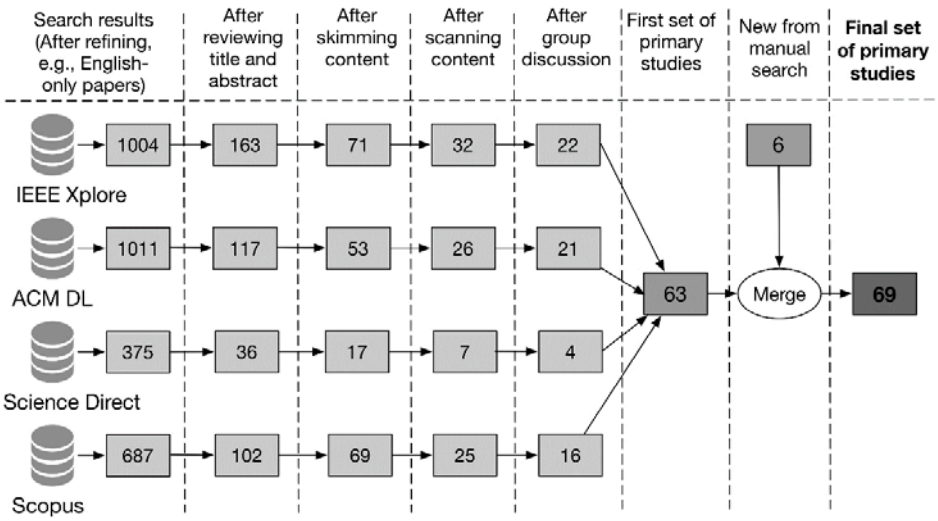

Figure 2:Overview of the search and selection process of the SMS

3.1.1.3 Taxonomy for data extraction and analysis

To analyse the primary studies for answering our research questions, we have defined a taxonomy as a classification and comparison framework for the primary studies. Some of the concepts in the taxonomy are extracted from former work on cloud deployment modelling languages [30]. Figure 3 shows our taxonomy that consists of the key aspects of deployment and orchestration for IoT. We group these key aspects into the following main categories: deployment and orchestration support, design support, and advanced support. We extracted and synthesised data from the primary studies to answer our research questions based on this taxonomy. Basically, the taxonomy provides the key aspects of orchestration and deployment approaches for IoT such as orchestration support (CommunicationType), deployment support (CloudProvisioning, DeploymentEngineType), adaptation (Runtime, Type), monitoring (RuntimeInfoToDesign), design support (Language, SpecificationCapabilities), tool scope (Pragmatic, TargetInfrastructure), evaluation type of approach (academic or industry case study, empirical study), and application domain where the approaches address. We also included in the taxonomy some ENACT project-specific aspects to compare and analyse the orchestration and deployment approaches for IoT such as trustworthiness (Safety, Security, Resilience, Reliability, Privacy). Finally, we included some aspects about the meta-info of the primary studies such as publication year, publication venue to be able to reason about the trends and landscape of research in this domain.

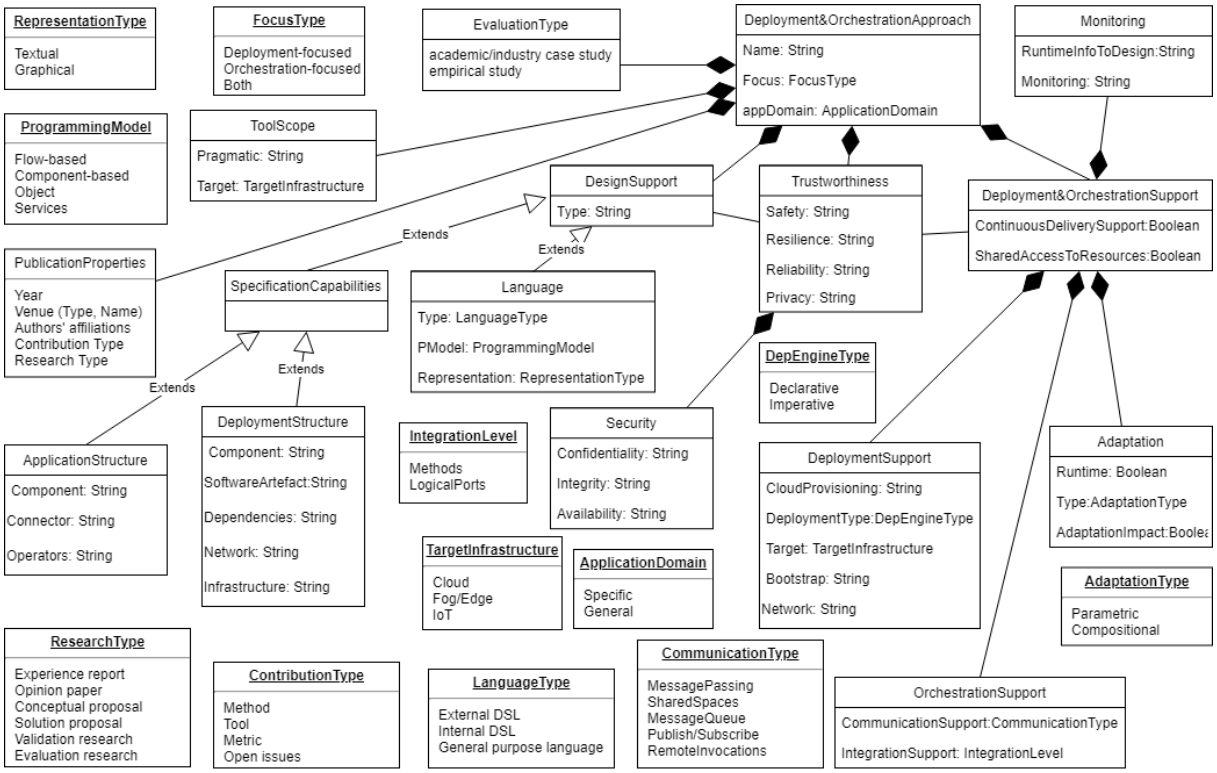

Figure 3: Our classification and comparison framework

The details of our taxonomy can be found in the Appendices. In the next section, we present the highlights of the results.

3.1.2 Some highlights of our SMS' results

The attached report is in the Appendix A . In the following subsection, we present the main results of our SMS extracted from this list to answer the RQs.

3.1.2.1 Overview

Answering RQ1.1: as we can see in Figure 4 (left), the earliest primary study was published in 2008, when IoT research was starting to emerge. There is a sharp rise in the number of publications in the last two years (2016, 2017), especially regarding the numbers of journal and conference papers (2016: 3J, 6C and 2017: 7J, 18C). This rise shows the crucial need of IoT research and more attention to this research area has gained from the research community. We conducted our search process early in March 2018 and already found five primary studies published (3J, 2C). We would expect that the total number of primary studies published in the whole year 2018 continues the trend of more and more research being done in this area.

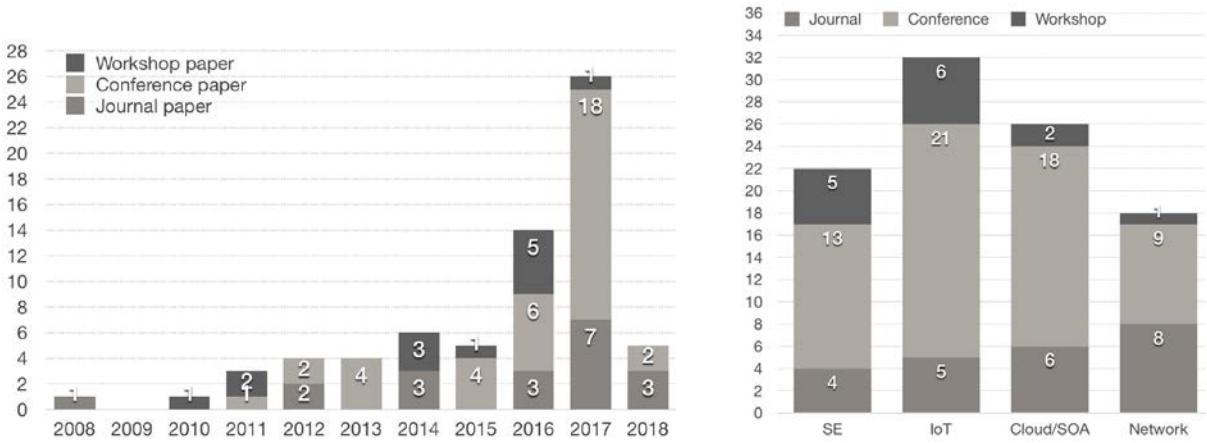

Figure 4: Primary studies published per year, per venue type (left) and Research topics per publication venue (right)

IoT is a heterogeneous research area that spans in multiple relevant research domains such as Software Engineering (SE), Cloud or Service-Oriented Architecture (SOA), Network, and recently specialized IoT research domain. Answering RQ1.2: Figure 4 (right) shows the times that each research domain appears in the calls for papers in the publication venues where the primary studies published are quite close (SE: 22, IoT: 32, Clould/SOA: 26, Network: 18). These numbers do reflect the heterogeneous nature of IoT research. Due to the increasing popularity of the IoT research domain, and because the tools for the deployment and orchestration of application and services in the cloud have lately reached a high level of maturity, the focus for the research on deployment and orchestration tools has moved toward the IoT and Edge spaces.

Answering RQ1.3: By checking the affiliations of the authors, we can see in Figure 5 (left) that most of the authors of the primary studies are researchers (86%). The involvement of industry in this research is still very limited, which is understandable for a relatively new area like IoT. Answering RQ2.5: Regarding the types of case studies used for evaluating the primary approaches, Figure 5 (right) also shows the similar dominance of academic approaches. But, the number of industrial case studies or empirical studies account for 13%, which is still encouraging. We would call for more collaboration between academia and industry in this practical research area.

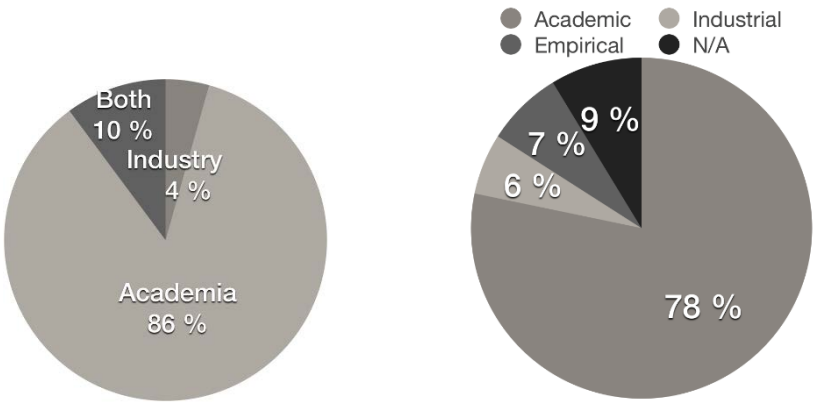

Figure 5: The affiliations of authors (left) and the Evaluation case studies (right)

3.1.2.2 Deployment and Orchestration support

This section gives our answers to the RQ2.1 and RQ2.2. Figure 6 (left) shows the distribution of primary studies based on the focus: orchestration, or deployment, or both orchestration and deployment. We can see that the orchestration-focused studies are nearly double the deployment-focused studies. The main reason could be that the orchestration-focused studies are around IoT data mash-up at cloud or edge level, which we show later in Figure 6 (right). These studies do not technically contribute to the low-level orchestration of IoT at IoT devices or gateways but rather make assumptions on low-level IoT

infrastructure. About one-third (37%) of the approaches support both deployment and orchestration. But we would like to note that the degrees of support for deployment and orchestration are not at the same level. In other words, approaches that support deployment somehow also support orchestration (as part of the deployment specifications overlap with orchestration specifications). However, the orchestration support in these cases is often at a higher level of abstraction than the orchestration-focused approaches that offer specific support for orchestration (*i.e.*, Logical Port vs. method level).

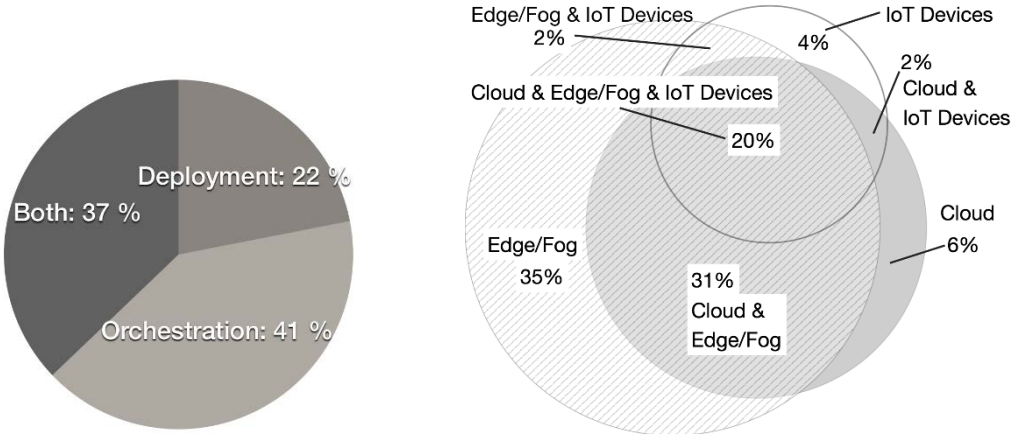

Figure 6: Deployment/orchestration focus (left) and Target infrastructure (right)

Figure 6 (right) shows how the Depo4IoT studies support for different layers of IoT infrastructure: cloud, edge, or IoT devices. Most studies (71%) discuss about mashing up data streaming from IoT devices at cloud (7%) or edge/fog (32%) or both (32%). But, few studies (29%) really support orchestrating and deploying software on IoT devices. We would argue that deployment and orchestration at IoT devices are the most challenging research problems because of the diversity of IoT devices, their networking protocols, and their different computing resource constraints. To really support for modern real IoT systems in which trustworthy aspects are crucial, research must advance to the technical details of edges and IoT devices. Even among the 29% mentioned above, we find very limited support for modern IoT systems as we discuss in the following paragraphs.

Looking closer into the studies that support deployment (59%, Figure 6, left), we find that nearly two-third (64%, Figure 7, left) of them have declarative deployment type vs. one-third have imperative type. We would argue this is because imperative approaches are typically more complicated to specify and reuse (as they require the specification of a deployment plan). On the other hand, they give full control over the deployment process, thus allowing its optimization. Also, among 59\% of the studies that support deployment, less than half (46%) have cloud provisioning before deployment (Figure 7, right). This means only one-third (59% times 46%) of the total primary studies have cloud provisioning, which is an important feature in any advanced approach.

Figure 7: Deployment engine (left) and Cloud provisioning (right)

Looking closer into the studies that support orchestration (78%, Figure 6, left), we find that the communication types in orchestrating IoT components are diverse (Figure 8, left). However, we observe that approaches offering the best decoupling in time and space [31] (between subscribers and publisher),

i.e., message passing, message queues and publish/subscribe, are largely adopted (73% in total). The integration level at methods (66%, Figure 8, right) for orchestration is more common than at logical port (34%). Only 11% support both levels, which often needed for orchestrating more complex IoT systems.

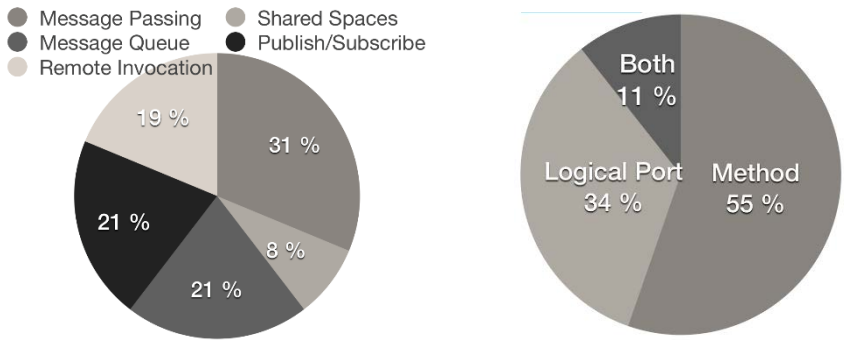

Figure 8: Orchestration communication types (left) Integration level (right)

3.1.2.3 Design support

This section gives more answers to the RQ2.1 and RQ2.2. We discuss the design support aspects, which are applicable for both orchestration and deployment. Figure 9 (left) shows that the most common term to depict deployment and orchestration entities is *service*, which is significantly more often than *component* and *node*. We find that most orchestration approaches have used services for data mash-up. *Node* term seems getting more common in supporting both deployment and orchestration (*e.g.*, in Node-RED-based approaches). The total number of primary studies that provide technical information about connectors is 43, which is two-third of the primary approaches. High-level operators (like BPMN operators) for supporting deployment and orchestration are not common because very few have been mentioned in the approaches.

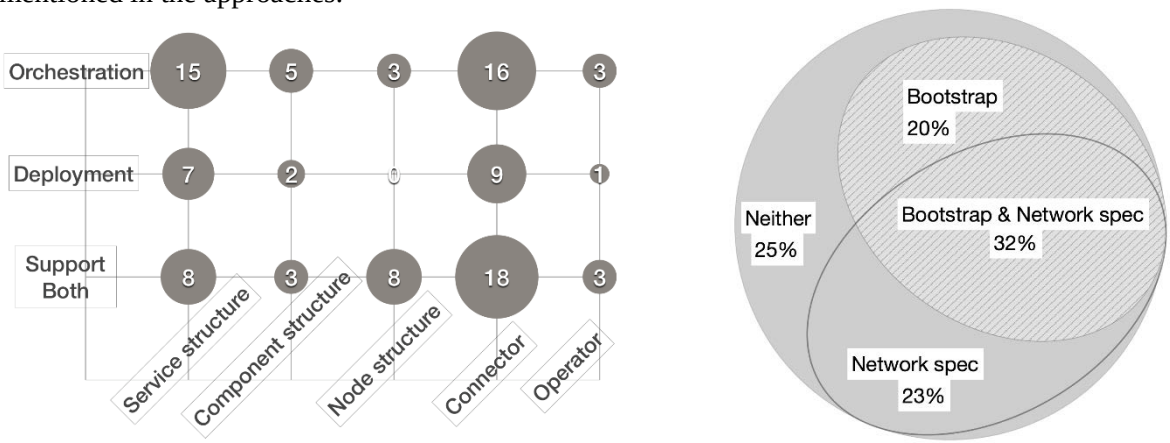

Figure 9: Application structure (left) and Bootstrap and network specification (right)

Any thorough approaches, especially for deployment and/or orchestration at IoT devices level, should have provided technical information about bootstrap and network specification. But, Figure 9 (right) shows that only less than one-third (32%) of the deployment approaches have provided any such information. This could be interpreted that most current approaches do not address the technical details of deployment and make assumptions on the support of underlying operating systems or execution engines. We also find in Figure 9 (right) only about half having either bootstrap (52%) or network specification (55%) with deployment. A thorough deployment approach should have included network specification(s) to support for low-level deployment on IoT devices, where diverse network topologies and protocols must be considered.

Aligned with the popular of "service" discussed in the previous paragraphs, service-oriented model is the most common (34%, Figure 10, left) programming model in the approaches, followed by component-based (19%), and flow-based (13%). The language support is an important aspect to make approaches more practical, but more than one-third of the approaches do not provide language support

for deployment and orchestration (Figure 10, middle). Using domain-specific languages (DSL) seems to be more popular than general programming languages (GPL). Figure 10 (right) shows that textual is the most popular form of language support (28%), compared to graphical (17%). Some approaches (12%) do have both graphical and textual formats.

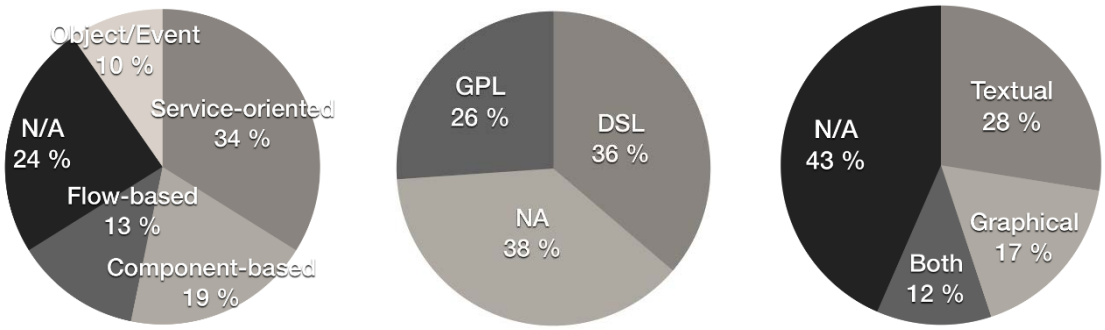

Figure 10: Programming model (left), Language types (middle), and Representation kinds (right)

3.1.2.4 Trustworthy and advanced supports

This section gives our answers to the RQ2.3 and RQ2.4. Trustworthy aspects must be supported in the orchestration and deployment of modern IoT systems. However, Figure 11 (left) shows that very few primary studies address trustworthy aspects: security (18 out of 69 studies), privacy (8), resilience (6), reliability (8). Among the trustworthy aspects, security is the most addressed one in the primary studies. None addresses safety, which should be also a crucial property of critical IoT systems. Figure 11 (right) shows that few primary studies provide advanced supports such as monitoring, run-time adaptation, shared access. It is worth to note that the approaches supporting both orchestration and deployment are more likely to support more advanced features such as monitoring and run-time adaptation (13, 12) than the approaches that solely supports only deployment (9, 8) or orchestration (6, 7). Shared access to resources is a rare feature in the existing approaches.

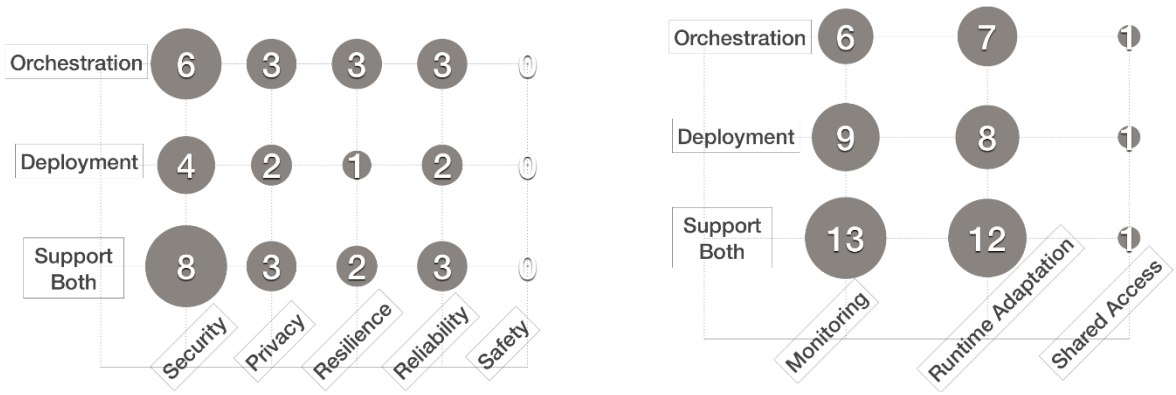

Figure 11: Trustworthiness aspects (left) and Advanced supports (right)

3.2 A systematic literature review of the key orchestration and deployment approaches for IoT

The results of the SMS give us a high-level overview of the research landscape of the existing Depo4IoT studies. To dive into more technical details, we select the most significant studies from the SMS to conduct deep analyses on them and discuss in detail on the main technical gaps. In other words, the SMS is the preliminary step of our systematic literature review (SLR) dedicated specifically to the primary deployment AND orchestration approaches for IoT.

3.2.1 The protocol of our SLR

We conducted our SLR by following the guidelines in [27-29]. In the following, we refine our research questions (RQs), then clarify the inclusion and exclusion criteria for selecting the primary studies of the SLR. Next, we show our strategy to find and select the primary studies for answering the research questions of the SLR.

3.2.1.1 The research questions of the SLR

The key contributions of this SLR are our answers to the following research questions and sub-questions:

RQ1: What are the technical details of the primary Depo4IoT approaches?

- *RQ1.1 - What are the publication status of Depo4IoT research?*
- *RQ1.2 - How do the primary Depo4IoT approaches support IoT deployment and orchestration?*
- *RQ1.3 - What are the design support aspects of the primary Depo4IoT approaches?*

RQ2: How do the existing primary Depo4IoT approaches address the trustworthiness aspects?

- *RQ2.1 - How do the existing primary Depo4IoT approaches address the trustworthiness aspects?*
- *RQ2.2 - Have any primary Depo4IoT approaches provided advanced supports such as monitoring, adaptation, and shared access to resources?*
- *RQ2.3 - How mature are the approaches in terms of tool support and evaluation?*

RQ3: What are the current technical challenges to be further investigated in this field?

3.2.1.2 Search and selection process of primary studies

The SMS study was the first round (group discussion 1) to select the primary studies for the SLR. After the SMS, we got on a list of primary studies focusing on either deployment OR orchestration. Using the SMS as the basis for further research [29], we continued with the second group discussion to select the most significant Depo4IoT studies. In the second round of group discussion (group discussion 2), from the first list, we shortlisted 17 primary studies that have research contributions in deployment for IoT as shown in Figure 12. Our main exclusion criterion to filter papers for the SLR is: "Papers only dealing with orchestration without deployment are excluded."

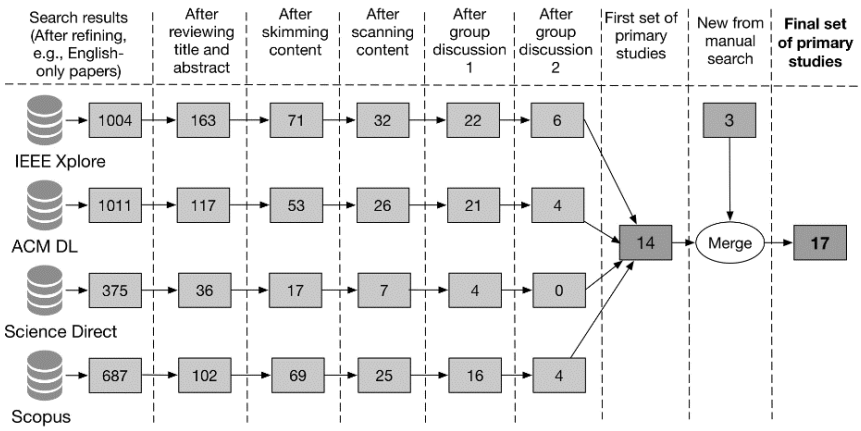

Figure 12: Overview of the search and selection steps of the SLR

We extracted data in detail from the list of 17 primary Depo4IoT studies of the SLR, following the taxonomy presented above. By analysing the extracted data, we can answer the research questions as presented in the next section.

3.2.2 Some highlights of our SLR's results

Table 2 gives an overview of the 17 primary Depo4IoT studies of the SLR. We have conducted in-depth analyses on these studies based on the taxonomy presented in Section 3.1.1.3. Answering RQ1.1, Figure 13 (left) shows a sharp rise in the number of primary Depo4IoT studies very recently. More than half of the studies (nine out of 17) were just published in 2017, which could indicate this research area is taking off to match with the important role of deployment and orchestration for IoT. One reason that Depo4IoT challenges are only receiving more attention recently is that so far IoT research might have focused more on fundamental IoT technologies, *e.g.*, approaches and tools for the development of IoT software components, communication protocols. But, once the IoT development approaches have advanced, the challenges of deployment and orchestration of IoT have popped up, which require more systematic and advanced supports in deployment and orchestration.

On another note, IoT is a heterogeneous research area that spans in multiple relevant research domains such as Software Engineering (SE), Cloud or Service-Oriented Architecture (SOA), Network, and IoT itself. Figure 13 (right) shows the times each research domain is the main topic in the calls for papers of the publication venues where the primary Depo4IoT studies are published. It is not surprising to see the dominance of IoT topic in the publication venues of the primary Depo4IoT studies (nine in total in Figure 13, right). But, the other related research domains are sharing publication venues with IoT (SE: four, Clould/SOA: four, Network: four).

Table 2: The primary deployment and orchestration studies of the SLR.

#	Year	Study	Title*	PV
S1	2017	FogTorch [32]	QoS-Aware Deployment of IoT Applications Through the Fog	J
S2	2017	ARCADIA [33]	A Middleware for Mobile Edge Computing	J
S3	2017	Chen et al. [34]	A Dynamic Module Deployment Framework for M2M Platforms	C
S10	2017	SoPIoT [35]	A Novel Service-Oriented Platform for the Internet of Things	C
S11	2017	Calvin [36]	Calvin Constrained: A Framework for IoT Applications in Heterogeneous Environments	C
S14	2017	Niflheim [37]	Niflheim: An end-to-end middleware for applications on a multi-tier IoT infrastructure	C
S15	2017	Verba et al. [38]	Platform-as-a-service gateway for the Fog of Things	J
S16	2017	Foggy [39]	Foggy- A Framework for Continuous Automated IoT Application Deployment in Fog Computing	C
S17	2017	TOSCA-BMWi [40]	A TOSCA-based Programming Model for Interacting Components of Automatically Deployed Cloud and IoT Applications	C
S12	2016	Cloud4IoT [41]	Cloud4IoT: A Heterogeneous, Distributed and Autonomic Cloud Platform for the IoT	C
S7	2015	D-NR [42]	Developing IoT Applications in the Fog: a Distributed Dataflow Approach	C
S9	2015	WComp [43]	A Generic Service Oriented Software Platform to Design Ambient Intelligent Systems	C
S13	2015	xWoT [44]	A component-based approach for the Web of Things	W
S5	2014	BeC3 [45]	BeC3: Behaviour Crowd Centric Composition for IoT applications	J
S8	2014	glue.things [46]	glue.things - a Mashup Platform for wiring the Internet of Things with the Internet of Services	W
S6	2013	SAaaS [47]	Application deployment for IoT: An infrastructure approach	C
S4	2011	D-LITE [48]	D-LITE: Distributed logic for internet of things services	C

*PV: Publication venue; J: Journal; C: Conference; W: Workshop; *: Sorted by year of publication*

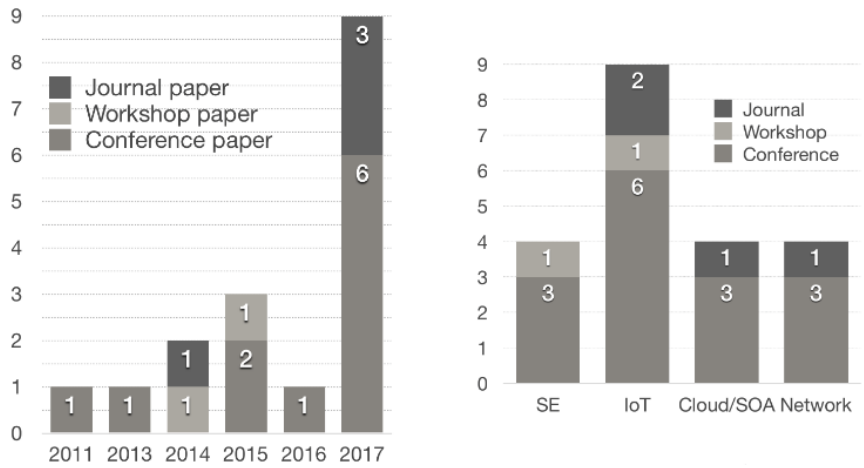

Figure 13: Publication of per year and Research topics per publication venue

Answering RQ1.2, Table 2 shows the technical details of the primary Depo4IoT approaches. Because of our selection criteria, most of the primary studies have their primary contributions in deployment for IoT. Even though a few primary studies (six out of 17) such as ARCADIA [S2, in Table 2], SoPIoT [S10], or WComp [S9], have orchestration as their primary focus, deployment is still present in these approaches, *e.g.*, in forms of mechanisms for the dynamic loading (aka. deployment) of WComp or OSGi components. We would expect modern IoT systems leveraging cloud computing should have used cloud provisioning techniques in the Depo4IoT approaches. However, this is not the case as only five approaches mention cloud provisioning. This observation illustrates that so far there is a lack of approaches/tools specifically designed for supporting the vertical depth of different IoT layers, *i.e.*, from cloud to fog/edge to IoT devices.

After analysing the data about deployment engine, we observe that more than two-third of the primary studies have declarative deployment type vs. less than one-fourth have imperative type. We would argue this is because imperative approaches are typically more complicated to specify and reuse (as they require the specification of a deployment plan). On the other hand, they give full control over the deployment process, thus allowing its optimization. Only [S17] has leveraged both declarative and imperative deployment types. This is because the authors of TOSCA-BMWi [S17] have focused on understanding and providing both declarative and imperative deployment types to TOSCA [49].

Looking at the target infrastructures, we can see that few studies really support orchestrating and deploying software on IoT devices. We would argue that deployment and orchestration on IoT devices are the most challenging research problems in Depo4IoT because of the diversity of IoT devices, their network protocols, and their different computing resource constraints. To really support for modern IoT systems in which trustworthy aspects are crucial, Depo4IoT studies must advance to the technical details of edges and IoT devices. Only in this way, IoT engineering can control the whole chain of IoT software deployed from cloud until IoT devices. Besides the heterogeneity, a huge challenge to bring IoT software engineering down to the low level of IoT devices could be the involvement of "black-box" software components or devices in IoT systems. Even worse, among the few studies that support Depo4IoT at IoT devices level, we find the technical details at IoT devices level very limited. WComp, SAaaS, BeC3 and D-LITE mention about managing and supporting deployment at IoT devices but no technical details are given. Niflheim also focuses more on the technical details of cloud and edge levels. Only Calvin[17], SoPIoT really provide some technical details at IoT devices level such as the service abstraction of devices in SoPIoT or hardware-specific features supported by Calvin.

[17] https://github.com/EricssonResearch/calvin-base/wiki/Tools

Table 3: How the primary deployment and orchestration approaches support IoT deployment and orchestration

Study	PRI	SEC	Cloud provisioning			Dep. En.		Target infras.			Bootstrap	Network	DCP	Communication type					Integ. lv.	
			PaaS	IaaS	SaaS	Dl	Im	C	E/F	De				MP	MQ	PS	RI	SS	M	LP
FogTorch [S1]	D	O	✓	✓	✓	✓			✓		Plain platforms (*e.g.*, Python, Ruby, and .NET)	Addressing		✓					✓	
ARCADIA [S2]	O	D		✓		✓		✓	✓		The ARCADIA controller	Addressing	Wifi, 3g				✓		✓	
Chen *et al.* [S3]	D					✓			✓		OSGi	Addressing	Wifi, Ethernet, Bluetooth		✓		✓		✓	
SoPIoT [S10]	O	D	✓	✓		✓		✓	✓	✓	SoPIoT runtime	Addressing	Wifi, Bluetooth, ZigBee		✓					✓
Calvin [S11]	O	D				✓		✓	✓	✓	Calvin runtime	Addressing		✓					✓	
Niflheim [S14]	D	O	✓	✓	✓	✓		✓	✓	✓		Addressing		✓					✓	✓
Verba *et al.* [S15]	D	O	✓			✓		✓	✓		OSGi-based Apache Karaf	Software-defined networking	434 MHz, rf24, Bluetooth and Xbee	✓	✓	✓			✓	✓
Foggy [S16]	D	O				✓			✓		Bluemix or Docker					✓				✓
TOSCA-BMWi [S17]	D	O	✓	✓		✓	✓	✓	✓		SSH and TOSCA runtime environment	Addressing		✓	✓	✓	✓	✓	✓	✓
Cloud4IoT [S12]	D	O				✓		✓	✓		Docker	Networking		✓	✓	✓	✓	✓		✓
D-NR [S7]	D	O					✓		✓		Distributed Node-RED	Addressing	Bluetooth		✓	✓			✓	
WComp [S9]	O	D				✓		✓	✓		WComp Container	Addressing	Wifi, Ethernet, Bluetooth, USB, EnOcean, ZigBee			✓	✓		✓	
xWoT [S13]	O	D					✓		✓		Plain platforms (*e.g.*, Node.js, Python, or Ruby)	Addressing	I2C (serial)	✓		✓			✓	
BeC3 [S5] and D-LITE [S4]	D	O				✓		✓	✓	✓	D-LITE	Addressing	Wifi, Ethernet, 802.15.4, 6LowPan	✓					✓	
glue.things [S8]	O	D				✓		✓			Node-RED, Cloud-Foundry	Addressing		✓		✓	✓		✓	
SAaaS [S6]	D	O	✓	✓	✓		✓	✓	✓	✓	Volunteered cloud manager	Addressing					✓		✓	

***PRI**: The primary focus; **SEC**: The secondary focus; **D**: Deployment; **O**: Orchestration; **PaaS**: Platform as a Service; **IaaS**: Infrastructure as a Service; **SaaS**: Software as a Service;*
***Dep. En.**: Deployment engine type; **Dl**: Declarative; **Im**: Imperative; **Infras.**: Infrastructure; **C**: Cloud; **E/F**: Edge/Fog; **De**: IoT devices; **DCP**: Device communication protocol;*
***MP**: Message passing; **MQ**: Message queue; **PS**: Publish/Subscribe; **RI**: Remote invocation; **SS**: Shared space; **Integ. lv.**: Integration level; **M**: Method; **LP**: Logical Port*

Most of the studies provide information about the bootstraps that the deployment approaches rely on. We can see in Table 3 that besides common (open source) run-time environments such as OSGi, Docker, Node-RED, Node.js, Python, there are solution-specific run-time environments that have been developed together with the Depo4IoT approaches such as Calvin run-time, WComp container, or D-LITE. In other words, we find two trends of using bootstraps in Depo4IoT. One trend uses as bootstrap main-stream execution environments, *e.g.*, Docker, Node.js, SSH and OS, which make them somehow easier to adopt (in the sense that it is easier to find a target with these main-stream execution environments). Another trend relies on more specific execution environments, *e.g.*, Node-red, Calvin. It is worth to note that this is typically the case of middleware, where the middleware itself must be running or installed on the target host. In this case, the mechanisms such as dynamic component loading, or class loading are used, *e.g.*, Node-red, WComp, OSGi.

Network specification among IoT elements is an important part of deployment and orchestration. Most of the primary approaches use straightforward network addressing. Only one approach provides an advanced support of software defined networking (Verba et al. [S15]). Less than half of the approaches provide some information about the supported communication protocols of the IoT devices such as Wi-Fi, Bluetooth, ZigBee, XBee. The primary studies do not really have explicit network specification and device communication protocols support but rather just briefly mention them. This observation is understandable because very few primary studies really support orchestrating and deploying software on IoT devices.

Table 3 depicts the different communication types used by the primary studies. We can see that the communication types in orchestrating IoT components are diverse and have a fair share in use, excepts Shared space (SS), which is rarely used. Overall, the approaches offering the best decoupling in time and space (between subscribers and publisher), *i.e.*, message passing, message queues and publish/subscribe, are largely adopted. Two studies TOSCA-BMWi and Cloud4IoT, which tick all the boxes for five communication types, are independent from communication used. The integration level at methods for orchestration is more common than at logical port level.

Only three studies Niflheim, Verba et al., and TOSCA-BMWi support both levels, which often needed for orchestrating more complex IoT systems. Indeed, orchestration is typically concerned with the behaviour of the system, which often has the integration at the method level based on the understanding of the semantics behind the methods. Vice versa, a deployment approach may not need to care about the actual behaviour of the system being deployed but just assumes the behaviour is correct. Here, the objective of deployment is to enable the communication between the elements of the system, which can be done at logical port level, not necessary at method level.

Table 4: The design aspects of the primary deployment and orchestration approaches

Study	Language support				Application structure		Deployment structure				
	Type	Name	Pro. model	Rep.	Entity	Connector	Entity	Dependencies	Softw. artifact	Infrastructure	Network
FogTorch [S1]	GPL	NA	component based	T	node	link	node	links	application	fog/edge	addressing
ARCADIA [S2]	DSL	NA	flow-based, services	G	microservice	method call	node	method call	node implementation	NA	addressing
Chen *et al.* [S3]	DSL	OSGi models	component-based	T	component	message, method call	component	method call	jar file	node	addressing
SoPIoT [S10]	DSL	SoPIoT script	services	T	services	message	services	composite service	service mashup	edge	addressing
Calvin [S11]	DSL	NA	dataflow, actor-based	T,G	actor (node)	port connection	actor	port connection	actor instance (implementation)	node (calvin runtime)	addressing
Niflheim [S14]	DSL	NA	microservices, component based	G	microservices	message	service	service bindings	java microservices	cloud, fog, mist	addressing
Verba *et al.* [S15]	DSL	Sensor Markup Lang	services		services	message	component	message	deployment, migration	container, PaaS	SDN
Foggy [S16]	GPL	NA	component based	T	node	method call	node	method call	application	fog/ edge	addressing
TOSCA-BMWi [S17]	DSL	TOSCA+	component based	T,G	node	relationship template	node	relationship template	implementation	container, PaaS	addressing
Cloud4IoT [S12]	DSL	NA	component based	T,G	component	link	container	links	image	container	addressing, overlay network
D-NR [S7]	DSL	Node-RED based	flow-based	T,G	node	dataflow	node+ sub-flow	master flow, sub-flows	node implementation	fog	addressing
WComp [S9]	DSL	WComp ADL	component, service, event	T,G	component, service	port connection	component, services	port connection, service bindings	lightweight service implementation	container	addressing
xWoT [S13]	DSL	xWoT	component, service, event	T	component	message	entities	resource	service mashup	edge	addressing
BeC3 [S5] and D-LITE [S4]	DSL	SALT	component, service	T,G	service	message	behavior	interaction patterns	D-LITE rules	D-LITE nodes	addressing
glue.things [S8]	DSL	Node-RED based	flow-based	G	node	message	node	node-flow	node implementation	cloud	addressing
SAaaS [S6]	NA	NA	NA	NA	service	method call	node	node-flow	NA	node	addressing

***GPL**: General programming language; **DSL**: Domain specific language; **Pro.**: Programming; **Rep.**: Representation kind; **G**: Graphical; **T**: Textual; **SDN**: Software defined networking; **NA**: Information not available*

Answering RQ1.3, based on Table 4, we discuss the design support aspects for both orchestration and deployment in the primary studies. The language support is an important aspect to make Depo4IoT approaches more practical. Table 4 shows that using DSL is much more popular than using GPL. Two studies FogTorch and Foggy use GPL such as Java to implement their deployment algorithms together with the IoT systems (FogTorch) or some components of the deployment framework (Foggy). Among the studies that use DSLs, there are some DSLs that are common such as flow-based programming of Node-RED. We can also see that textual form and graphical form are equally popular in language support. There are some primary approaches that do propose both graphical and textual formats [S4, S5, S7, S9, S11, S12, S17]. Component-based model and service-oriented model are the most common programming models in the Depo4IoT approaches, followed by flow-based.

We can also see from Table 4 that in most of the primary Depo4IoT studies, the aspects of application structure are almost identical to the corresponding aspects of deployment structure. Indeed, there is often an overlap between deployment and orchestration specifications. Even though, when the focus is deployment, orchestration specification stays at a higher level of abstraction (*e.g.*, microservice vs. node in ARCADIA) and the opposite for the approaches focusing on orchestration, deployment specification stays at a higher level of abstraction. Checking the deployment structure, we find that most approaches do provide some details about the software artefacts used ranging from node implementation (abstract)

to jar file or docker image (detail). Plain support for network specification (simply addressing) is common. Verba et al. [S15] is the only approach that provides software defined networking support.

Table 5: The trustworthy and advanced support aspects

Study	Trustworthy aspects					Mo.	Ada.	SAR	Tool support	
	Sec	Pri	Res	Rel	Saf				Pragmatic	Level
FogTorch [S1]			✓	✓			✓		QoS-aware deployments of IoT applications on Fog.	semi
ARCADIA [S2]						✓	✓		Middleware for applications over telecom networks and legacy data centers	auto
Chen *et al.* [S3]							✓		M2M platform for deployment on devices optimized to user service requirements	auto
SoPIoT [S10]				✓					Abstract over IoT devices and applications via service layer	semi
Calvin [S11]	✓						✓		Development, deployment and execution on resource constrained devices, Edge, and cloud	auto
Niflheim [S14]				✓				✓	Middleware for applications on a multi-tier IoT infrastructure	semi
Verba *et al.* [S15]			✓	✓					Deployment of microservices architecture	man
Foggy [S16]		✓				✓	✓		Deployment with optimal quality of service	semi
TOSCA-BMWi [S17]							✓		Generic language and tooling for deployment of Edge and Cloud-based systems	auto
Cloud4IoT [S12]							✓		Autonomic Cloud Platform for the IoT	auto
D-NR [S7]	✓		✓				✓		Building distributed dataflows	semi
WComp [S9]			✓			✓	✓	✓	Orchestration and dynamic execution environment for ubiquitous systems	semi
xWoT [S13]									The model compiler for generating almost ready code to deploy xWoT components	semi
BeC3 [S5] and D-LITE [S4]							✓		Mash up and composition tool for the IoT	auto
glue.things [S8]	✓	✓							Mashup platform for the IoT	auto
SAaaS [S6]									Sensing resources as a service	semi

***Sec**: Security; **Pri**: Privacy; **Res**: Resilience: **Rel**: Reliability; **Saf**: Safety; **Mo.**: Monitoring; **Ada.**: Adaptation; **SAR**: Shared access to resource;*

Answering RQ2 - On the support for trustworthiness: Table 5 shows that very few primary studies address trustworthiness aspects: security (four studies), privacy (two), resilience (three), reliability (four). Indeed, the trustworthiness aspects are only briefly mentioned in these few studies, neither with details nor in explicitly systematic manner. For example, regarding security, we do not find any primary studies that have put their approaches into the context of a Security Development Lifecycle[18]. None addresses safety, which should be also a crucial property of critical IoT systems. In the IoT context, safety often linked with actuation conflicts, which can occur, *e.g.*, when concurrent applications have a shared access to the actuators. We can also see that shared access to resources is a rare feature in the existing approaches because only two studies [S9, S14] have mentioned this feature. Even so, these two approaches have not gone further to provide real technical solutions. This is a hard problem that requires more extensive research work. The uncertain, dynamic, and partially known nature of the physical environment makes it very difficult or even illusory to assess at run-time the conformity of the effects of actions in this environment with deterministic models.

3.3 Other related approaches

For some years now, multiple tools have been available on the market to support the deployment and configuration of software systems – *e.g.*, Puppet[19], Chef[20], CFEngine[21]. These tools were first defined as configuration management tools aiming at automating 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 [50] (*i.e.*, systems deployed across multiple cloud solutions from different providers) such as CloudMF [51], OpenTOSCA [52], Cloudify[22], and Brooklyn[23]. Those are tailored to provision and manage virtual machines or PaaS solutions. In addition, similar tooling focus on the management and orchestration of containers – *e.g.*, Docker Compose[24], Kubernetes[25]. Opposed to hypervisor virtual machines, containers such as Docker containers leverage lightweight virtualization technology (in the case of Docker it was originally LinuX Containers (LXC), and now libcontainer), which executes directly on the operating system of the host. As a result, Docker shares and exploits a lot of the resources offered by the operating system thus reducing containers'

[18] Microsoft's SDL, https://www.microsoft.com/en-us/sdl
[19] https://puppet.com/
[20] https://www.chef.io/chef/
[21] https://cfengine.com/
[22] http://cloudify.co/
[23] https://brooklyn.apache.org
[24] https://docs.docker.com/compose/
[25] https://kubernetes.io

footprint. In addition, Docker exploits cgroup to share and manage hardware resources and UnionFS as fast and lightweight file system. Compared to classical virtual machines, containers do not provide full isolation but isolate processes from each other. Thanks to these characteristics, container technologies are not only relevant for cloud infrastructure but can also be used on Edge devices. It is also worth noting that within the TOSCA standardization process, a language for describing deployment of cloud application, an ad-hoc group has been created to investigate the extension of TOSCA toward Edge infrastructure.

On the other side, few tools such as Resin.io[26] and ioFog[27] 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 continuous deployment on IoT devices as depicted in Figure 14. Once the code for the software component to be deployed 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.*, ARMv6 for a Raspberry Pi) and a Docker image is created before being deployed on the target hosting device(s). However, Resin.io offers limited support for the deployment and management of software components on tiny devices that cannot host containers.

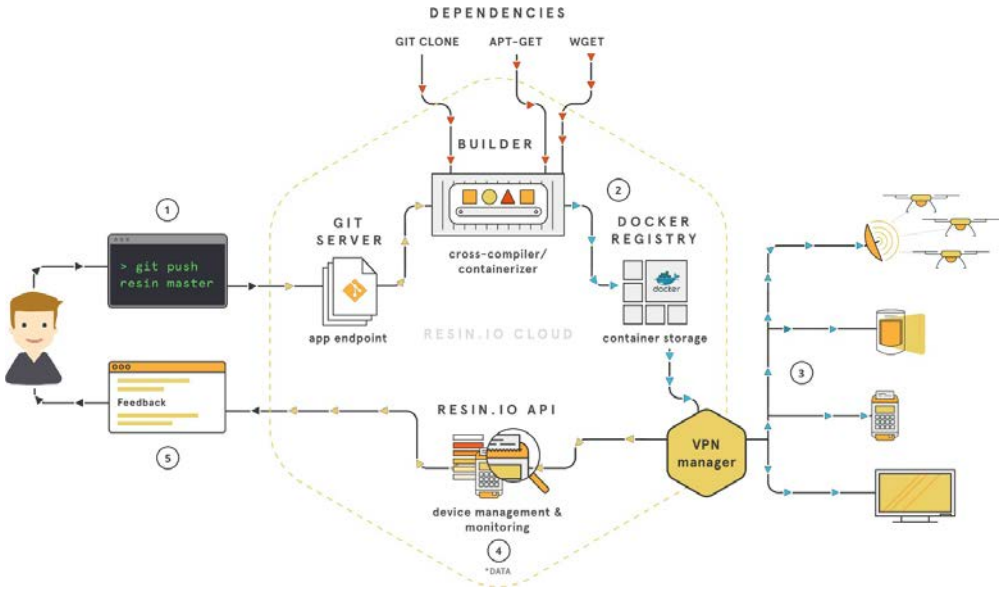

Figure 14: Code deployment using Resin.io[28]

So far, none of the tools aforementioned have specifically been designed for deployment across the whole IoT, Edge, and cloud space.

3.4 Analysis and discussion of the state of the art

From the results of our SMS, SLR, and other related approaches presented above, we analyse and discuss trends and the gaps in the state of the art regarding support for continuous orchestration and deployment for SIS, with trustworthiness.

Research on deployment and orchestration for IoT is getting much more attention: Our SMS and SLR have showed a sharp rise in the number of primary Depo4IoT studies very recently, which could indicate this research area is taking off to match with the important role of deployment and orchestration for IoT. One reason that Depo4IoT challenges are only receiving more attention recently is that so far IoT research might have focused more on fundamental IoT technologies, *e.g.*, approaches and tools for the development of IoT software components, communication protocols. But, once the IoT development approaches have advanced, the challenges of deployment and orchestration of IoT have popped up, which require more systematic approaches in deployment and orchestration. As we are discussing in the

[26] https://resin.io
[27] https://projects.eclipse.org/proposals/iofog
[28] https://resin.io/how-it-works/

following, the existing Depo4IoT approaches are still immature with technical gaps and lack of advanced supports and addressing trustworthiness aspects for IoT. In this context, the ENACT project's work will be very timely to address these issues.

The technical gaps in the deployment and orchestration approaches: In the SMS, we found that the number of orchestration approaches, often for IoT data mash-up, is nearly double the number of deployment approaches. The orchestration approaches for IoT data mash-up often make assumptions of readily deployed IoT systems in operation. The IoT deployment's challenges must be focused more because deployment would be more fundamental for IoT than orchestration. Without proper deployment approaches to deploy IoT systems, orchestration approaches cannot thrive.

From both the SMS and the SLR, it is worth to note that most of the primary Depo4IoT approaches do not really support deployment and orchestration at IoT devices. The SMS shows that network communication and IoT device communication protocols are not considered, or only briefly mentioned in the primary studies. The existing primary studies are still quite immature in the technical aspects of deployment and orchestration at IoT devices level such as bootstrap, and network specification supports. The trend of compute moving from cloud towards the edge and "things" is obvious but regarding Depo4IoT, the SLR shows that cloud provisioning is rare and does not cover the vertical depth of different IoT layers, *i.e.*, from cloud to fog/edge, to IoT devices. Not covering the vertical depth of all IoT layers is a problem because in practice, supporting deployment and orchestration at (resource-constrained) IoT devices is very challenging for any Depo4IoT approach. Only by going into low-level details, IoT engineering can really control the whole chain of IoT software deployed from cloud until IoT devices. In this way, it will be more likely that the trustworthy aspects and advanced supports can be addressed more systematically and efficiently.

The lack of advanced supports and addressing the trustworthiness aspects: The existing Depo4IoT studies are immature in providing advanced supports, *i.e.*, regarding monitoring, adaptation, and shared access to resources. Moreover, the number of existing Depo4IoTstudies addressing trustworthy aspects is very small. Even among those studies, we have not really found any that explicitly and systematically considered trustworthy aspects. Based on these observations, we would propose to do research focusing on the real, low level technical details of deployment and orchestration, especially deployment. New Depo4IoT research should address more thoroughly and systematically the trustworthy aspects and advanced supports, which are crucial for modern IoT systems. Only by going into low-level details, IoT engineering can really control the whole chain of IoT software deployed from cloud until IoT devices. In this way, it will be more likely that the trustworthy aspects and advanced supports can be addressed more systematically and efficiently.

The lack of research collaboration between industry and academia: The dominance of academia-only in this research area shows that there is a big gap to make the proposed approaches more practical and closer to the needs in industry. There should be more collaboration between academia and industry because of the important role that deployment and orchestration have in the engineering of critical modern IoT systems. In the scope of the ENACT project, we aim to foster the collaboration between research partners and industry partners.

3.5 Towards GeneSIS, a framework for the deployment and orchestration of SIS

In this section, we first introduce the requirements that drove the development of GeneSIS (cf. Section 3.5.1). Then, Section 3.5.2 present the baseline technologies we plan to use to develop our approaches and tools for supporting continuous orchestration and deployment of trustworthy SIS. In particular, we briefly recall Model-Based Engineering (MDE), which is an approach for tackling the complexity of software and systems such as SIS and ThingML that we will use to develop our approaches and tools upon. Finally, we show our plan and strategy to develop a core technology enabler for supporting continuous orchestration and deployment of trustworthy SIS, called GeneSIS in Section 3.5.3.

3.5.1 Requirements

In this section we first present high-level requirements that will drive the development of the Secure Orchestration and Deployment tool before Table 6 introduces more concrete requirements.

The following simple motivating example is used to motivate our requirements. SensAct is a company delivering application for smart building that connects IoT sensors and actuators to analytics services running in the cloud. SensAct has to develop and deploy a new SIS in a house which is already equipped with some sensors and actuators facilitating dynamic control of blinds and lights with controllable windows and heating system. The new system should maximize exploitation of daylights and regulate the in-door temperature whilst minimizing the energy consumption. A purposely simplified version of the part of the system related to the control of the blinds is depicted in Figure 15. In short, a RFXtrx433E Transceiver is used to control the blinds as well as to receive temperature and humidity metrics from probes installed in different location of the house. This device is plugged to a Raspberry Pi (called Raspberry Pi2 in Figure 15) via a USB port. The latter is running a software service responsible for managing the access to the blinds. An Arduino with a screen is used to display (i) the temperature in the house or (ii) an alarm message in case one of the blinds is not responding. In addition, it is equipped with a button to close or open all the blinds. This Arduino is plugged via serial port to a second Raspberry Pi (called Raspberry Pi in Figure 15). The latter is hosting two software services to (i) manage the accesses to the Arduino and (ii) upload all data from the sensors and actuators in a data store running on Amazon EC2.

SensAct identifies that the actuator to control the blinds can be accessed concurrently (*e.g.*, the blind is used to control both the temperature and the light level) for different behaviours, which have conflicting goals and effects (*i.e.*, actions optimizing temperature may be hindered by actions optimizing light level). A device controller (*i.e.*, components whose role is to manage access to actuators) is added to the SIS to handle this problem.

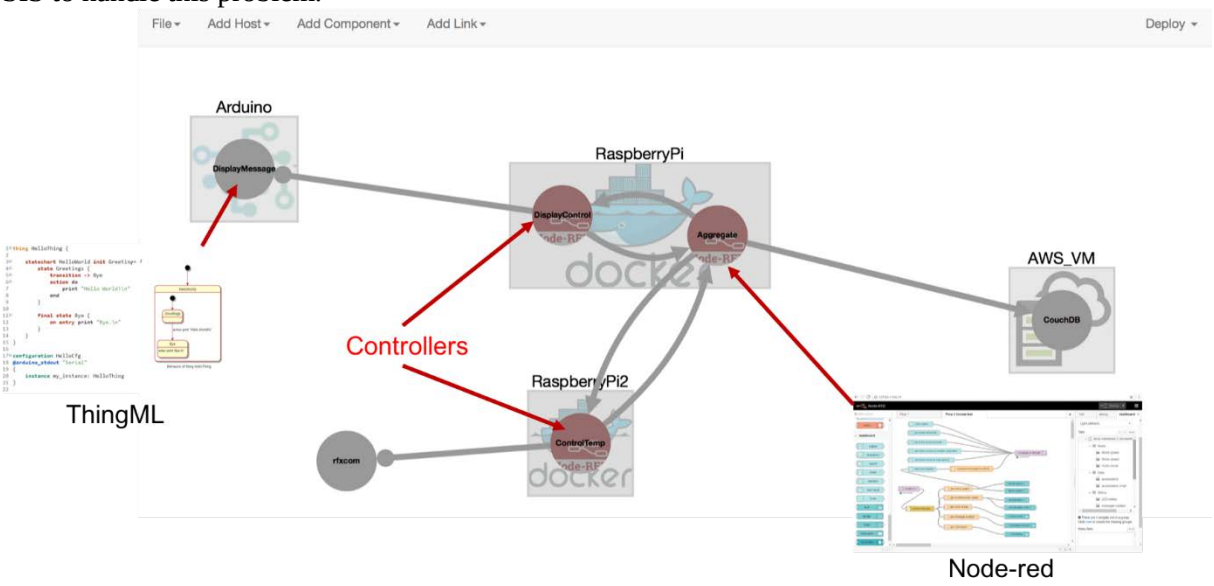

Figure 15: Screenshot of the motivating example in the GeneSIS web editor.

This example motivates for the following requirements that should be addressed by GeneSIS:

- **Separation of concerns and reusability**: Our solution shall provide a modular, loosely-coupled specification of the data flow and its deployment so that the modules can be seamlessly substituted and reused. Elements or tasks should be reusable across scenarios, for example, deploying Node-red.
- **Abstraction and Infrastructure independence**: Our solution shall provide a domain-specific language to describe the orchestration and deployments of SIS over IoT, edge and cloud infrastructures in both a device- and platform-independent and -specific way. In addition,

GeneSIS shall provide a continuously up-to-date, abstract representation of the running system. This facilitates the reasoning, simulation, and validation of operation activities.

- **White- and black-box infrastructure:** Our framework shall support white- and black-box devices. This requires coping with various degrees of delegation of control over underlying infrastructures and platforms.
- **Trustworthiness:** Our solution shall integrate with the ENACT solutions for trustworthiness. This includes security and privacy mechanisms as well as facilitating the identification and management of actuation conflicts, in particular, concurrent accesses to actuators.

Table 6 provides a detailed list of requirements from WP2 for GeneSIS (a.k.a., the secure orchestration and deployment enabler). GeneSIS will be used in all use cases: Intelligent Transport System (UC1), Digital Health (UC2), and Smart Building (UC3).

Table 6. WP2 requirements for GeneSIS

ReqID	Requirement	Description
UC1-3 **R1**	Scalability	GeneSIS should be able to deploy SIS involving hundred sensors/actuators.
UC1-3 **R2**	Scalability	The GeneSIS modelling language should be able to represent deployments involving hundred sensors/actuators.
UC1-3 **R3**	Trustworthiness and Agility	GeneSIS should support the re-deployment (*e.g.*, moving one software node from one host to another), re-configuration, and update (install new version of a software node) of software components.
UC3 **R4**	Trustworthiness	GeneSIS should help identifying direct actuation conflicts (*i.e.*, concurrent accesses to a same component).
UC1-3 **R5**	Scope	GeneSIS should be able to deploy SIS involving IoT, edge and cloud infrastructures.
UC1-3 **R6**	Scope	The GeneSIS modelling language should be able to represent deployment over IoT, Edge, and cloud infrastructure
UC1-3 **R7**	Trustworthiness	The GeneSIS language will support the specification (i) of the security mechanisms to be deployed and (ii) of metadata (*e.g.*, software version) for each of the elements in a model.
UC1,3 **R8**	Integration	GeneSIS should properly integrate with classical IoT middleware (*e.g.*, SMOOL, SOFIA2)
UC1-3 **R9**	Elasticity	GeneSIS should support the provisioning of cloud resources.
UC1-3 **R10**	Elasticity	The GeneSIS modelling language should provide the necessary concept for specifying the cloud resources to be provisioned.
UC2 **R11**	Monitoring	The GeneSIS language should include the necessary concepts to reflect directly in the language run-time data. In particular information about the deployment and infrastructure status as well as about the execution flow of ThingML programs.

In the following Section, we present the baseline technologies that will be used by GeneSIS to fulfil these requirements.

3.5.2 Baseline technologies

Model-driven engineering (MDE) [53] is a branch of software engineering which aims at improving the productivity, quality, and cost-effectiveness of software development by shifting the paradigm from code-centric to **model-centric.** This approach is commonly summarised as "model once, generate anywhere"**.** Models and modelling languages as the main artefacts of the development process enable developers as well as reasoning engines to work at a high level of abstraction by focusing on cloud concerns rather than implementation details. Model transformation serves as the primary technique to generate (parts of) software systems restrains developers from repetitive and error-prone tasks.

MDE has been applied to cloud computing in many works such as the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [54] which provides a language for specifying the topology of cloud applications along with the processes for their orchestration, the

CloudScale[29] FP7 project models cloud-based system design alternatives and the analysis of their effect on scalability and cost. A challenge has been that models are not synchronized with the running system and may become irrelevant as the system evolves. CloudML developed within the REMICS[30], MODAClouds[31], and PaaSage[32] FP7 projects leverages Models@Run-time to address this issue, also supporting run-time adaptation as modifications on the model can be enacted dynamically on demand.

MDE techniques have been investigated to design and reconfigure a network of resource-constrained devices. One trend is to provide an architecture-based approach to design and build flexible embedded systems, *e.g.*, the Koala model [55] and Think [56]. Kevoree[33] relies on models at run-time to support the dynamic adaptation of distributed cloud and cyber physical systems supporting hot deployment of component types (*i.e.*, not foreseen deployments). Fleurey et al. describes a DSL to derive adaptive firmware for microcontrollers [57], modelling adaptive behaviours of embedded systems and compiling these models. The HEADS FP7 EU project proposes ThingML [58], a modelling language with a tool chain for engineering embedded and distributed systems, especially IoT systems on various IoT platforms. More specifically, ThingML is a practical MDE approach comprises a modelling language, a set of tools, and a methodology that target resource constrained embedded systems such as low-power sensor and microcontroller-based devices, gateways. More importantly, ThingML provides methods and tool support for facilitating the integration of resource constrained embedded systems with more powerful computing resources such as servers and cloud, which are the common for IoT systems. In other words, ThingML facilitates the collaboration between IoT service developers and IoT infrastructure/platform operators in a large range of processing nodes and protocols with high heterogeneity [59]. We plan to develop on this special integration support of ThingML to provide the ability of continuous deployment, operations, and management of applications across IoT, edge, and cloud infrastructures.

ThingML is developed as a textual domain-specific modelling language (DSML) that includes concepts to describe both software components and communication protocols. The formalism used is a combination of architecture models, state machines and an imperative platform-independent action language. The ThingML toolset includes text editors to create and edit ThingML models, a set of transformations to create diagrams from ThingML models and an advanced multiplatform code generation framework [58] that support multiple target programming languages. One of the key features is ThingML's template mechanism integrated into the language to integrate with third-party (or legacy) API, rather than re-developing them from scratch. This makes ThingML practical for integrating with different existing IoT services, supporting many kinds of target platforms of IoT systems such as Java, NodeJS, Arduino, C.

Another practical aspect is that ThingML allows abstracting from the heterogeneous platforms and IoT devices to model the desired IoT system's architecture. This is helpful because in practice, platforms and devices, as well as the final distribution of software components, typically are not known during the early design phases. The architecture model consists of components, ports, connectors, and asynchronous messages. Once the general architecture is defined, ThingML supports specifying the components' business logic in a platform-independent way using statecharts and the action language. ThingML statecharts include composites, parallel regions, and history states. The state machines typically react to events corresponding to incoming messages on a component's port. The action language lets developers specify the guards, actions, variables, and functions in the component and lets the state machine send and receive messages on the component's port [59].

3.5.3 Conceptual design of the GeneSIS framework

The objective of the GeneSIS framework is to support the orchestration and deployment of IoT systems whose software building blocks can be deployed over IoT, edge and cloud infrastructures. The target user group for our framework is thus mainly software developers and architects.

[29] FP7 CloudScale: http://www.cloudscale-project.eu/
[30] FP7 REMICS: http://www.remics.eu/
[31] FP7 MODAClouds: http://www.modaclouds.eu/
[32] FP7 PaaSage: http://www.paasage.eu/
[33] http://www.kevoree.org/

GeneSIS includes: (i) a domain specific modelling language to model the orchestration and deployment of SIS across the IoT, edge and cloud spaces; and (ii) an execution engine that supports the orchestration of IoT, edge and cloud services as well as their automatic deployment over IoT, edge and cloud resources.

Our framework is agnostic to any development paradigm and technology, meaning that the developers can design and implement the applications based on their preferred paradigms and technologies. The GeneSIS framework is also agnostic to any specific business domain.

To avoid the limitation that for each targeted infrastructure or platform the deployment engine and the deployment actions or code has to be tailored and customized for the specific platform, the GeneSIS deployment engine will embed a plug-in mechanism.

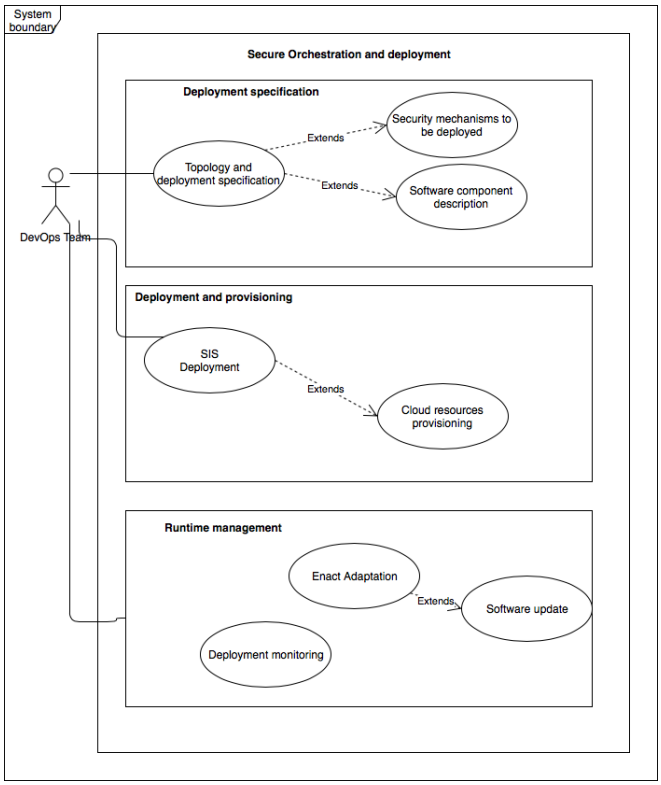

Figure 16: Use case diagram - GeneSIS

The features offered by GeneSIS are summarized in Figure 16. It is worth noting that the work presented in this section is still in its early stage and might evolve in the forthcoming deliverable D2.2.

3.5.3.1 The GeneSIS Modelling Language

The GeneSIS Modelling language is inspired by our former work on CloudML and is thus inspired by component-based approaches to facilitate separation of concerns and reusability. In this respect, deployment models can be regarded as assemblies of components. In general, one of the objectives when we developed this language was to keep it as simple as possible, with minimal set of concepts but still keeping it extensible. The type part of the GeneSIS modelling language metamodel is depicted in Figure 17.

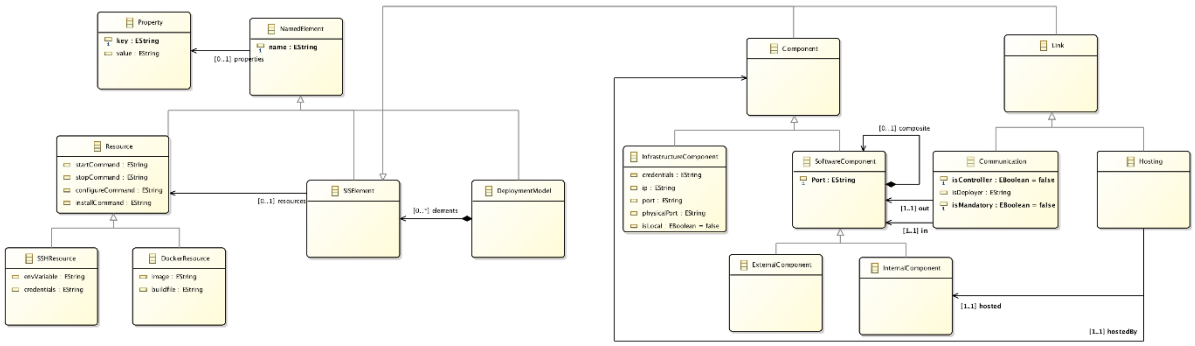

Figure 17: Type part of the GeneSIS metamodel

A *DeploymentModel* consists of *SISElement*s, which can be associated with properties. The two main types of *SISElement*s are *Component*s and *Link*s.

A *Component* represents a reusable type of node that will compose a *DeploymentModel*. A *Component* can be a *SoftwareComponent* representing a piece of software to be deployed on a host (*e.g.*, an Arduino sketch can be deployed on an Arduino board). A *SoftwareComponent* can be an *InternalComponent* meaning that it is managed by the framework (*e.g.*, an instance of Node-red to be deployed on a Raspberry PI), or an *ExternalComponent* meaning that it is managed by an external provider (*e.g.*, a database offered as a service) or is hosted on a blackbox device (*e.g.*, RFXCom transceiver). The property *port* of a *SoftwareComponent* represents a logical port. A *SoftwareComponent* can be associated with *Resource*s (*e.g.*, scripts, configuration files) adopted to manage its deployment life-cycle (*i.e.*, download, configure, install, start, and stop). In particular, there are two main predefined types of resources: *DockerResource*s and *SSHResource*s.

An *InfrastructureComponent* provides hosting facilities (*i.e.*, it provides an execution environment) to *SoftwareComponent*s. The properties *IP* and *port* represent the IP address and port that can be used to reach the *InfrastructureComponent*. The property *isLocal* depicts that a physical connection is required to deploy a *SoftwareComponent* on an *InfrastructureComponent* via a *PhysicalPort*.

There are two main types of *Link*s. A *Hosting* depict that an *InternalComponent* will execute on a specific host. This host can be any *Component*, meaning that it is possible to describe the whole software stack required to run an *InternalComponent*. A *Hosting* can be associated with *Resource*s specifying how to configure the components so that the contained component can be deployed on the container component. A *Communication* represents a communication binding between two *SoftwareComponent*s. A *Communication* can be associated with *Resource*s specifying how to configure the components so that they can communicate with each other. The property *isMandatory* of *Communication* represents that the *SoftwareComponent*s depend on this feature (*e.g.*, the service hosted on Raspberry Pi2 will not work if the communication with RFXtrx433E is not properly set up). The property *isController* depicts that the *SoftwareComponent* associated to the *in* attribute is controlled by the other (*e.g.*, all messages going to the Arduino should pass through the service hosted on Raspberry Pi). Finally, the property *isDeployer* specifies that the *InternalComponent* indicated in the property should be deployed from the location of the other *SoftwareComponent* (*e.g.*, the artefact to be executed on the Arduino will be deployed from the Raspberry Pi).

It is worth noting that we applied the *type-instance* pattern [60] to *SoftwareComponent*s thus facilitating the reuse of generic type of components. This pattern exploits two flavours of typing, namely *ontological* and linguistic, respectively [61].

At the time of this deliverable, the concrete syntax of the GeneSIS language is graphical as depicted in Figure 15. In future work, a textual syntax will be considered.

From a deployment model specified using the GeneSIS Modelling language, the GeneSIS deployment engine is responsible for: (i) deploying the software artefacts, (ii) ensuring communication between them, (iii) provisioning cloud resources, and (iv) monitoring the status of the deployment.Details about

the GeneSIS run-time environment can be found in the deliverable D3.1 and will be further presented in D2.2.

3.5.4 How GeneSIS contributes to improving the continuous deployment and trustworthiness of SIS

As detailed before, the objective of the GeneSIS framework is to support the continuous orchestration and deployment of IoT systems whose software building blocks can be deployed over IoT, edge and cloud infrastructures. GeneSIS will thus contribute improving the trustworthiness of SIS in two main ways:

1. By supporting the continuous deployment of SIS, it increases the agility of the overall SIS delivery process, enabling continuous reaction to failures and quick release of patches. This also includes providing developers with the ability to adapt a SIS that would be in a situation that could not have been anticipated during development.
2. GeneSIS will integrate with the ENACT trustworthiness mechanisms and tools (*e.g.*, diversifier, security and privacy control) and must be responsible for the deployment of mechanisms addressing specific aspects of trustworthiness such as security, privacy and resilience. GeneSIS will also integrate with the Actuation conflict management tool, facilitating the deployment of controllers responsible for managing shared access to actuators. In particular, GeneSIS will (i) help identifying direct concurrent accesses to actuators, and (ii) provide specific concepts in the GeneSIS modelling language to represent shared accesses controllers.

Any system defined as an orchestration of deployed software components is exposed to the problem of sharing some resources used to design it. In case of SIS, the involved components are not necessarily stateless (this is the case of actuators for instance) and multiple applications can interact with these shared resources leading to different types of conflicts. The next section will study how to identify, analyse and manage actuation conflicts.

4 Identify, analysing and managing actuation conflicts

IoT applications have been limited to collecting field information for a long time; smart objects utilization is mainly motivated by their capacity at gathering environmental information from sensors as means to support users in decision making (*e.g.*, BUTLER), for enabling the development of secure and smart life assistant applications thanks to contextual pervasive information system. If actions arise, they are carried out under users' control [62, 63]. However, IoT infrastructures must support multiple applications. Many works address this problematic through a three-layers model:

- an infrastructure layer consisting of a set of shared sensor and actuator, immersed in the physical environment;
- a top layer where applications are deployed and,
- one or more intermediate layers for ensuring the overall coherency of the system between applications and the shared infrastructure. The U-Test project introduces, for instance, an "integration" layer for this purpose[34].

The problem is then rather technological or technical and aims at providing the infrastructure layer with sensors access control mechanism [64].

However, SIS are not only composed of sensors, but also of actuators taking electrical input and turning it into physical actions.

[34] U-Test H2020 Project, http://www.u-test.eu/

Therefore, in case of infrastructures with shared actuators, interactions with the physical environment raise additional challenges that cannot be ignored, for example in M2M applications [65]. Actuators not only raise concurrent and possibly conflicting accesses problems. They also raise problems of semantic coherency between the considered actions and their resulting effects in the environment (*e.g.*, opening a window while heating). The consequences of actions and their impacts in the physical environment may put at risk applications functionalities. Indeed, applications being no longer isolated processes, they are not immune to the effects of the concurrent applications sharing the same environment and potentially producing antagonistic effects [66]. The conformity of the applications behaviour with a predefined model is illusory because of the uncertainties pertaining the physical environment, which is open and non-deterministic [67].

In the taxonomy we used in the systematic mapping study and systematic literature review of the orchestration and deployment approaches for SIS (Section 3.1), "actuation" is a subtopic of the "shared access to resources" concept.

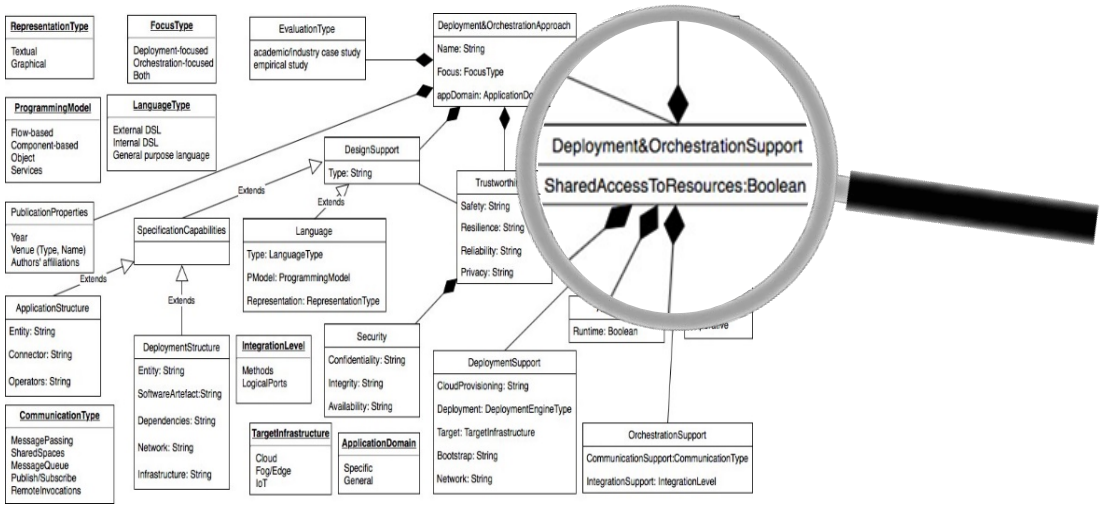

Figure 18 : Shared Access to Resources in SMS/SLR

It results from the systematic mapping study, that actuation is not a first-class concern. However, this study is limited to actuation management in the context of orchestration and deployment for SIS. The only conclusion that can be drawn is that, to the best of our knowledge, there is no mechanisms for managing actuation conflicts in systems for the orchestration and deployment of SIS.

We then conducted a complementary systematic mapping study focusing on "Actuation" in IoT systems (in the broad sense, outside the scope of orchestration and deployment our primary SMS/SLR was limited to). The objective here was to address all the research work done on actuation conflict topic being not necessarily part of a complete orchestration and deployment framework.

The remainder of this section is as follows. In Subsection 4.1, we do present the research landscape on the existing approaches and tools used for supporting the actuation conflict management of IoT systems. Then in Subsection 4.2, we do provide a comprehensive analysis of the existing approaches and identify some important gaps. In Subsection 4.3, we do describe requirements pertaining the ENACT project for supporting actuation conflict management. Finally, Subsection 4.4 provides the baseline technologies and plan for developing our approaches and tools.

4.1 A Systematic Mapping Study (SMS)

The systematic study conducted for the orchestration and deployment of IoT systems highlighted the lack of consideration of the actuation conflict management in the existing solutions. Research community and industry have been proposing different approaches, methods and tools for supporting actuation conflicts identification, analysis and management in IoT and CPS domains. It is not clear

however what are the existing primary approaches for supporting this issue, how advanced they are nor if even they are integrated in existing IoT or CPS systems.

To provide a clear picture of the research landscape on the existing approaches for supporting actuation conflicts management, a Systematic Mapping Study (SMS) has been conducted on actuation conflict management of IoT and CPS systems. Section 4.1.1 gives an overview of the SMS and the research questions addressed along with the review protocol. Some highlights concerning SMS outcomes are provided in section 4.1.2.

4.1.1 Protocol

We conducted this SMS by following guidelines described in section 3.1.1 of this document. Based on the research questions developed in subsection 4.1.1.1, we conducted a rigorous search and selection process presented in subsection 4.1.1.2 providing us with a set of primary studies. Subsection 4.1.1.3 presents the key concepts used for extracting data and analyse it according to the identified research questions.

4.1.1.1 Research questions

The study seeks answers to the following research questions.

RQ1: What are the primary studies statistics?

- *RQ1.1: What is the rate of publication over years?*
- *RQ1.2: In which types of venue (workshop, conference, journal) were the primary studies published?*
- *RQ1.3: What is the distribution of publications in terms of academic and industrial affiliation and location?*
- *RQ1.4: What are the application domains considering actuation conflict management?*

RQ2: What are the approaches implemented in the primary studies and how advanced are they?
RQ2 is split down into the following sub-questions:

- *RQ2.1: How comprehensive the actuation model is?*
  - *RQ2.1.1: Does it consider direct actuation conflict or also indirect ones?*
- *RQ2.2: What are the approaches used to identify actuation conflicts?*
  - *RQ2.2.1: What are the models used to identify actuation conflicts?*
  - *RQ2.2.2: During which stage of the DevOps cycle does the identification conflicts take place (Compile time, Deploy time, Run-time)?*
- *RQ2.3: What are the approaches used to solve the actuation conflicts?*
  - *RQ2.3.1: What are the models and methods underlying actuation conflicts resolution?*
  - *RQ2.3.2: During what stage of the DevOps cycle does the conflicts resolution mechanism take place?*
  - *RQ2.3.3: At which level does the conflicts resolution take place (at device level, edge or in the cloud)?*
- *RQ2.4: What is the readiness level of the proposed approaches?*

RQ3: What are the open issues to be further investigated in this field?

Based on the primary studies, we seek to find out open issues that would deserve more investigation in the future along with the potential directions to tackle them.

- *RQ3.1: What are the open issues of the current research on actuation conflict management?*
- *RQ3.2: What research directions could be envisioned for tackling these issues?*

4.1.1.2 Search and selection process of the primary studies

From the research questions, we derived a set of keywords. To complete these keywords, we also manually selected papers from the domains of concern and automatically extracted the words frequency in the title and abstract of these papers. We classified all of them into three groups of search terms:

- Group of keywords relative to the involved entities: actuator, actuation, effector, feature interaction[35]
- Group of keywords relative to their interaction: conflict, critical, interaction, interference, constraint, detection, shared, concurrency
- Group of keywords relative to the research domains of interest: Internet of Things, IoT, Web of things, WoT, Cyber-Physical Systems, CPS.

The following search string was built as a disjunction of the keywords within each group and a conjunction of all the groups of keywords: *("Internet of Things" OR "IoT" OR "Web of Things" OR "WoT" OR "Cyber-Physical" OR "CPS") AND ("Actuator" OR "Actuation" OR "Effector" OR "Feature Interaction") AND ("Conflict" OR "Critical" OR "Interaction" OR "Interference" OR "Constraint" OR "Detection" OR "Shared" OR "Concurrency").*

Libraries not using metadata-based search (Springer) and indexing non-peer-reviewed paper (Google scholar) were excluded from the automatic search process. This ended up by addressing four main publication repositories (IEEE Xplore[36], ACM DL[37], Scopus[38] and Science Direct[39]) for which the search string was adapted, and results filtered to only make appear the last 10 years publications (IoT research started to emerge in 2008). To complete this automatic search process, we also included the manually selected papers used to complete the keywords list.

More than 2000 candidate papers were identified from automatic and manual search processes from which 22 were selected as a basis for answering the research questions. The fully detailed selection process is depicted in Figure 19.

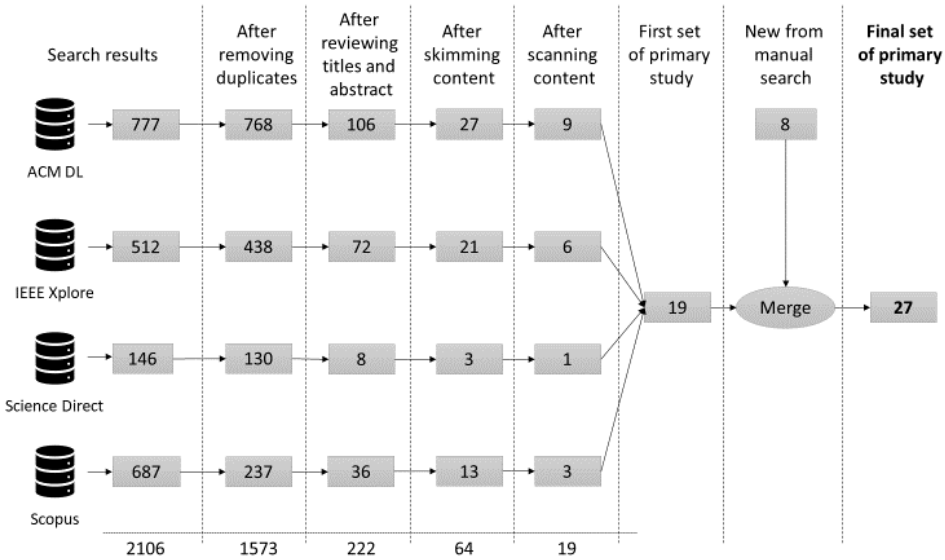

Figure 19: The search and selection process for primary studies

4.1.1.3 Data extraction

A comparison framework based on key concepts has been defined to analyse the primary studies and help us answering the research questions. The key concepts identified are the following: "models

[35] Wikipedia: "Feature interaction is a software engineering concept. It occurs when the integration of two features would modify the behaviour of one or both features."
[36] http://ieeexplore.ieee.org
[37] https://dl.acm.org
[38] https://www.scopus.com
[39] https://www.sciencedirect.com

support", "conflict identification/detection methodology", "resolutions approaches", "application domains", "type of evaluation and application domain". We also included some aspects about the meta-info of the primary studies such as publication year, publication venue to be able to reason about the trends and landscape of research in this domain.

4.1.2 Results Highlights

The following sections are devoted to providing an analysis of the selected publications according to the research questions identified in section 4.1.1.1.

4.1.2.1 Overview

This section provides answers to the research question RQ1 and associated sub-questions RQ1.1, RQ1.2 and RQ1.3. IoT research started to emerge in 2008 but until 2011, there was few papers about actuation conflict management for IoT systems. In Figure 20, although limited, we can note the increasing trend of the publications on the subject over this period. From then, the amount of publications increases, showing more attention to the IoT and CPS actuation conflict research area is given by the community. We conducted our search process in June 2018 which can explain the low number of papers during this year. We would argue that the total number of primary studies published in the whole year 2018 should be at least the same as the previous years. Anyway, the total number of primary studies is not very large.

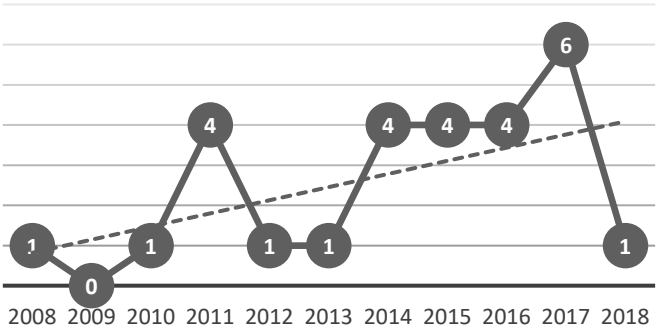

Figure 20:Primary studies published per year

Answering RQ1.2: Figure 21 shows the distribution of primary studies published over years and per venue type. There is a constant number of publications during the period 2014-2016. 30% are journal papers and 55% are conference papers (the rest are workshop papers).

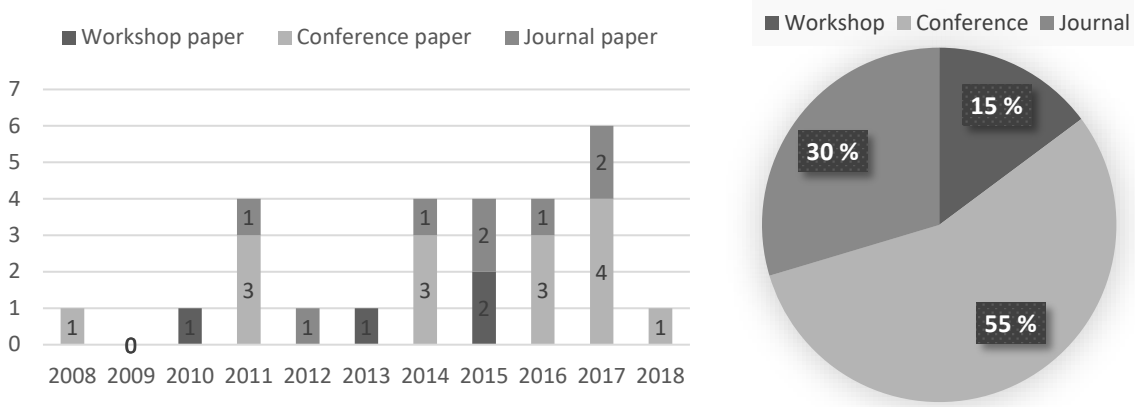

Figure 21: Primary studies published per year, per venue type

Answering research question RQ1.3: by checking the affiliation of the authors, we can see in Figure 22 (left) that most of the authors of the primary studies are researchers (70%). The involvement of industry in this research area is still limited (11%) which is more than the 4% noticed in the Depo4IoT SMS. There are also joint researches between academic and industry (19%). This underline the raising importance of actuation conflict management concern in industrial use cases. Figure 22 (right) shows

the countries researchers come from. The top most countries are United States of America (7), France (4) and Japan (4) followed by Canada (3), Austria (3), Germany (2). Researchers from other countries are also involved in this research for a total of 15 countries.

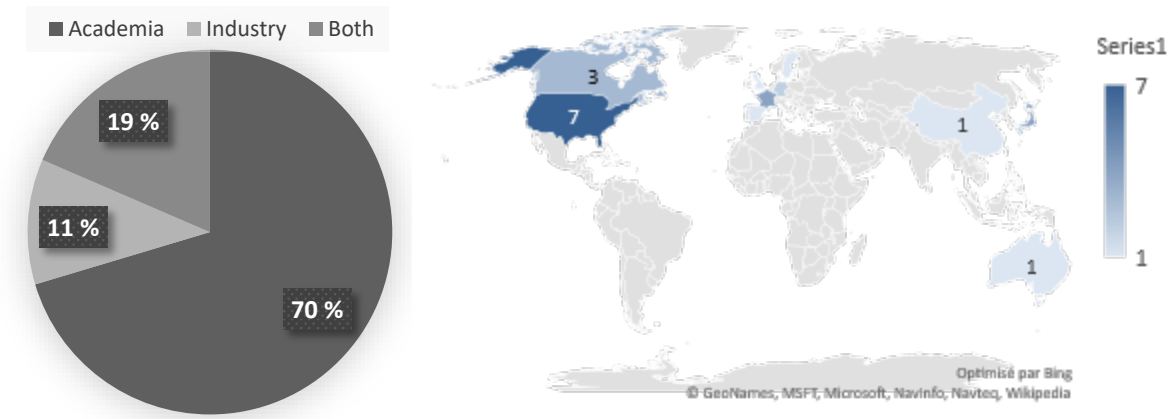

Figure 22: Primary studies per affiliation type (left) and countries (right)

4.1.2.2 Domains and readiness level

Regarding the domains of case studies presented in primary studies, Figure 23 (left) shows the predominance of Cyber-Physical Systems over Internet of Things in terms of application domain. We can also notice that Web of Things (included in our keywords) is not present. One paper speaks about Systems of Systems encompassing CPS and IoT domains. The Smart Home (grouping Smart Home and Smart Office thematic) is more than half of the domains case studies and about one quarter are Automotive case studies. The other ones consider Smart Factory, Smart Health, or Robotics domains and 7% of the primary studies does not apply or illustrate the proposed work with any application domain.

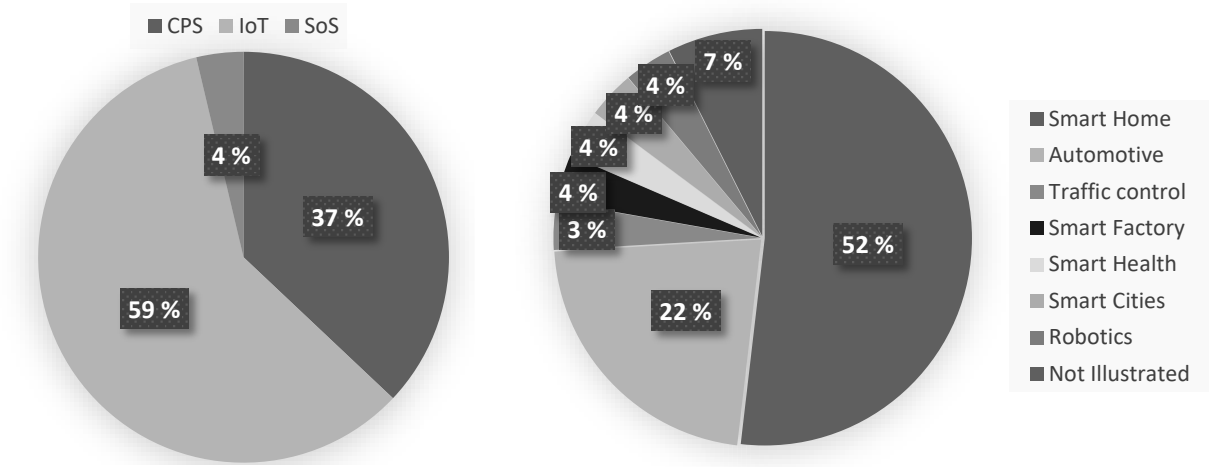

Figure 23: Application domains

Considering the readiness level of primary studies (Figure 24), 74% are limited to approaches than at the maximum are simulation (no experimentation, only theoretical work, use case description of in-silico simulation). 22% of the primary studies present a laboratory prototype and one 1 primary study (4%) made in vivo experimentation.

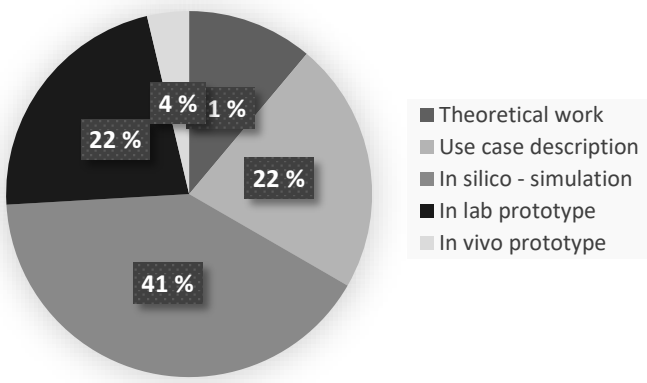

Figure 24:Readiness level

4.2 Analysis and discussion of the state of the art

In the broad sense, a feature describes a unit of functionality that satisfies a requirement [68]. When a global functionality is obtained from a set of shared features, there is a risk for unintended and undesirable interactions between the features. This problem is known as feature interactions (FIs)[69]. This term is general enough to encompass 'conflicts' as being unexpected and unintended interactions between features. Thereby, either one or the other term are used in the literature. FIs concerns are introduced in Subsection 4.2.1. Subsection 4.2.2 presents the modelling technics used for describing FIs. FIs identification methodologies are investigated in Subsection 4.2.3 followed by methodologies for their resolution in Subsection 4.2.4.

4.2.1 Features Interactions (FIs)

FIs concerns have been first identified in the field of telecommunications and subsequently in many other areas such as software composition in information systems [70]. FIs identification and resolution has been widely addressed since then in the literature, proposed solutions relying on state-of-the-art modelling frameworks of the concerned fields. For instance, in software engineering, as part of software product lines (SPLs) methods, feature models describe possible alternative choices for the conception of a software product through FIs dependency and combination rules.

In the Cyber-Physical Systems (CPS) and Internet of Things (IoT) domains, the overall functionality of a system often results in the composition of a set of distributed features (services, actuators, sensors, etc.), relying on Component-Based Software Engineering (CBSE) approaches for representing features and their logical interaction [71]. In this context, FIs is about (1) shared features, possibly subject to conflicting control (structural conflict such that ON and OFF commands are send simultaneously to a lamp), (2) shared environments, possibly subject to conflicting effects (environmental conflict such that a heater is required to warm a room while, at the same time, an air-conditioner is required to cool the same room). Moreover, in these fields one may consider FIs as emergent behaviours not deductible from individual features [69]. Therefore, FIs identification and resolution solutions in this domain not only have to be data oriented as in classical approaches, but have also to be physical effects oriented, taking into account the dynamics of the physical environments within which CPS and IoT systems operate. This is widely recognized in the CPS and IoT literature. Just to name a few, in the smart-home field, authors in [72] emphasis the fact that actuators being entities that affect the surrounding environment, conflicts may occur when applications try to use a single actuator or when they use different actuators causing different effects. In the field of Smart-city applications, the rising number of services increases potential for undesired FIs leading unsafe situations and disruption of the benefits the services were originally intended to provide [73] [69]. Although FIs problem in these fields is recognized, their identification and resolution methodologies are still in their infancy. For instance, authors in [74] find that actuation control other than *ad-hoc* is currently limited in the IoT field due to the lack of mechanisms providing safety guarantees toward FIs. This aspect is crucial as developers cannot objectively anticipate all possible interactions and their consequences, their number being exponential in the number of

features [70]. This situation is aggravated by uncertainties pertaining the non-isolated physical environments CPS and IoT applications interact with.

In the sequel, we do investigate on the modelling frameworks currently used for describing FIs along with associated tools used for identifying and resolving unintended and undesired FIs.

4.2.2 FIs models

In this section, we do investigate on the modelling frameworks currently used in CPS and IoT fields for describing FIs. This analysis provides an answer to the research question RQ2.1. Table 7 provides a summary of the work considered in the study (some papers present evolutions of older approaches. When it is the case, papers are grouped in the same row). It is interesting to notice that only half of the work consider environmental FIs. Among them however, none uses analytical models (*e.g.*, hybrid models) for modelling the physical environment interactions. In [75] however, authors use a model of the environment but this one is extremely limited (considering only speed and distance variables). Instead, (some limited) physical interactions are implicitly defined into the models. FIs are mainly described through rules and dependency graphs (where interactions among a set of features are specified explicitly) or formal modelling techniques hereby allowing automatic verification and detection or preventing undesired interactions to occur. Besides the modelling approaches, it is worth noting that no dedicated development tool for supporting designers in eliciting the FIs models are presented. However, OCL (UML-based), XML/JSON OWL-based SWRL and Finite State Machines (FSM) based models can leverage existing editing tools.

Table 7 : FIs modelling frameworks

	Bib. References	Publication year	FI description	Modelling framework or language	Feature Interactions (FIs) (structural/environmental)
1	An Application Conflict Detection and Resolution System for Smart Homes	2015	Metadata	Kripke structure that represents statuses of sensors and actuators	Structural Environmental
2	DepSys: Dependency Aware Integration of Cyber-Physical Systems for Smart Homes	2014	Metadata	XML	Structural Environmental
3	ECA rules for IoT environment: A case study in safe design	2014	ECA rules	Heptagon/BZR rule system representation	Structural
4	Discrete Control for Smart Environments Through a Generic Finite-State-Models-Based Infrastructure	2014			
5	Safe Composition in Middleware for the Internet of Things	2015	Description Constraint Language (DCL)	Synchronous Mealy machines further transformed to Kripke structures.	Structural Environmental
6	Toward Validated Composition in Component-based Adaptive Middleware	2011			
7	Event management for simultaneous actions in the Internet of Things	2017	Sensors/Actuators Dependency graph	Complex Event Processing (CEP)	Environmental
8	Towards a Model-**Based Verification** Methodology for Complex Swarm Systems	2016	Forbidden actions (FAs)	Object Constraint Language (OCL)	Structural Environmental
9	Knowledge-Aware and Service-Oriented Middleware for deploying pervasive services	2012	JSON metadata	Sensor and Actuator Mapping Description (SAMD) language	Structural
10	S^3: Smart Shadow System for Real World Service and Its Evaluation with Users	2011	Metadata	Universal Service Description Language (USDL) can be regarded as the semantic counterpart to the syntactic WSDL description	Structural
11	SPIRE: Scalable and Unified Platform for Real World IoT Services with Feature Interactions	2016			
12	Toward user-centric feature composition for the Internet of Things	2015	Features actuation control program	Finite State Machine (FSM)	Structural
13	Programming support for distributed optimization and control in cyber-physical Systems	2011	Actuator zones and regions of interest	Not specified.	Structural Environmental

14	Distributed programming framework for fast iterative optimization in networked cyber-physical systems	2014			
15	Feature Interactions Detection and Resolution in Smart Homes Systems	2016	Users' preferences and description of the services as a hierarchy of prioritized atomic features	Not specified	Structural Environmental
16	Taming and optimizing feature interaction in software-intensive automotive systems	2018		Features dependency graphs	Structural Environmental
17	Semantic Web-based policy interaction detection method with rules in smart home for detecting interactions among user policies	2011	Rules for detecting interactions of users policies	Semantic Web Rule Language (SWRL) + SPIDER ontology	Structural Environmental
18	Minimalist qualitative models for model checking cyber-physical feature coordination	2017		Finite State Machine (FSM) + Physical model	Environmental
19	Feature interaction detection in the automotive	2008	Feature design models	STATEFLOW	Structural
20	Continuous variable-specific resolutions of feature interactions	2017	Metadata	Feature Action Language	Structural

4.2.3 FIs Identification/Detection

In this section, we do investigate on the FIs identification methodologies. This analysis provides an answer to the research question RQ2.2. Table 8 provides a summary of the work considered in the study. Most of the approaches rely on classical model checking techniques applied at compilation or deployment phases. Thus, the proposed approaches are mainly based on deterministic models and/or assume that all possible interactions have been identified upfront. This indicates that the proposed approaches are domain dependent, if not *ad-hoc*. Moreover, it raises the question of the scalability of the proposed approaches assuming that the number of all the possible interactions and their consequences is exponential in the number of features [70]. Scalability is considered a first-class concern for three approaches [76-78]. It is not clear however how methodologies involved address this problem for approaches described in [76, 77]. Only two approaches [77, 79] consider the dynamicity of the underlying infrastructure, typical of the IoT field where features availability, when embedded in "things", depends on the mobility thereof.

Table 8 : FIs Identification methodologies

	Bib. References	Publication year	FIs identification methodology	DevOps stage
1	An Application Conflict Detection and Resolution System for Smart Homes	2015	Checking LTL temporal logic formulas that asserts "no two apps are creating conflicting effects at the same location".	Deployment
2	DepSys: Dependency Aware Integration of Cyber-Physical Systems for Smart Homes	2014	Custom, from metadata parsing.	Deployment
3	ECA rules for IoT environment: A case study in safe design	2014	Model checking (Sigali).	Compilation
4	Discrete Control for Smart Environments Through a Generic Finite-State-Models-Based Infrastructure	2014		
5	Safe Composition in Middleware for the Internet of Things	2015	Model checking (CLEM) on parallel composition of synchronous Mealy machines.	Compilation
6	Toward Validated Composition in Component-based Adaptive Middleware	2011		
7	Event management for simultaneous actions in the Internet of Things	2017	ECA rules against data dependency graph.	Run-time
8	Towards a Model-**Based Verification** Methodology for Complex Swarm Systems	2016	Incremental Model checking (not specified), injecting forbidden actions one at a time.	Compilation
9	Knowledge-Aware and Service-Oriented Middleware for deploying pervasive services	2012	Pervasive Business Definition Language (PBDL), with the aim of defining composition rules.	Run-time
10	S^3: Smart Shadow System for Real World Service and Its Evaluation with Users	2011	Not specified.	Run-time
11	SPIRE: Scalable and Unified Platform for Real World IoT Services with Feature Interactions	2016		

12	Toward user-centric feature composition for the Internet of Things	2015	A formal model of the composition algorithm for enumerated settings has been written in Alloy and checked with the Alloy Analyzer ensuring that a feature's current setting (if any) at a priority p and with a timestamp t is in force unless: 1. Some feature has a different current setting at a priority greater than p, or 2. Some feature has a different current setting at priority p, and with time t or greater than t.	Compilation
13	Programming support for distributed optimization and control in cyber-physical Systems	2011	Distributed Shared Memory (DSM) architecture + lock primitives that guarantee serializability. Any read or Write access to shared actuator variables is guarded by explicit lock acquire and release operations	Run-time
14	Distributed programming framework for fast iterative optimization in networked cyber-physical systems	2014		
15	Feature Interactions Detection and Resolution in Smart Homes Systems	2016	Negotiating algorithm.	Run-time
16	Taming and optimizing feature interaction in software-intensive automotive systems	2018	Feature coupling metric.	Compilation
17	Semantic Web-based policy interaction detection method with rules in smart home for detecting interactions among user policies	2011	Reasoning with SWRL	Compilation
18	Minimalist qualitative models for model checking cyber-physical feature coordination	2017	Model checking (Computation Tree Logic (CTL) properties)	Run-time
19	Feature interaction detection in the automotive	2008	SMV model checker	Compilation
20	Continuous variable-specific resolutions of feature interactions	2017	Not specified	Compilation

4.2.4 FIs resolution approaches

In this section, we do investigate on the FIs resolution methodologies. This analysis provides an answer to the research question RQ2.3. Table 9 provides a summary of the work considered in the study. Most of the approaches handle automatic FIs resolution at run-time, relying on controller instances (*e.g.*, Java controller [80, 81], Constraint component [77], Orchestrator [79], Agents [82], Negotiator [83], Coordinator [75, 84]). FIs Resolution methodologies mainly rely on three approaches, given hereafter by order of importance. (1) based on Behaviour-Interaction-Priority (BIP) [77, 80, 81], where FIs are resolved based on prioritization policies; (2) based on formal constraints checking [77, 80, 81, 85]; (3) based on quality of service (QoS) optimization [79]. It is worth noting that BIP approaches do generally not scale well, nor are well accepted by end users [82].

Among the approaches [77-79] considering the dynamicity of the underlying infrastructure, only the approaches detailed in [77, 78] implements a dynamic FIs resolution mechanism.

Finally, considering the whole flow, reusability must be a first-class concern. However: (1) Rules descriptions rely on meta-models being either ontologies [86] or specific description languages. In any case, there is no standard emerging from the approaches studied. (2) FIs resolution controller reusability is not identified as a first-class concern.

Table 9: FIs resolution approaches

	Bib. References			Publication year	Deployment target	DevOps Stage	Automation level	Resolution methodology and underlying model
1	An Application Conflict Detection and Resolution System for Smart Homes	2015	?	Deployment	Manual		It classifies all possible conflicts into a number of situations meaningful to the users, providing helpful information when they prioritise apps	
2	DepSys: Dependency Aware Integration of Cyber-Physical Systems for Smart Homes	2014	Edge	Run-time	Semi-automatic		Prioritization automated from users' preference learning and apps categorization (Semantic Aware Multilevel Equivalence Class based Policy (SAMECP))	
3	ECA rules for IoT environment: A case study in safe design	2014	Edge	Run-time	Automatic		Java controller built from Heptagon/GZR representation. Inconsistency at run-time is	

4	Discrete Control for Smart Environments Through a Generic Finite-State-Models-Based Infrastructure	2014				controlled with the use of different policies. Contradictory signals can be discarded or delayed depending on the chosen policy.
5	Safe Composition in Middleware for the Internet of Things	2015	Edge	Run-time	Automatic	Constraint component with device inhibitor (managing dynamic device appearance/disappearance management)
6	Toward Validated Composition in Component-based Adaptive Middleware	2011				
7	Event management for simultaneous actions in the Internet of Things	2017	Edge	Run-time	Automatic	Extended Parallel rule execution model → Rules are instantiated if not conflicting.
8	Towards a Model-Based **Verification Methodology for** Complex Swarm Systems	2016	Edge	Compilation	Manual	Forbidden actions (FAs) against the behavioural model. Either PASS or FAIL (in which case the model has to be fixed by the designer).
9	Knowledge-Aware and Service-Oriented Middleware for deploying pervasive services	2012	Edge	Run-time	Automatic	The Orchestrator oversees solving concurrency situations when clients compete for shared resources. Response time and throughput optimization-based.
10	S^3: Smart Shadow System for Real World Service and Its Evaluation with Users	2011	Edge	Run-time	Automatic	Field agents (FAs). Resolution policies: 1. Legacy (Last-user-wins) approach 2. First-user-wins approach 3. User Interaction (UI) approach 4. Artificial Intelligence (AI) approach
11	SPIRE: Scalable and Unified Platform for Real World IoT Services with Feature Interactions	2016				
12	Toward user-centric feature composition for the Internet of Things	2015	Edge	Run-time	Automatic	Priority-based with Last-user-wins approach
15	Feature Interactions Detection and Resolution in Smart Homes Systems	2016	IoT/Edge	Run-time	Automatic	Feature usage request is negotiated from a negotiator returning: Accept, Compromise (by taking the middle point between the performance values of two conflicting features), counter proposal, reject.
16	Taming and optimizing feature interaction in software-intensive automotive systems	2018	N/A	Compilation N/A	Semi-automatic N/A	Features coordinator. Hill-climbing optimization algorithm. N/A
17	Semantic Web-based policy interaction detection method with rules in smart home for detecting interactions among user policies	2011				
18	Minimalist qualitative models for model checking cyber-physical feature coordination	2017				Features coordinator.
19	Feature interaction detection in the automotive	2008		Compilation	Manual to semi-automatic	Domain experts can determine an appropriate resolution scheme or provide a correction to the design features, further stored in a knowledge base.
20	Continuous variable-specific resolutions of feature interactions	2017	Edge	Compilation	Manual	The developer is responsible for devising a resolution strategy for each output variable

The study of the state-of-the-art allows to identify the following weaknesses in the current approaches: the low degree of importance neither given to the physical environment modelling as part of the conflicts identification process, nor the reusability, the scalability and the dynamicity as part of the resolution process. These points are first-class concerns in the requirements and conceptual design sections.

4.3 Requirements

In this section, we present requirements associated with the development of the actuation conflict management tools. The distribution of the forecasted tools within the DevOps loop is presented Table 10. Actuation conflict management will be used in the Smart Building (UC3) use case.

Table 10: Actuation conflict management requirements

Req. ID	Req. Type	Comments
UC3 **R1**	Accuracy	Modelling tool for a comprehensive actuation system integrating models of the physical environment and its evolutions according to the interactions with competing applications
UC3 **R2**	Usability	Shall integrate a GUI tool for actuation modelling support, based on simple and interpretable modelling frameworks.
UC3 **R3**	Trustworthiness (reliability)	Shall help identifying complex environmental actuation conflicts
UC3 **R4**	Reusability	Shall permit the reusability of solutions already designed for similar cases
UC3 **R5**	Trustworthiness (reliability)	Shall aid in the design of the conceptual model of the conflicts controller (*e.g.*, Formal test & verification)
UC3 **R6**	Trustworthiness (reliability)	Shall provide tools for testing conflicts controller through an operational model (intended to validate conflict resolution solutions in an operational context)
UC3 **R7**	Trustworthiness (reliability)	Shall manage actuation conflicts despite black box modelled components.
UC3 **R8**	Adequacy	Conflicts resolution at run-time shall be automated as much as possible.
UC3 **R9**	Trustworthiness (safety)	Shall provide actuation conflicts alerts during the deployment on ENACT platform.
UC3 **R10**	Monitoring & trustworthiness (safety)	Shall continuously monitor behavioural drift to assess deployed solutions.
UC3 **R11**	Monitoring & traceability	Shall trig a new development cycle from the quantitative behavioural drift assessment value/threshold.
UC3 **R12**	Scalability	Shall provide tools allowing to manage hundreds of sensors and actuators, thus tens of actuation systems.
UC3 **R13**	Scope	Actuation conflicts management tools shall support several targets ranging from IoT, Edge to cloud.
UC3 **R14**	Integration	Actuation conflicts management tools shall support different kind of frameworks (GeneSIS, ThingML, Node-Red) and middleware (SMOOL, SOFIA2, etc.).

4.4 Actuation Conflict Management Enablers

In this section, we review baseline technologies and detail our plans for developing the approach and tools based on requirements detailed in the previous section.

4.4.1 Actuation Conflict Models

As we are going to see further on, the actuation conflict analysis relies on several models defined during the development and for the deployment of SIS. In particular, three types of models defined during the developments, characterize the issue of managing devices operating within a physical environment (cf. Figure 25):

1. Models of the physical environment describing dependencies between actuator effects that may hamper the deployed applications expected behaviour,
2. Models of the devices characterizing interactions between applications implementing them simultaneously (*i.e.*, SIS orchestration),
3. Functional behavioural model of the devices.

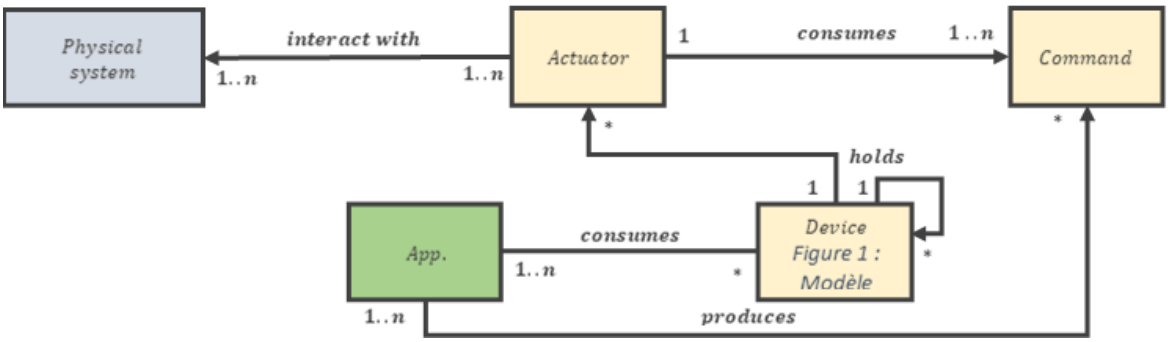

Figure 25: Interaction model between software applications and operationnal contexte for IoT

Furthermore, the following two additional model types are relative to the deployment stages of software applications in ENACT.

- Deployment models implemented in Genesis framework and introduced in the first part of the present deliverable (see Section 3.5.3),
- Implementation models such as ThingML and Node-RED required for the project experimental needs.

4.4.2 Actuation conflict management conceptual design

Actuation conflict management relies on two major components. The first one is dedicated to the identification of the actuation conflicts and the deployment of a known solution (Figure 26). Indeed, in the absence of conflict, the solution can be deployed straight away. Should this not be the case, a solution is sought in a repository of known solutions. This can be achieved by designers through queries to the repository or from a predefined strategy. The controller associated with the solution is then integrated on the point of contention of the service orchestration which then become valid and deployable. The approach emphasizes reusability and, with regards to efficiency, can build upon the experience acquired in the operational phase.

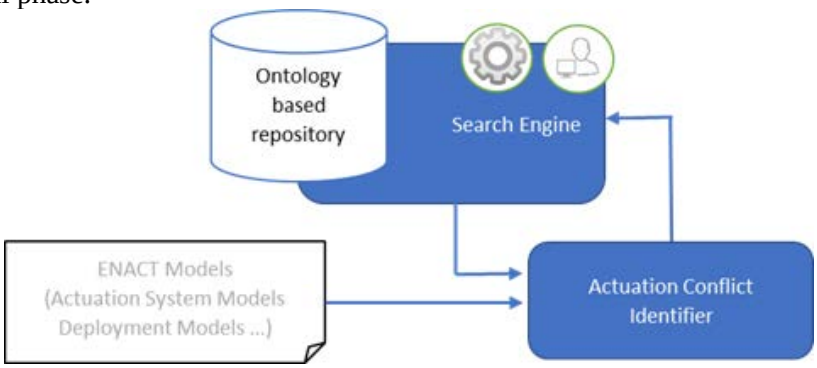

Figure 26: Action conflict identification and solution recovery

In case of unsuccessful query, a solution design tool takes over, supporting different tests and validation levels ranging from the conception to the generation of a new conflict management solution.

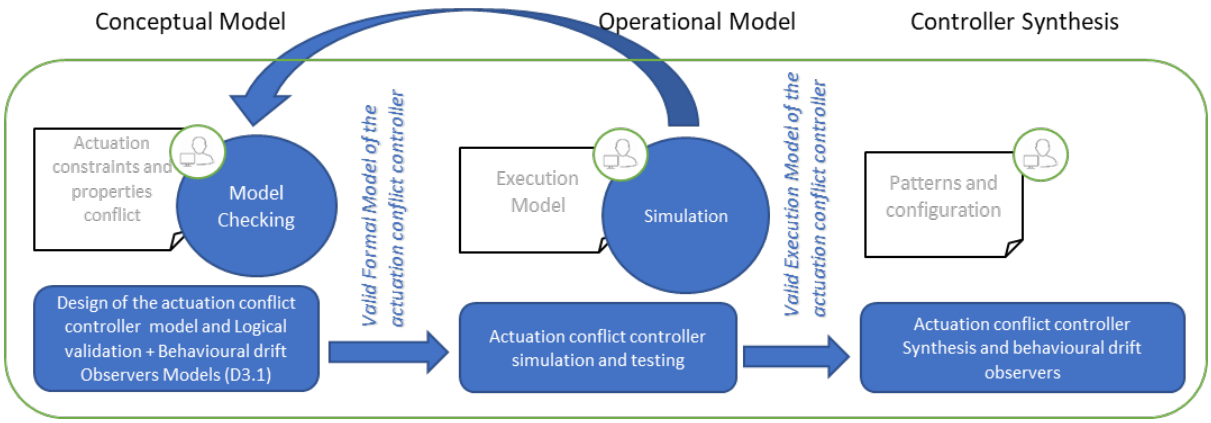

Figure 27: Actuation conflict controller generation flow

The second enabler follows a three stages design process (Figure 27).

The first stage provides assisting tools for designing a conflict resolution solution at a purely logical and conceptual level. This stage relies on formalisms that can be associated with proof tools such as formal model checkers. The CNRS draws on earlier work based on synchronous automata as modelling tool [77].

The second stage aims at integrating the logic of the validated solution into an execution controller according to various models. This will therefore be about operational model, considering asynchronous interactions that may jeopardize results obtained during the first stage. It is worth verifying by simulation if the proposed solution is still valid in its operational model. Here, the CNRS draws on earlier work based on DEVS formalism and associated simulation tools [87].

During the last stage, the validated solution is synthetized according to a pattern fitting the target. For instance, it could be through the generation of a software component at GeneSIS level, one or more Node-RED software components but also through other code adaptation mechanisms at ThingML level such as Aspects.

4.5 How actuation conflict management contributes to trustworthiness in ENACT

Trustworthiness as defined in ENACT addresses the following concepts: Security, Privacy, Resiliency, Reliability and Safety. Among these concepts, the actuation conflict manager addresses Reliability and Safety through the following identified requirements (Table 11). Reliability refers to the ability of the CPS to deliver stable and predictable performance in expected conditions. For instance, while a new feature is introduced in the SIS, it must be guaranteed that its interactions will not jeopardize the stability and predictability of the SIS. 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. For instance, suppose there is door which must be closed in case of fire, while a new device introduced to the system also has a control to the door, then we need to ensure that the door will always be kept closed in case of fire. Otherwise, there is a safety issue.

Table 11: Requirements addressing trustworthiness concern

R3	Trustworthiness (reliability)	Shall help identifying complex environmental actuation conflicts
R5	Trustworthiness (reliability)	Shall aid in the design of the conceptual model of the conflicts controller (*e.g.*, Formal test & verification)
R6	Trustworthiness (reliability)	Shall provide tools for testing conflicts controller through an operational model (intended to validate conflict resolution solutions in an operational context)
R7	Trustworthiness (reliability)	Shall manage actuation conflicts despite black box modelled components.

R9	Trustworthiness (safety)	Shall provide actuation conflicts alerts during the deployment on ENACT platform.

5 Test, simulation and emulation Services for SIS

The last stage of the development cycle of the DevOps approach, before the operational part, is the test, simulation or emulation of the system to be deployed. This section presents the challenges and requirements associated to this stage.

5.1 IoT Simulation and Testing

Software testing is a crucial step in the software development process. However, having access to a production-like environment that reproduces the same condition where a piece of software would run is usually tricky. Still, developers need to test their applications to ensure its minimum faultiness.

The issue of not having a proper testing environment for applications is particularly challenging in the IoT arena. The access to devices might not be trivial, or it can be limited due to many factors. Networks of physical deployed devices are typically devoted to production software, and testing applications on top of those networks might involve additional testing software, which might affect overall performance —and hence the revenue generated by the system— of the devices if, for instance, they need to be stopped to load the new versions of applications.

Software simulators proved to be valuable in filling this gap, providing developers a testing environment to —at least— start testing their applications. When it comes to IoT application testing, simulation tools allow developers to have an initial testing platform that enables them to develop their applications before putting them into a production IoT devices network. This way, the impact of the application development on IoT systems is minimized.

IoT Testbeds also play a relevant role when it comes to testing applications. Testbeds offer a deployed network of IoT devices. Developers can upload their applications onto these networks and test their software in a real environment. IoT-Lab[40] and SmartSantander[41] are good examples of IoT testbeds. Testbeds often have a predefined fix configuration and architecture. They are also usually shared with other users, which can be a problem when it comes to measuring application performance. Hence, the main drawbacks of the testbed approach can make simulators more attractive, since they can provide a more custom environment and more control over it. Furthermore, simulators avoid the need of having a physical network of devices.

In the recent years, both academia and the commercial market offered solutions in the IoT simulation field. Although the area is the same, their approach is entirely different. Academic solutions implement cutting-edge technology in the form of proofs-of-concept, which are usually not ready for production systems demands. On the other hand, commercial solutions focus on producing a stable and flawless solution, even though the technology behind might not be at the cutting-edge state-of-the-art.

5.1.1 Challenges in IoT simulation and testing

IoT creates a unique scenario where several challenges are met. The most common challenges identified by the literature are listed in this section. Ideally, products that deal with IoT simulation and testing should take into consideration these set of ideas during its design stage.

- **Low energy consumption.** IoT devices are very heterogeneous, which leads to a wide range of applications for these devices. These applications might involve setting up devices in inaccessible locations, where, for instance, a power supply is not guaranteed, and batteries are needed. Software running on an IoT device should be optimized concerning power

[40] https://www.iot-lab.info/
[41] http://www.smartsantander.eu/

consumption, and protocols need to be adapted to lightweight implementations. Simulation tooling must be ready for this low power scene, dealing with lightweight protocols, and the simulation of a power supply layer.

- **Connectivity problems.** Device connectivity must be ensured for the system to work as expected. Network signal quality in mobile networks may fluctuate in such a way that devices can lose connectivity in particular locations, affecting the overall performance.
- **Scalability.** Networks of IoT devices can be large. The physical scope is quite variable. From configurations that take one floor of a building to an entire city, IoT presents a unique scenario regarding variability. Simulation tools must support both small and vast networks. For that matter, implementations should rely on parallel or distributed [88] architectures to meet the scalability requirements.
- **Lack of standardization.** Emergent fields tend to suffer the problem of finding a common language that every player can understand and that fulfils the requirements of the field itself. IoT is not an exception to that matter. The volatility of the implementations and the immaturity of the field makes difficult to generate synergies to leverage different tools and make them work together.
- **Security and privacy.** Security concerns come up [89, 90] since IoT systems can be widely spread within a vast geographical region, and devices can remain exposed. Private network communications must be ensured in a lightweight fashion to fulfil the low energy consumption requirement.

5.1.2 Academic approach to IoT simulation and testing

In the recent years, IoT gained focus in academia. The number of publications related to IoT has increased from 2012 until now [-]. Within the IoT scope, the subfield of simulation and testing in IoT also has gained momentum when it comes to generating novel, cutting-edge ideas.

In 2017, academia listed several IoT simulators produced directly out of research efforts. These simulators usually focus on a particular layer of a stack. For instance, Cooja [91] and OMNeT++ [92] focus on simulating networking aspects of the systems. Other simulators, like SimIOT [93] or IOTsim [94], focus on data analytics rather than lower aspects of the systems. Another approach, like iFogSim [95] try to perform a complete simulation. However, having a full stack simulation from a single component or product can be challenging. Therefore, academia started several efforts in hybrid models [96], which try to leverage several simulators of particular single layers to reproduce the behaviour of a system from a holistic perspective.

Research-based simulators often exude the problems such as, the lack of standardization of the systems, which poses a challenge when it comes to creating synergies and interoperation between different simulators. A research-based simulator can be volatile and can change its application interface rapidly. This volatility generates extra overhead since developers need to adapt their code to the new changes. On the other hand, simulators created by research, often without economic support, may become stable, as such, as they appear unmaintained or discontinued. Bugs often remain, and new features or improvements might not be added. On the other hand, these simulators are often open source, which facilitates that community contributes to its development and maintenance.

5.1.3 Industry approach to IoT simulation and testing

Commercial needs are far from the academic world. Production ready software needs to be stable, usable, and robust. The industry needs resilient software in which users and companies can rely upon without dealing with many difficulties. From this perspective, current IoT simulators for production systems focus on these aspects, rather than implementing the latest novelties in the field.

In the Cloud scene, big players offer several solutions for IoT simulation, such as Microsoft with Azure IoT[42] or Amazon with AWS IoT[43]. KnowThings.io[44], a company from CA Accelerator, models the behaviour of actual devices to reproduce them later on at scale. In a more general way, MATLAB[45] is also used to carry out simulation tasks. Iotify[46] provides services for simulating devices at scale.

IoT applications often need to consider volatile and growing environments hence the primary focus of commercial solutions are the scalability issues. However, industry solutions also face challenges. Since simulators often describe successful scenarios, they usually fail to simulate unexpected events or failures. Cyber-attacks are not usually taken into account when it comes to simulation. Moreover, lack of an emerged standard in IoT means that the interaction between simulators from different providers or different abstraction levels is not as straightforward as it should. Multiple efforts of providing middleware to ease such deployments have been made —ThingML [97] or Sofia2[47].

5.2 Addressing use cases

This section describes the requirements that Test, Simulation and Emulation Enabler should implement to fulfil the Use Cases' needs.

5.2.1 Requirements

The use cases that required simulation facilities are the Intelligent Transportation System use case, referred in this section as UC1, and the Digital Health use case, referred in this section as UC2. The following table introduces a set of high-level requirements for the Simulation enabler derived from the descriptions of the case studies.

Table 12: Use cases requirements list

ReqID	Requirement	Description
UC1 **R1**	Scalability	The simulator should easily scale the number of components that are involved in the simulation.
UC1 **R2**	Component modelling	The simulator should model virtual devices that reproduce the behaviour of real devices.
UC1 **R3**	Simulation of multiple sensor events	The simulator should generate signals from multiple types of sensors: – Accelerometer: ADXL362Z, SPI – GNSS: A2035H, UART – XBEE radio for RSSI: Xbee868LP, SPI – RFID: SparkFun Simultaneous RFID Reader - M6E Nano, UART – Battery monitoring (load current, battery voltage)-analog voltage on ADC inputs of energy harvesting module internal in EDI node (controller). – Analog measurement circuits, including current-voltage converter and instrumentation amplifier circuits is being developed by BOSC. Particular IC is not chosen yet.
UC1 **R4**	Simulation of multiple communication protocols	The simulator should simulate several communication protocols: – IEEE 802.15.4 ZigBee – IEEE 802.3
UC1 **R5**	Simulation of dynamic geographical position	The simulator should simulate changes of geographical position of the system, taking into account possible mobile network disconnections, and other possible situations derived of the position changes of the system physical platform.

[42] Iot hub - microsoft azure. [Online]. Available: https://azure.microsoft.com/es- es/services/iot-hub/
[43] Aws iot. [Online]. Available: https://aws.amazon.com/iot/
[44] Knowthings.io - self learning iot virtualizer. [Online]. Available: https://knowthings.io/
[45] Matlab - internet of things. [Online]. Available: https://www.mathworks.com/solutions/internet-of-things.html
[46] Iotify. Available: https://iotify.io/
[47] The sofia2 project, smart & cloud architectures and platforms, 2017. [Online]. Available: http://sofia2.com/

UC1 **R6**	Failures simulation	The simulator should simulate the possible failures of the system. Failures can be related with networking issues, device disconnection, or fake readings.
UC1 **R7**	Attack simulation	The simulator should generate possible attacks to the system. Attacks include data poisoning, device disconnection, or device hijacking.
UC2 **R1**	Scalability	The simulator should easily scale the number of components that are involved in the simulation.
UC2 **R3**	Simulation of multiple sensor events	The simulator should generate signals from multiple types of sensors listed in D1.1
UC2 **R4**	Simulation of multiple communication protocols	The simulator should simulate several communication protocols as described in D1.1
UC2 **R6**	Failures simulation	The simulator should simulate the possible failures of the system. Failures can be related with networking issues, device disconnection, or fake readings.
UC2 **R7**	Attack simulation	The simulator should generate possible attacks to the system. Attacks include data poisoning, device disconnection, or device hijacking.
UC2 **R8**	Real environment interoperability	The simulation should be able to be plugged into a real system so that it can interact as a real part of it.
UC2 **R9**	External actors simulation	The simulation should simulate the interaction of external actors. An external actor can be a human being or another system.

5.3 Testing and Simulation Conceptual Design

From a high-level perspective, Testing and Simulation enabler is envisioned as a package that comprises two main differentiated modules.

First module handles the Simulation itself, extracting the behaviour of actual devices and modelling it to create virtual devices that later on will be played back. Architectural information of the system to be simulated can be passed as input so the simulator will know what devices need to point to sniff its traces and extract its behaviour.

On parallel, the second main module will manage Event Generation. This module will provide most of the value related to trustworthiness. It will simulate both malicious and non-malicious events that could modify the normal operation of applications or systems. It will enable developers to anticipate potential issues and allow them to build a more resilient and robust applications, as well as to anticipate security problems regarding cyberattacks.

During simulation, developers will be able to monitor the simulation and to evaluate the results for testing once the simulation is finished. To that matter, developers will upload a description of the alerts and tests to assess during the simulation.

5.4 Addressing trustworthiness through Testing and Simulation

We define trustworthiness as the conjunction of the concepts of Security, Privacy, Resiliency, Reliability, and Safety. Applications involving IoT typically deal with critical systems. Hence it is required for IoT solutions to fulfil the trustworthiness principles. This section explores the Testing and Simulation related literature, examining how the different ideas and solutions related to Testing and Simulation in IoT address these issues, and then how the Testing, Simulation, and Emulation Enabler deals with them.

IoT application development processes must ensure that the applications meet trustworthiness. Software testing is a crucial step to fulfil this requirement. Besides guaranteeing that the application does what users expect, proper testing enables developers to find application bugs and to thoroughly detect and address issues before taking the application into production, where dealing with application malfunctioning becomes more critical.

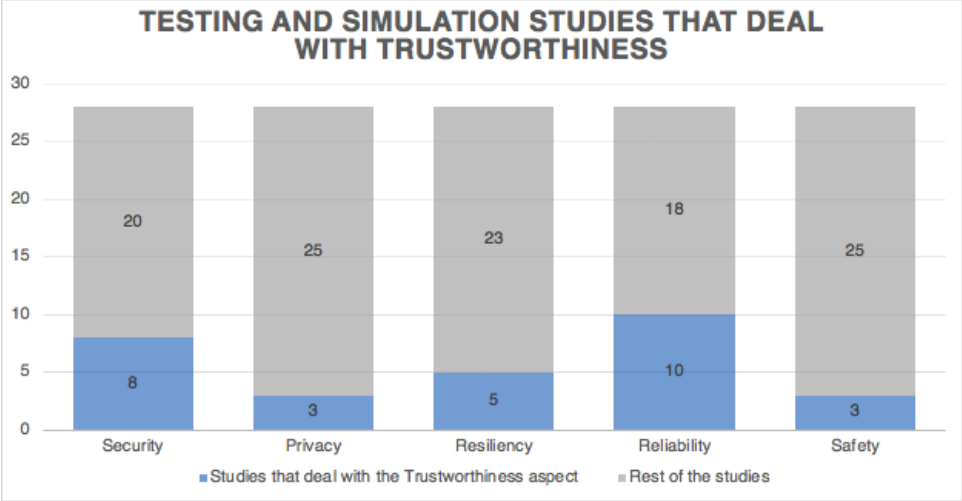

Figure 28: Number of IoT simulation related sources that consider trustworthiness aspects

Figure 28 shows that very few studies [88, 89, 93-96, 98-118] consider trustworthiness aspects. In particular, 8 out of 28 studies address Security, 3 of them Privacy, 5 of them Resiliency, 10 of them Reliability, and 3 of them Safety. These numbers show an apparent gap in IoT application testing when it comes to ensuring trustworthiness.

Considering the conceptual design described previously, the most relevant ENACT's contribution to the field of IoT simulation concerning Trustworthiness is the Event Generator module. This module will run on top of the actual device simulator. It will inject either malicious (*e.g.*, cyberattacks) and non-malicious (*e.g.*, network disconnections) situations into a running simulation. It will allow developers to anticipate potential unexpected scenarios (resiliency) that the application may face, as well as ensuring that the application securely keeps its data (security and privacy). Reliability aspect will be ensured by the actual simulation of the devices. As seen, the ENACT's Simulation Enabler can contribute to most of the Trustworthiness aspects.

6 Conclusion

WP2 aims at providing enablers for the development and deployment of SIS. WP2 will provide enablers with capabilities to (i) manage risk, (ii) orchestrate and deploy software components, (iii) identify, analyse and manage actuation conflict, and (iv) test, simulate and emulate provided services.

The purpose of this deliverable was to conduct and analyse state-of-the-art on the four aforementioned topics. To achieve this, two systematic mapping study and one systematic literature review have been conducted. For each topic, a rigorous analysis of the state-of-the-art was conducted resulting in (i) the specification of the requirements, (ii) the identification of the enablers and potential gaps with the requirements.

Thus, WP2 will provide the following enablers: a risk management service, a secure framework to orchestrate and deploy SIS, an actuation conflict manager and services to test, simulate, and emulate SIS (Figure 29).

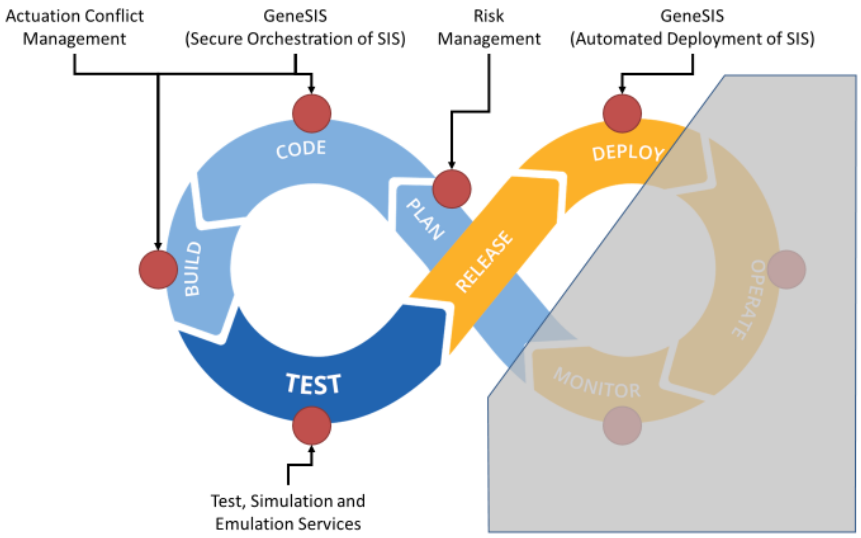

Figure 29: WP2 Enablers for the Dev part of the DevOps cycle

The next steps will be devoted to implement these enablers, with the help of the defined conceptual design of each enabler, keeping in mind the defined requirements.

Appendix A A systematic mapping study of deployment or orchestration approaches for IoT

All the details are in the technical report SMS_Depo4IoT.pdf, which is attached with this deliverable. Table 13 gives the full list of the primary studies of the SMS.

Table 13. The list of the primary Depo4IoT studies in the SMS

#	***Title****	***Year***	v	f
1	Challenges and Solutions in Fog Computing Orchestration	2018	J	O
2	Deploying Edge Computing Nodes for Large-scale IoT: A Diversity Aware Approach	2018	J	D
3	Cloud-Fog Interoperability in IoT-enabled Healthcare Solutions	2018	C	D
4	A visual programming framework for distributed Internet of Things centric complex event processing	2018	J	B
5	Enhancing Middleware-based IoT Applications through Run-Time Pluggable QoS Management Mechanism	2018	C	D
6	A Dynamic Module Deployment Framework for M2M Platforms	2017	C	D
7	A Middleware for Mobile Edge Computing	2017	J	B
8	A service orchestration architecture for Fog-enabled infrastructures	2017	C	O
9	Distributed Orchestration in Large-Scale IoT Systems	2017	C	O
10	Internet of Things: From Small- to Large-Scale Orchestration	2017	C	O
11	QoS-Aware Deployment of IoT Applications Through the Fog	2017	J	D
12	Service Orchestration in Fog Environments	2017	C	O
13	Towards Container Orchestration in Fog Computing Infrastructures	2017	C	O
14	A Framework based on SDN and Containers for Dynamic Service Chains on IoT Gateways	2017	W	D
15	A framework for MDE of IoT-Based Manufacturing CPS	2017	C	O
16	A Novel Service-Oriented Platform for the Internet of Things	2017	C	O
17	Design and Implementation of a Message-Service Oriented Middleware for Fog of Things Platforms	2017	C	O
18	Empowering End Users to Customize their Smart Environments	2017	J	O
19	Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi	2017	C	B
20	Semantics Based Service Orchestration in IoT	2017	C	O
21	An Object-Oriented Model for Object Orchestration	2017	C	B
22	A TOSCA-based Programming Model for Interacting Components of Automatically Deployed Cloud and IoT Applications	2017	C	B

23	An edge-based platform for dynamic Smart City applications	2017	J	D
24	Calvin Constrained: A Framework for IoT Applications in Heterogeneous Environments	2017	C	B
25	InterCloud Communication Through Gatekeepers to Support IoT Service Interaction in the Arrowhead Framework	2017	J	O
26	Internet of things out of the box Using TOSCA for automating the deployment of IoT environments	2017	C	B
27	Platform-as-a-service gateway for the Fog of Things	2017	J	B
28	Runtime deployment and management of CoAP resources for the internet of things	2017	J	D
29	Composing Continuous Services in a CoAP-based IoT	2017	C	B
30	Niflheim: An end-to-end middleware for applications on a multi-tier IoT infrastructure	2017	C	B
31	Foggy- A Framework for Continuous Automated IoT Application Deployment in Fog Computing	2017	C	D
32	A Web of Things Based Device-Adaptive Service Composition Framework	2016	C	O
33	Application Orchestration in Mobile Edge Cloud: Placing of IoT Applications to the Edge	2016	W	O
34	Optimizing Elastic IoT Application Deployments	2016	J	D
35	FRED- A Hosted Data Flow Platform for the IoT built using NodeRED	2016	W	B
36	Incremental deployment and migration of geo-distributed situation awareness applications in the fog	2016	C	D
37	On Building Smart City IoT Applications- a Coordination-based Perspective	2016	W	B
38	Orchestrating the Internet of Things Dynamically	2016	W	B
39	SoIoT: Toward A User-Centric IoT-Based Service Framework	2016	J	O
40	A Container-based Edge Cloud PaaS Architecture-based on Raspberry Pi Clusters	2016	C	B
41	Automated Deployment of SmartX IoT-Cloud Services based on Continuous Integration	2016	C	D
42	Cloud4IoT: A Heterogeneous, Distributed and Autonomic Cloud Platform for the IoT	2016	C	B
43	Reliable services composition method for the internet of thing using directed service-object graph deployment scheme	2016	C	O
44	Integration of Heterogeneous Services and Things into Choreographies	2016	W	O
45	A Scalable Framework for Provisioning Large-Scale IoT Deployments	2016	J	D
46	A Data-Centric Framework for Development and Deployment of Internet of Things Applications In Clouds	2015	C	D
47	Towards a Semantic Model for Automated Deployment of IoT Services across Platforms	2015	C	D
48	A component based approach for the Web of Things	2015	W	O
49	A Generic Service Oriented Software Platform to Design Ambient Intelligent Systems	2015	C	O
50	Developing IoT Applications in the Fog: a Distributed Dataflow Approach	2015	C	B
51	A Full End-to-End Platform as a Service for Smart City Applications	2014	W	B
52	A Novel Deployment Scheme for Green Internet of Things	2014	J	D
53	glue.things - a Mashup Platform for wiring the Internet of Things with the Internet of Services	2014	W	B
54	Toward a Distributed Data Flow Platform for the Web of Things	2014	W	O
55	BeC3: Behaviour Crowd Centric Composition for IoT applications	2014	J	B
56	Dioptase: a distributed data streaming middleware for the future web of things	2014	J	B
57	Application deployment for IoT: An infrastructure approach	2013	C	B
58	Orchestration in distributed web-of-objects for creation of user-centered iot service capability	2013	C	O
59	Towards Automated IoT Application Deployment by a Cloud-Based Approach	2013	C	D
60	Mobile Fog: A Programming Model for Large-Scale Applications on the Internet of Things	2013	C	D

61	Gateway as a service- A cloud computing framework for web of things	2012	C	O
62	Behaviour-Aware Compositions of Things	2012	C	O
63	Knowledge-Aware and Service-Oriented Middleware for deploying	2012	J	B
64	Mashing up the Internet of Things: A framework for smart environments	2012	J	O
65	D-LITE: Distributed logic for internet of things services	2011	C	B
66	Adaptable Service Composition for Very-Large-Scale IoT Systems	2011	W	O
67	INOX: A managed service platform for inter-connected smart objects	2011	W	B
68	Connecting Smart Things through Web Services Orchestrations	2010	W	O
69	COSMOS: a middleware for integrated data processing over heterogeneous sensor networks	2008	J	O

[v] Venue type: J = Journal (19 papers), C = Conference (37 papers), W = Workshop (13 papers).
[f] Focus: D = Deployment, O = Orchestration, B = Both Deployment and Orchestration.
* The titles are clickable to link to the corresponding publications

Appendix B Advances in deployment and orchestration approaches for IoT -A systematic review

All the details are in the technical report SLR_Depo4IoT.pdf, which is attached with this deliverable.

References

1. Edward R. Griffor, C.G., David A. Wollman, Martin J. Burns, *Framework for Cyber-Physical Systems: Volume 1, Overview*. June 26, 2017, National Institute of Standards and Technology.
2. ISO/IEC, *27000: 2012, Information technology - Security techniques - Information security management systems - Overview and vocabulary*. 2012, International Organization for Standardization.
3. Bishop, M., *Computer security: art and science*. 2003: Addison-Wesley Professional.
4. ISO/IEC, *29100.2011 Information technology—Security techniques—Privacy framework*. 2011, International Organization for Standardization.
5. Wallmüller, E., *Risk management for IT and software projects*, in *Business continuity*. 2002, Springer. p. 165-178.
6. Rao, L.M. and S. Firdose, *Study of Existing Risk Management Models and Prior Research Contribution.* Adarsh Journal of Information Technology, 2016. **4**(1): p. 10-20.
7. Karabacak, B. and I. Sogukpinar, *ISRAM: information security risk analysis method.* Computers & Security, 2005. **24**(2): p. 147-159.
8. Alberts, C.J. and A. Dorofee, *Managing information security risks: the OCTAVE approach*. 2002: Addison-Wesley Longman Publishing Co., Inc.
9. Lund, M.S., B. Solhaug, and K. Stølen, *Model-driven risk analysis: the CORAS approach*. 2010: Springer Science & Business Media.
10. Howard, M. and S. Lipner, *The security development lifecycle*. Vol. 8. 2006: Microsoft Press Redmond.
11. Merkow, M. and L. Raghavan, *An ecosystem for continuously secure application software*. CrossTalk, March/April, 2011.
12. Khoo, B. *RFID as an Enabler of the Internet of Things: Issues of Security and Privacy*. in *Internet of Things (iThings/CPSCom), 2011 international conference on and 4th international conference on cyber, physical and social computing*. 2011. IEEE.
13. Williams, R., et al. *Identifying vulnerabilities of consumer Internet of Things (IoT) devices: A scalable approach*. in *Intelligence and Security Informatics (ISI), 2017 IEEE International Conference on*. 2017. IEEE.
14. Mohsin, M., et al., *IoTRiskAnalyzer: A Probabilistic Model Checking Based Framework for Formal Risk Analytics of the Internet of Things.* IEEE Access, 2017. **5**: p. 5494-5505.
15. Atlam, H., et al., *An overview of risk estimation techniques in risk-based access control for the internet of things.* 2017.
16. Liu, C., et al. *Research on Dynamical Security Risk Assessment for the Internet of Things inspired by immunology*. in *Natural Computation (ICNC), 2012 Eighth International Conference on*. 2012. IEEE.
17. Hong, J., et al. *Don't Talk Unless I Say So! Securing the Internet of Things with Default-Off Networking*. in *Internet-of-Things Design and Implementation (IoTDI), 2018 IEEE/ACM Third International Conference on*. 2018. IEEE.
18. S. Konig, S.S., M. Latzenhofer, P. Mayer, and G. Fitzpatrick, *Internet of Things Risk Analysis and Assessment.* 2018: p. 1-75.
19. Yan, Z., P. Zhang, and A.V. Vasilakos, *A survey on trust management for Internet of Things.* Journal of network and computer applications, 2014. **42**: p. 120-134.
20. Fitzgerald, B. and K.-J. Stol, *Continuous software engineering: A roadmap and agenda.* Journal of Systems and Software, 2017. **123**: p. 176-189.
21. Womack, J.P. and D.T. Jones, *Lean thinking—banish waste and create wealth in your corporation, revised and updated.* Journal of the Operational Research Society, 2003. **48**(11): p. 1148-1148.
22. Ries, E., *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. 2011: Crown Books.
23. Rogers, E.M., *Diffusion of innovations*. 2010: Simon and Schuster.
24. Carzaniga, A., et al., *A characterization framework for software deployment technologies*. 1998, COLORADO STATE UNIV FORT COLLINS DEPT OF COMPUTER SCIENCE.

25. Dearie, A. *Software deployment, past, present and future*. in *Future of Software Engineering, 2007. FOSE'07*. 2007. IEEE.
26. Fitzgerald, B. and K.-J. Stol. *Continuous software engineering and beyond: trends and challenges*. in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. 2014. ACM.
27. Petersen, K., S. Vakkalanka, and L. Kuzniarz, *Guidelines for conducting systematic mapping studies in software engineering: An update.* Information and Software Technology, 2015. **64**: p. 1-18.
28. Kitchenham, B., *Guidelines for performing systematic literature reviews in software engineering.* EBSE Technical Report, 2007.
29. Kitchenham, B.A., D. Budgen, and O.P. Brereton, *Using mapping studies as the basis for further research–a participant-observer case study.* Information and Software Technology, 2011. **53**(6): p. 638-651.
30. Bergmayr, A., et al., *A Systematic Review of Cloud Modeling Languages.* ACM Computing Surveys (CSUR), 2018. **51**(1): p. 22.
31. Eugster, P.T., et al., *The many faces of publish/subscribe.* ACM computing surveys (CSUR), 2003. **35**(2): p. 114-131.
32. Brogi, A. and S. Forti, *QoS-aware deployment of IoT applications through the fog.* IEEE Internet of Things Journal, 2017. **4**(5): p. 1185-1192.
33. Carrega, A., et al., *A middleware for mobile edge computing.* IEEE Cloud Computing, 2017. **4**(4): p. 26-37.
34. Chen, B.-L., et al. *A Dynamic Module Deployment Framework for M2M Platforms*. in *Cloud and Service Computing (SC2), 2017 IEEE 7th International Symposium on*. 2017. IEEE.
35. Lee, H., et al. *A novel service-oriented platform for the internet of things*. in *Proceedings of the Seventh International Conference on the Internet of Things*. 2017. ACM.
36. Mehta, A., et al. *Calvin Constrained—A Framework for IoT Applications in Heterogeneous Environments*. in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. 2017. IEEE.
37. Small, N., et al. *Niflheim: An end-to-end middleware for applications on a multi-tier IoT infrastructure*. in *Network Computing and Applications (NCA), 2017 IEEE 16th International Symposium on*. 2017. IEEE.
38. Verba, N., et al., *Platform as a service gateway for the Fog of Things.* Advanced Engineering Informatics, 2017. **33**: p. 243-257.
39. Yigitoglu, E., et al. *Foggy: a framework for continuous automated IoT application deployment in fog computing*. in *2017 IEEE International Conference on AI & Mobile Services (AIMS)*. 2017. IEEE.
40. Zimmermann, M., U. Breitenbücher, and F. Leymann. *A TOSCA-based programming model for interacting components of automatically deployed cloud and IoT applications*. in *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS)*. 2017.
41. Pizzolli, D., et al. *Cloud4iot: A heterogeneous, distributed and autonomic cloud platform for the iot*. in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. 2016. IEEE.
42. Giang, N.K., et al. *Developing IoT applications in the fog: a distributed dataflow approach*. in *Internet of Things (IOT), 2015 5th International Conference on the*. 2015. IEEE.
43. Lavirotte, S., et al. *A generic service oriented software platform to design ambient intelligent systems*. in *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. 2015. ACM.
44. Ruppen, A., et al. *A component based approach for the Web of Things*. in *Proceedings of the 6th International Workshop on the Web of Things*. 2015. ACM.
45. Cherrier, S., et al., *BeC 3: behaviour crowd centric composition for IoT applications.* Mobile Networks and Applications, 2014. **19**(1): p. 18-32.
46. Kleinfeld, R., et al. *glue. things: a Mashup Platform for wiring the Internet of Things with the Internet of Services*. in *Proceedings of the 5th International Workshop on Web of Things*. 2014. ACM.

47. Distefano, S., G. Merlino, and A. Puliafito. *Application deployment for IoT: An infrastructure approach*. in *Global Communications Conference (GLOBECOM), 2013 IEEE*. 2013. IEEE.
48. Cherrier, S., et al. *D-lite: Distributed logic for internet of things services*. in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*. 2011. IEEE.
49. Breitenbücher, U., et al. *Combining declarative and imperative cloud application provisioning based on TOSCA*. in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. 2014. IEEE.
50. Petcu, D. *Multi-Cloud: expectations and current approaches*. in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. 2013. ACM.
51. Ferry, N., et al., *CloudMF: Model-Driven Management of Multi-Cloud Applications.* ACM Transactions on Internet Technology (TOIT), 2018. **18**(2): p. 16.
52. da Silva, A.C.F., 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*. 2016. ACM.
53. Schmidt, D.C., *Model-driven engineering.* COMPUTER-IEEE COMPUTER SOCIETY-, 2006. **39**(2): p. 25.
54. Palma, D. and T. Spatzier, *Topology and orchestration specification for cloud applications (TOSCA).* Organization for the Advancement of Structured Information Standards (OASIS), Tech. Rep, 2013.
55. Van Ommering, R., et al., *The Koala component model for consumer electronics software.* Computer, 2000. **33**(3): p. 78-85.
56. Fassino, J.-P., et al. *Think: A Software Framework for Component-based Operating System Kernels*. in *USENIX Annual Technical Conference, General Track*. 2002.
57. Fleurey, F., B. Morin, and A. Solberg. *A model-driven approach to develop adaptive firmwares*. in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2011. ACM.
58. Harrand, N., 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*. 2016. ACM.
59. Morin, B., N. Harrand, and F. Fleurey, *Model-based software engineering to tame the iot jungle.* IEEE Software, 2017. **34**(1): p. 30-36 %@ 0740-7459.
60. Atkinson, C. and T. Kühne, *Rearchitecting the UML infrastructure.* ACM Transactions on Modeling and Computer Simulation (TOMACS), 2002. **12**(4): p. 290-321.
61. Kühne, T., *Matters of (meta-) modeling.* Software & Systems Modeling, 2006. **5**(4): p. 369-385.
62. Jeon, J., S. Kang, and I. Chun, *Cps-based model-driven approach to smart manufacturing systems.* INTELLI 2016, 2016: p. 146.
63. Patel, P. and D. Cassou, *Enabling high-level application development for the Internet of Things.* Journal of Systems and Software, 2015. **103**: p. 62-84.
64. Cecchinel, C., S. Mosser, and P. Collet. *Software development support for shared sensing infrastructures: A generative and dynamic approach*. in *International Conference on Software Reuse*. 2015. Springer.
65. Song, J., et al., *Connecting and managing m2m devices in the future internet.* Mobile Networks and Applications, 2014. **19**(1): p. 4-17.
66. Teixeira, T., et al. *Service oriented middleware for the internet of things: a perspective*. in *European Conference on a Service-Based Internet*. 2011. Springer.
67. Garlan, D. *Software engineering in an uncertain world*. in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. 2010. ACM.
68. Apel, S. and C. Kästner, *An overview of feature-oriented software development.* Journal of Object Technology, 2009. **8**(5): p. 49-84.
69. Bruns, G. *Foundations for Features*. in *FIW*. 2005. Citeseer.
70. Apel, S., et al. *Feature interactions: the next generation (dagstuhl seminar 14281)*. in *Dagstuhl Reports*. 2014. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
71. Kusmenko, E., et al. *Modeling architectures of cyber-physical systems*. in *European Conference on Modelling Foundations and Applications*. 2017. Springer.

72. Yagita, M., F. Ishikawa, and S. Honiden. *An application conflict detection and resolution system for smart homes*. in *Proceedings of the First International Workshop on Software Engineering for Smart Cyber-Physical Systems*. 2015. IEEE Press.
73. Ma, M., et al. *Detection of runtime conflicts among services in smart cities*. in *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. 2016. IEEE.
74. Balaji, B., et al., *Modeling Actuation Constraints for IoT Applications*. arXiv preprint arXiv:1701.01894, 2017.
75. Rathmair, M., C. Luckeneder, and H. Kaindl. *Minimalist qualitative models for model checking cyber-physical feature coordination*. in *Software Engineering Conference (APSEC), 2016 23rd Asia-Pacific*. 2016. IEEE.
76. Takeuchi, S., M. Takemoto, and M. Matsuo. *SPIRE: Scalable and Unified Platform for Real World IoT Services with Feature Interaction*. in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*. 2016. IEEE.
77. Sarray, I., et al. *Safe composition in middleware for the internet of things*. in *Proceedings of the 2nd Workshop on Middleware for Context-Aware Applications in the IoT*. 2015. ACM.
78. Balani, R., L.F. Wanner, and M.B. Srivastava, *Distributed programming framework for fast iterative optimization in networked cyber-physical systems*. ACM Transactions on Embedded Computing Systems (TECS), 2014. **13**(2s): p. 66.
79. Corredor, I., et al., *Knowledge-aware and service-oriented middleware for deploying pervasive services*. Journal of Network and Computer Applications, 2012. **35**(2): p. 562-576.
80. Cano, J., et al. *Eca rules for iot environment: a case study in safe design*. in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2014 IEEE Eighth International Conference on*. 2014. IEEE.
81. Zhao, M., et al. *Discrete control for smart environments through a generic finite-state-models-based infrastructure*. in *European Conference on Ambient Intelligence*. 2014. Springer.
82. Umakoshi, K., et al. *S^3: Smart Shadow System for Real World Service and Its Evaluation with Users*. in *Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on*. 2011. IEEE.
83. Alfakeeh, A. and A. Al-Bayatti, *Feature Interactions Detection and Resolution in Smart Homes Systems*. 2016.
84. Dominka, S., et al. *Taming and optimizing feature interaction in software-intensive automotive systems*. in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. 2018. IEEE.
85. Juarez Dominguez, A. *Feature interaction detection in the automotive domain*. in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. 2008. IEEE Computer Society.
86. Hu, H., et al., *Semantic Web-based policy interaction detection method with rules in smart home for detecting interactions among user policies*. IET communications, 2011. **5**(17): p. 2451-2460.
87. Sehili, S., et al., *IoT Efficient Design Using WComp Framework and Discrete Event Modeling and Simulation*, in *Simulation and Modeling Methodologies, Technologies and Applications*. 2016, Springer. p. 49-67.
88. D'Angelo, G., S. Ferretti, and V. Ghini. *Simulation of the Internet of Things*. in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. 2016. IEEE.
89. Chernyshev, M., et al., *Internet of Things (IoT): research, simulators, and testbeds*. IEEE Internet of Things Journal, 2018. **5**(3): p. 1637-1647.
90. Luthra, S., et al., *Analyzing challenges to Internet of Things (IoT) adoption and diffusion: An Indian context*. Procedia Computer Science, 2018. **125**: p. 733-739.
91. Osterlind, F., et al. *Cross-level sensor network simulation with cooja*. in *Local computer networks, proceedings 2006 31st IEEE conference on*. 2006. IEEE.
92. Varga, A., *OMNeT++*, in *Modeling and tools for network simulation*. 2010, Springer. p. 35-59.
93. Sotiriadis, S., et al. *Towards simulating the internet of things*. in *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*. 2014. IEEE.
94. Zeng, X., et al., *IOTSim: A simulator for analysing IoT applications*. Journal of Systems Architecture, 2017. **72**: p. 93-107.

95. Gupta, H., et al., *iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments.* Software: Practice and Experience, 2017. **47**(9): p. 1275-1296.
96. Kirsche, M. *Simulating the internet of things in a hybrid way*. in *Proceedings of the Networked Systems (NetSys) 2013 PhD Forum*. 2013.
97. Fleurey, F. and B. Morin. *ThingML: A Generative Approach to Engineer Heterogeneous and Distributed Systems*. in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW 2017), Gothenburg, Sweden, 5-7 April 2017*. 2017.
98. Eldabi, T., et al. *Hybrid simulation: Historical lessons, present challenges and futures*. in *Winter Simulation Conference (WSC), 2016*. 2016. IEEE.
99. Mosterman, P.J. *An overview of hybrid simulation phenomena and their support by simulation packages*. in *International Workshop on Hybrid Systems: Computation and Control*. 1999. Springer.
100. Gluhak, A., et al., *A survey on facilities for experimental internet of things research.* IEEE Communications Magazine, 2011. **49**(11): p. 58-67.
101. Kecskemeti, G., et al., *Modelling and simulation challenges in internet of things.* IEEE cloud computing, 2017. **4**(1): p. 62-69.
102. Looga, V., et al. *Mammoth: A massive-scale emulation platform for internet of things*. in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*. 2012. IEEE.
103. McKee, D.W., et al. *The internet of simulation, a specialisation of the internet of things with simulation and workflow as a service (sim/wfaas)*. in *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. 2017. IEEE.
104. D'Angelo, G. and M. Marzolla, *New trends in parallel and distributed simulation: From many-cores to cloud computing.* Simulation Modelling Practice and Theory, 2014. **49**: p. 320-335.
105. Han, S.N., et al. *DPWSim: A simulation toolkit for IoT applications using devices profile for web services*. in *2014 IEEE World Forum on Internet of Things (WF-IoT)*. 2014. IEEE.
106. Krajzewicz, D., et al., *Recent development and applications of SUMO-Simulation of Urban MObility.* International Journal On Advances in Systems and Measurements, 2012. **5**(3&4).
107. Luke, S., et al., *Mason: A multiagent simulation environment.* Simulation, 2005. **81**(7): p. 517-527.
108. Mehdi, K., et al. *Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool*. in *7th International ICST Conference on Simulation Tools and Techniques, Lisbon, Portugal, 17-19 March 2014*. 2014. Institute for Computer Science, Social Informatics and Telecommunications Engineering (ICST).
109. Zeng, X., R. Bagrodia, and M. Gerla. *GloMoSim: a library for parallel simulation of large-scale wireless networks*. in *ACM SIGSIM Simulation Digest*. 1998. IEEE Computer Society.
110. Martínez-Salio, J.-R., et al. *A comparison of simulation and operational architectures*. in *2012 Fall Simulation Interoperability Workshop (SIW), Simulation Interoperability Standards Organization (SISO)*. 2012.
111. Shekhar, S., et al., *A simulation as a service cloud middleware.* Annals of Telecommunications, 2016. **71**(3-4): p. 93-108.
112. Schmucker, K., *A taxonomy of simulation software.* Learning Technology Review, 1999: p. 40-75.
113. Miller, J.A., et al. *Investigating ontologies for simulation modeling*. in *Proceedings of the 37th annual symposium on Simulation*. 2004. IEEE Computer Society.
114. McKee, D.W., et al. *DIVIDER: Modelling and Evaluating Real-Time Service-Oriented Cyberphysical Co-Simulations*. in *Real-Time Distributed Computing (ISORC), 2015 IEEE 18th International Symposium on*. 2015. IEEE.
115. Sargent, R.G. *Verification and validation of simulation models*. in *Simulation Conference (WSC), Proceedings of the 2009 Winter*. 2009. IEEE.
116. Fishwick, P.A., *A taxonomy for simulation modeling based on programming language principles.* IIE transactions, 1998. **30**(9): p. 811-820.

117. Blochwitz, T., et al. *The functional mockup interface for tool independent exchange of simulation models*. in *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 2011. Linköping University Electronic Press.
118. D'Angelo, G., S. Ferretti, and V. Ghini, *Multi-level simulation of internet of things on smart territories.* Simulation Modelling Practice and Theory, 2017. **73**: p. 3-21.