Title:	*Requirements and conceptual design of techniques and methods for trustworthy & agile operation of smart IoT systems*
Authors:	*Haroon Ahmed (CA), Nicolas Ferry (SINTEF), Smrati Gupta (CA), Stéphane Lavirotte (CNRS), Andreas Metzger (UDE), Victor Muntés (CA), Phu NGuyen (SINTEF), Alexander Palm (UDE), Andreas Reiss (CA), Marc Solé (CA), Jean-Yves Tigli (CNRS)*
Editors:	*Stéphane Lavirotte (CNRS), Jean-Yves Tigli (CNRS)*
Reviewers:	*Anne Gallon (Evidian), Eider Iturbe Zamalloa (Tecnalia)*
Identifier:	*Deliverable # D3.1 v1.5*
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 operation of IoT systems. In addition, it will characterize the requirements to be considered and provide an initial design of the solutions developed in WP3.

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
Fundacion Tecnalia 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	Stéphane Lavirotte	Outline and summaries
12 April	0.1	Jean-Yves Tigli	Outline and table of content
25 May	0.2	Andreas Metzger	Second release of table of content (partners validated)
27 July	0.3	Jean-Yves Tigli	Merging each partners' contributions, now working on the same document
14 September	0.4	Marc Solé	Section 4
19 September	0.5	Nicolas Ferry, Phu NGuyen	Section 5.1
19 September	1.0	Stéphane Lavirotte	Complete version of document with formatting and style modifications. Comments added to partners' sections for modifications
24 September	1.0	Andreas Metzger	Additions to Section 1
02 October	1.0	Nicolas Ferry	Review of the whole document
04 October	1.1	Nicolas Ferry, Stéphane Lavirotte, Andreas Metzger, Phu Nguyen, Jean-Yves Tigli	Addressing comments and modifications requests
05 October	1.2	Stéphane Lavirotte, Jean-Yves Tigli	Revision for internal review
17 October	1.3	Anne Gallon, Eider Iturbe Zamalloa	Comments and modifications added by reviewers
24 October	1.4	Nicolas Ferry, Stéphane Lavirotte, Andreas Metzger, Phu NGuyen, Alexander Palm, Marc Solé, Jean-Yves Tigli	Corrections after internal review
31 October	1.5	Stéphane Lavirotte, Jean-Yves Tigli	Final release of the deliverable

Contents

1 Introduction

1.1 Context and objectives

The operation of large-scale and highly distributed IoT system can easily overwhelm operation teams. Major challenges are to improve their efficiency and the collaboration with developer teams for rapid and agile evolution of the system. In particular, automated solutions for run-time operations are required in order to ensure timely reaction to problems and changes of the IoT system's environment.

WP3 aims to develop enablers for the operational part of the DevOps process (see Figure 1). WP3 thus will provide enablers that furnish the IoT systems with capabilities to (i) monitor their status, (ii) indicate when their behaviour is not as expected, (iii) identify the origin of the problem, and (iv) automatically perform typical operation activities (including self-adaptation of the systems). As it is impossible to anticipate all problems and environment situations systems may face when operating in open contexts, there is an urgent need for mechanisms that will automatically learn and update the operation and adaptation activities of Smart IoT Systems (SIS).

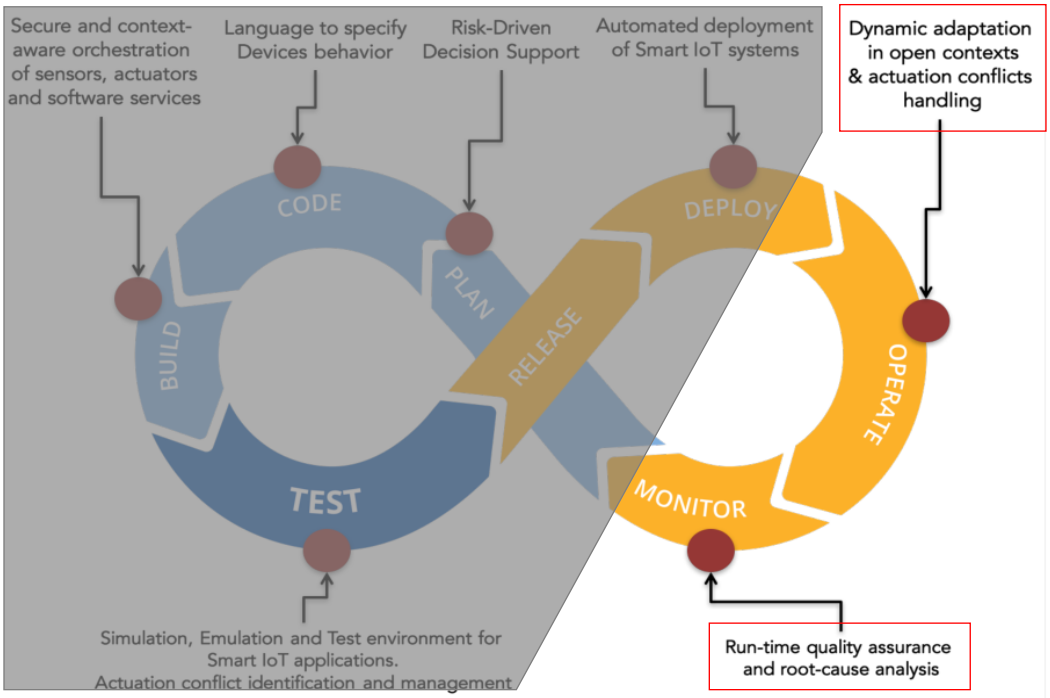

Figure 1: Focus on the Ops of the DevOps cycle

The three enablers developed by WP3 are:

- **Online Learning Enabler:** Because anticipating all possible context situations that SIS may encounter during their operation is not possible, it is difficult for software developers to determine how a run-time adaptation of the system may impact the satisfaction of the system behaviour and of the interactions with the environment. To address this challenge, this enabler will apply online learning techniques to improve the way a SIS adapts during its operation. Online learning means that learning is performed at run-time, taking into account observations about the actual system execution and system context. Online learning incrementally updates the SIS's knowledge base; *e.g.*, its adaptation rules or the models based on which adaptation decisions are made.
- **Behavioural Drift Analysis Enabler:** Because of the uncertain, dynamic, and partially known nature of the physical environment, it is very difficult or even illusory to assess at run-time the conformity of the effects of actions in this environment with deterministic models. This enabler will provide a set of observers to monitor the behavioural drift of SIS that may arise when

operating in such open context. In addition, it will exploit the computed drift measure to dynamically adjust the behaviour of the system.

– **Root Cause Analysis Enabler:** When anomalous conditions start to arise in a complex system, determining which anomalies are related and to which part focus attention is crucial to reduce the mean time to resolution. Thus, the root-cause analysis enabler will try to sensibly group anomalies related to the same problem and compute likely culprits of that problem with the least amount of human involvement possible. Since the number of open incidents in a large deployment can be large, it will as well prioritize the different grouped problems by potential impact, based on past experience.

In line with these three enablers, WP3 pursues the following three main objectives (note that these objectives have been refined and detailed based on the discussions and insights during the first months of the project[1]):

– **O1 - Online Learning:** Design and prototypical implementation of *online learning* techniques to self-improve the way a SIS adapts during its operation, thereby providing run-time adaptation mechanisms for SIS allowing to adapt the systems to context changes not anticipated during design time (task T3.1);

– **O2 - Behavioural Drift Analysis:** Design and prototypical implementation of *behavioural drift analysis* of applications, to allow SIS to react in the given run-time context toward uncertainties pertaining the physical environment and conflicting actuations possibly jeopardizing their functionalities (task T3.2);

– **O3 - Root Cause Analysis:** Design and prototypical implementation of *root cause analysis* of SIS to establish trust in the qualities of SIS (Task T3.3).

The figure below (Figure 2) shows a high-level architecture positioning the three enablers (objectives) of WP3. The Online Learning enabler sits on top of the self-adaptation logic to continuously improve the set of adaptation actions available at run-time. The Behavioural Drift Analysis enabler monitors the system logic in operation and if behavioural drifts are detected, these may either lead to automatic actions (via the Online Learning enabler) or to feedback to the Development cycle of DevOps. The Root Cause Analysis enabler detects root causes at run-time in order to either trigger adaptation (by pinpointing to the root cause for the problem) or to feed back the root causes to development.

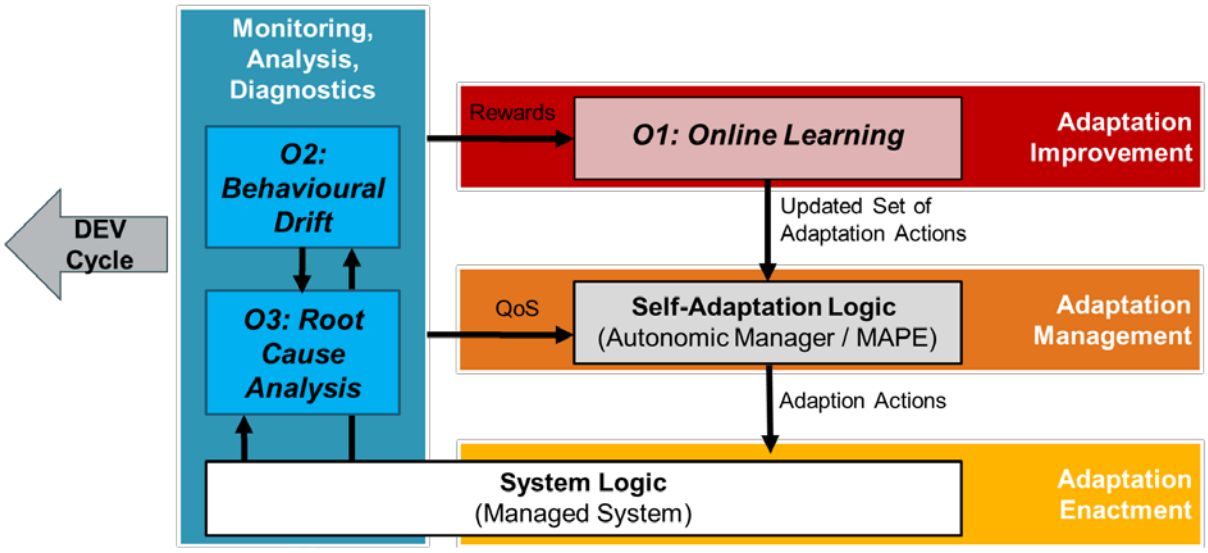

Figure 2: Positioning of the WP3 Enablers in the Conceptual ENACT Architecture

[1] In particular, for the time being, it was decided to focus the UDE resources on Task 3.1 and the interaction and collaboration with T3.2, and not work on the run-time model checking elements of T3.3.

This deliverable D3.1 focuses on describing the relevant state-of-the-art for the three enablers of WP3, as well as to provide a set of requirements and an initial, conceptual design. These enablers will need support for adaptation which is provided by the GeneSIS enabler (WP2) described in D2.1. This deliverable describes the state-of-the-art, requirements and design for adaptation enactment to serve as a technical basis for self-adaptation. Finally, this deliverable describes the interdependencies and synergies among the three enablers developed in WP3.

1.2 Achievements

Objectives	Achievements
State-of-the-art on facilitating the operations of SIS	We conducted an analysis of the state-of-the-art on approaches to facilitate the operations of SIS. Both industrial and academic approaches were considered. We specifically focused on four topics: 1. Online-learning for adaptation self-improvement 2. Behavioural drift analysis 3. Root-cause analysis 4. Support for self-adaptation
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, and (ii) what will serve as baseline technology.

1.3 Structure of the document

The remainder of the document is structured as follows.

In Section 2, the online learning for adaptation self-improvement of SIS is presented. The state-of-the-art (Subsection 2.1) emphasises online-learning, focussing on reinforcement learning (2.1.1), case-based reasoning (2.1.2) and search-based and multi-agent learning (2.1.3). As a result, requirements (Subsection 2.2) and conceptual design (Subsection 2.3) for the online-learning enabler are described.

In section 3, the state-of-the-art of context awareness and monitoring is provided, focussing on behavioural drift analysis, overlooked in most of the current approaches. Subsections 3.1.3 deals with static modelling approaches while Subsection 3.1.4 deals with dynamic modelling approaches. Requirements (3.2) and conceptual design (3.3) conclude this section.

Section 4 covers root-cause analysis (RCA) whose objective is to infer what is causing an observed anomaly by going back through the causal chains governing the system under analysis. A state-of-the-art of root-cause analysis is presented in Subsection 4.1 followed by the identified requirements (4.2) and conceptual design (4.3).

The next section (5) presents the support and interrelationships among techniques for agile operations of SIS. Subsection 5.1 presents the support for self-adaptation. It is followed by the interrelationship between the behavioural drift analysis as input for online-learning (Subsection 5.2), root-cause analysis as input for online-learning (Subsection 5.3) and the interrelationship between root-cause analysis and behavioural drift analysis (Subsection 5.4).

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

2 Online Learning for Adaptation Self-improvement of Smart IoT Systems

In this chapter the state-of-the-art on online-learning and the requirements for, as well as the conceptual design of the "Online Learning enabler" are described.

In simple terms, Online learning uses observations about the actual system execution and environment (in the form of rewards; see Figure 2) to incrementally update and thus improve the self-adaptation logic of the IoT systems. This means, referring to Figure 2, we plan to use Online Learning to improve the Autonomic Manager at run time.

2.1 State-of-the-art on Online Learning

This section discusses existing online learning techniques for self-adaptive systems. We focus on rule-based approaches (or model-free learning), because they offer specific benefits in the setting of SIS. On the one hand, rules can be executed more efficiently than models (because there is no need for additional reasoning on models) and thus can be executed also on IoT devices with limited resources. On the other hand, rules are human-readable and thus provide additional feedback for the "Dev" part of the DevOps cycle.

We analyse the state-of-the-art techniques with respect to two key characteristics: (1) How fast learning converges to an improved set of adaptation rules. (2) How system evolution is taken into account. This means, we analyse the state-of-the-art techniques with respect to the following two key requirements (also see Section 2.2 below):

- *Convergence:* An important concern when using online learning to improve the self-adaptation logic is the time it takes for the learning process to converge. Due to the trial-and-error process of selecting and executing adaptation actions, online learning may require many observations to learn adaptation actions that are effective in adapting the running system. While online learning is converging, the self-adaptive system may thus execute ineffective adaptation actions. Executing ineffective adaptations can have real and negative consequences, because the adaptation actions modify the live system, which is of a particular concerning in the IoT setting.
- *System evolution:* During evolution, software developers may modify the system, for instance, to correct bugs, to remove seldom used features, or to introduce new features to meet new requirements, thereby changing the potential adaptation actions of the self-adaptive system. The speed of online learning becomes a particular concern in the presence of continuous evolution and deployment (a key characteristic of DevOps), as the time between evolution cycles gets shorter and shorter. Learning thus needs to happen fast enough such that it has converged before the next evolution cycle occurs to make sense.

We structure the analysis of the state-of-the-art into the three categories: Reinforcement learning, Case-based reasoning and Search-based and Multi-Agent Learning.

2.1.1 Reinforcement Learning

In reinforcement learning, the effectiveness of system actions are learned through interactions with its environment [1]. The system receives a reward value as feedback for applying an action. The overall aim of reinforcement learning is to maximize cumulative rewards. Reinforcement learning relies on two different mechanisms: exploitation, which recommends the actions with the highest reward to be executed, and exploration, which recommends an action (typically chosen randomly) regardless of its reward value to prevent learning sub-optimal solutions. Unlike supervised learning, which learns from a set of labelled training data, reinforcement learning relies only on the feedback from the environment. Informally, the action that leads to the highest reward for a given environment becomes the rule for use whenever this environment is encountered [2].

Tesauro et al. suggest combining reinforcement learning with queuing models to improve resource allocation policies (*i.e.*, rules) in data centers [3]. The queuing models are used to generate adaptation decisions during an offline reinforcement learning phase. Convergence is facilitated by first learning offline, thereby avoiding potentially poor rule effectiveness during the beginning of online learning. However, this rests on the assumption that the queuing models are good enough to generate effective enough adaptations. System evolution is not addressed. Tesauro et al. do not consider how possible new adaptation actions may be considered during online learning.

Amoui et al. propose using reinforcement learning to learn the best adaptation actions for any given environment state of a self-adaptive software system [4]. They describe different solutions to increase convergence of learning, which include performing offline learning before deployment and using a simulation of the software system to generate observations. In addition, they observe that different reinforcement learning algorithms (such as Q-Learning or SARSA) may exhibit different convergence rates depending on the concrete application context. System evolution is not addressed.

Kim and Park propose Q-Learning as a reinforcement learning algorithm for improving adaptation rules at run-time [5]. Goal and scenario models are used to manually determine a set of possible states and actions as input to learning. They show that online learning may gradually optimize the rule set but provide no further analysis of convergence rates. They do not consider how changes in the states and actions due to an evolution of the goal or scenario models may be captured during online learning. Barrett et al. propose using Q-Learning for the autonomic resource allocation in the cloud [6]. To facilitate convergence, they propose parallel learning. However, this requires that several systems concerned with the same resource allocation tasks exist and thus could share the information they learn in parallel. System evolution is not explicitly addressed, and in principle would become quite difficult if involved systems underwent different forms of evolution in parallel.

Jamshidi et al. apply fuzzy Q-Learning to derive and improve fuzzy adaptation rules [7]. Adaptation rules are evaluated based on their long-term impacts, that are calculated by predefined reward functions. By observing the actual impact of adaptations on the system performance at run-time, adaptation rules are modified to match the observed system and environment state. With respect to convergence, Jamshidi et al. observe that the exploitation and exploration rates influence convergence. As an extension of their initial work, they also demonstrate that transfer learning may accelerate learning [8]. However, transfer learning is beneficial only if observations from the source environment are much cheaper to collect than samples from the target environment. Their approach does not address system evolution, as it assumes that the set of system actions to be explored is fixed.

Filho and Porter [9] use a variant of reinforcement learning to determine which composition of software components best suits the current operating environment. Their experimental results indicate that the size of the observation window, *i.e.*, how long observations are collected before deciding on rewards, has an impact on how many iterations of exploration are required. However, they provide no further analysis on how this may be exploited for increasing convergence rates. In their approach, candidate components are determined by scanning from an initial, main component all required and provided interfaces of linked components. Thereby, system evolution may be supported, by rescanning the component interfaces after the system has been modified. Online learning would hence be able to explore the new configuration space. Yet, as their learning is agnostic to which changes have been made, it is not guided towards exploring new or changed components.

Sharifloo et al. [10], describe the dependencies between online learning and system evolution. On the one hand, they indicate how feedback from learning itself (*e.g.*, if no effective system configuration could be found) may trigger system evolution. On the other hand, they analyse how the configuration space may change during system evolution and how such a change may affect online learning. Further they describe how these dependencies may be considered by extending the exploitation and exploration phases of reinforcement learning. However, their work does neither provide concrete algorithms nor experimental results considering the convergence of online learning.

In summary, convergence in reinforcement-learning-based approaches has been addressed from the point of view of the concrete learning algorithm (such as Q-Learning vs. SARSA), the balance between exploitation and exploration, as well as by first performing an offline learning phase. However, unlike

the strategies we aim to deliver in ENACT, none of the existing approaches use the system structure to better guide selection of adaptations to be explored during online learning. Also, none of the approaches take evolution into account.

2.1.2 Case-based Reasoning

Case-based reasoning aims to solve new problems by reusing solutions for similar past problems [11]. To this end, case-based reasoning involves four main steps. First, given a target problem, similar cases are retrieved from the case base. Second, similar cases are mapped to the target problem by modifying the retrieved solutions if needed. Third, the modified solution is tested on the target problem and further revised if needed. Fourth, successful solutions are stored in the case base.

Qian et al. employ case-based reasoning for storing and retrieving adaptation rules [12]. When facing a new situation, similar cases are retrieved from the case base to find an adaptation whose effectiveness has been shown earlier. They learn from past executions to identify which adaptation would be more effective for the system in its current environment. If no similar case can be found in the case base, their approach resorts to using goal models to derive new adaptation rules. The use of goal models to derive new adaptation rules provides a more structured approach than randomly deriving new adaptation rules. Qian et al. provide no analysis of the convergence rate of their approach and whether using goal models to derive new adaptation rules may speed up convergence. The approach uses a goal model defined at design time and does not address how this goal model may be updated due to system evolution and how the update of the goal model would impact the case base.

Zhao et al. propose using case-based reasoning in combination with reinforcement learning to generate and update adaptation rules [13]. To populate the case base, they use offline reinforcement learning to learn adaptation rules for different system goals. During run-time, case-based reasoning is used to select the best fitting rule and reinforcement learning is used to fine-tune this rule. They present exemplary scenarios that indicate that even though their approach may take as long to converge on an optimal rule set as online learning from scratch, it starts with a higher effectiveness of the adaptation rules. Even though the approach explores different settings of system goals, it does not consider an evolution of the system itself.

In summary, convergence was not explicitly addressed in the above contributions. Also, evolution was not considered.

2.1.3 Search-based and Multi-Agent Learning

Ramirez et al. [14] employ genetic algorithms to improve reconfiguration plans at run-time. Genetic algorithms are a class of stochastic-based search techniques applying selection, cross-over and mutation operators to a set of candidate solutions. They incorporate monitoring information into the genetic algorithms, such that a change in the environment drives the search process. With respect to convergence, their experimental results show that a relatively good set of reconfiguration plans may be attained after 500 iterations (*i.e.*, generations of candidate solutions). System evolution is not explicitly supported by their approach, as the introduction of new system features would require the manual definition of new operators for the genetic algorithm.

Moustafa and Zhang propose multi-agent, collaborative reinforcement learning for adaptive service compositions [15]. They express the service composition as a Markov decision process and use reinforcement learning to determine the service selection policy that leads to the highest reward. To speed up convergence, they propose using collaborative learning, where multiple systems simultaneously explore the set of concrete services to be composed. They observe that such collaborative learning significantly increases the speed of exploration. However, like for Barrett et al. [6], this requires that several systems with the same learning goal exist. With respect to system evolution, they do not address how a change in the structure of the underlying Markov decision process may impact on learning.

Wang et al. combine multi-agent reinforcement learning with game theory for adaptive service compositions [16]. Their experimental results indicate that convergence depends on the learning rate (*i.e.*, to what degree newly observed rewards override past rewards), the number of agents collaborating, as well as the size of the composition problem (measured in terms of the service composition structure and number of concrete services). They observe that the speedup of convergence peaks at a certain number of agents involved and adding more agents after that point has a negative effect on convergence. As for Moustafa and Zhang [15] as well as Barrett et al. [6], their approach requires that several systems exist that have the same learning goal. Similarly, concerning evolution, the paper does not address how a change on the structure of the underlying service composition model (a Markov decision process) may impact on learning.

In summary, even though convergence was analysed in the above contributions, no specific proposals for exploiting the system structure (like we plan in ENACT) was proposed. Also, evolution was not explicitly addressed.

2.2 Requirements for Online Learning enabler

As discussed in detail above, the state-of-the-art on online learning for adaptive systems is addressing the "convergence" and "evolution" aspects, important for SIS, only to a very limited degree. Those two aspects are covered in requirements R5 and R6 below.

In addition, additional requirements for the Online Learning enabler, which need to be addressed by the conceptual design, are described in the following paragraphs. An overview of the requirements is given in Table 1.

Table 1: WP3 Requirements for Online Learning enabler

ReqID	Requirement	Description
R1	Coping with unknown environment	Online Learning must be able to cope with environments that are unknown at design time (uncertainty)
R2	Learn online during live operation	Online Learning must continuously learn during operations, in parallel to the live system
R3	Frequent updates of adaptation rules	Online Learning must frequently enough update the adaptation rules to cope with fast and dynamic changes in the system and its environment
R4	Minimum overhead	Online Learning may only impose a small resource overhead such that it can be executed also in resource-constrained (IoT / embedded) devices
R5	Fast convergence	Online Learning must converge quickly in order to ensure that adaptation actions are effective
R6	Accounting system evolution	Online Learning must take into account system evolution (from DEVelopment) in order to accommodate new adaptation actions introduced during system evolution[2]
R7	Coping with non-stationarity	Online Learning must work in the presence on non-stationary environments
R8	Coping with large state and action spaces	Online Learning must be able to cope with large action and environment state spaces to be practically applicable

The Online Learning enabler should enable an IoT-system to *cope with unknown environments* (R1), which means that adaptation rules that describe how the system should adapt itself to the environment are provided and also updated based on actual observations during operations. Based on this, it is

[2] In particular, this evolution also entails a change of the adaption rules by developers. As such, possible conflicts between manual modification of the rules vs. automated modification of the rules by means of Online learning have to be considered.

assumed that the system (also called agent) does not have any prior knowledge about the environment and the environment dynamics.

To reduce human intervention, the Online Learning enabler should *enable the system to learn online* (R2). This means that learning is done during live operation and no explicit training phase should be passed to learn an initial set of adaptation rules (according to this, an initial set of adaptation rules needs to be provided a priori and is then directly being optimized through the Online Learning enabler).

To make learning as efficient and fast as possible, the system should be enabled to learn from every piece of experience it can gather, so that *updates of the adaptation rules can be done in each timestep* (R3).

Another important requirement is to *reduce overhead as much as possible* (R4) (concerning computation complexity and memory consumption), as also devices with very limited computing power should be able to compute the according online learning algorithms.

Although gathering experience during live operation might be a bit costly, it is the only opportunity while relinquishing simulations in an offline phase. According to this, the Online Learning algorithms used in the Online Learning enabler should provide a *fast convergence* (R5), which means that the algorithms should be optimized so that live operation of the system is not restricted in any way.

The Online Learning enabler should also *take system evolution into consideration* (R6), which means that new actions that did not exist in prior time steps, should be taken into consideration as soon as possible and not only existing system capabilities should be considered during optimization of the adaptation rules.

As the dynamics of the environments (which are not known to the agent) might change over time (so-called non-stationary environments), the frequency of *changes in the environment need to be considered properly* (R7). This means that in an optimal case, learning has converged before a change in the environment dynamics has occurred and a set of optimized adaptation rules might be mapped to a certain situation of environment dynamics.

Last, the Online Learning enabler and the underlying algorithms/techniques should be able to *cope with very large state spaces* (R8) resulting from continuous environment variables and a possible large set of actions (*e.g.*, if different configurations of a system are considered as actions, millions of them might be possible). In this case possibly the field of function approximation should be investigated.

2.3 Conceptual design of Online Learning enabler

The choices that have been made concerning the conceptual design of the Online Learning enabler are described and justified in this section. Each choice directly refers to one or more requirements.

When it comes to enabling an agent to cope with an unknown environment (cf. requirement R1 from above), Reinforcement Learning (RL) is the means of choice. An agent using RL gains knowledge about the environment and the values of the actions he can perform in the environment through perceiving evaluative feedback (in contrast to *e.g.*, supervised learning, which is using instructive feedback). This means that no a-priori knowledge about the environment dynamics are needed. Based on the environment state S_t, the agent chooses an action a_t to perform. The action then has a direct influence on the environment state, so that the environment transitions into a new state s_{t+1}. In addition, the environment responds with a certain reward (a scalar value) R_{t+1} the agent can perceive and use to compute the value of certain states or the performance of certain actions in certain states. This interaction between an agent and its environment is modelled in Figure 3.

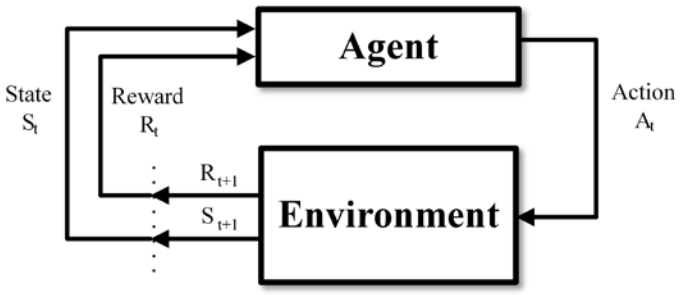

Figure 3: Basic idea of Reinforcement Learning

In the field of RL and beyond (*e.g.*, sequential decision making), Markov Decision Processes (MDP) are a suitable framework to model the underlying problem [1]. A MDP is formally defined as MDP = (S, A, T, R) with a (finite) set of states (environment states), a (finite) set of actions, a transition matrix T defining the environment dynamics through state transitions and according probabilities and a reward function R. Based on whether the environment dynamics T are given or not, one can use different methods to compute a solution for the underlying MDP in form of an optimal policy.

In the case of adaptive IoT systems, actions refer to the adaptation actions that may be carried out at run-time, while the environment states describe the environment situations that may be faced by the IoT system. The reward that an IoT system receives by carrying out an action in a given environment state is determined by monitoring. The MDP is a theoretical concept to describe the problem of finding an optimal adaptation policy for the IoT system in a given environment. How the MDP (or its elements) are implemented, depends on the concrete way how we approach the learning problem, which is described below.

If the environment dynamics are known a priori, techniques like Dynamic Programming (DP) can be used to compute the values of states and state-action-pairs by solving corresponding systems of linear equations [1]. However, to cope with unknown environments (cf. R1), as stated before, RL is the mean of choice [1]. The techniques of the RL field can therefore be divided into the subfields model-based and model-free methods [17].

Model-based methods are some kind of hybrid methods using parts of model-free methods and DP. Through trial and error, a model of the environment dynamics is sampled and as it is sufficient, DP methods are used to compute state or state-action-pair values and in the end an optimal policy to solve the underlying MDP.

In contrast to that, model-free methods (the ones which we plan to use in ENACT, see justification in Section 2.1 from above) do not rely on an explicit model of the environment dynamics and especially in the area of Temporal Difference Learning (TD-Learning) values of state-action-pairs can be updated as soon as an immediate reward has been perceived (in contrast to Monte-Carlo updates) [17]. This enables frequent updates of the knowledge and enables the methods to work online during live operation. Both aspects are required through R2 and R3.

In TD-Learning no intensive forward-search through the state space (like in DP or Monte-Carlo methods) of the underlying MDP is necessary, which reduces the overhead for computing the values of state-action-pairs (cf. R4). Through the adjustment of certain parameters (like the learning-rate and discount factor in Q-learning) the convergence of TD-learning algorithms can be tuned regarding their convergence speed.

Figure 4 depicts the conceptual design for integrating reinforcement learning into the architecture introduced in Figure 2. In particular, it should be noted that by using and expanding on reinforcement learning, we can cover both the adaptation improvement layer (by means of exploration), and the adaptation management layer (by means of exploitation). A fundamental characteristic of RL is the need for the explicit handling of exploitation and exploration [17] [1]. Exploration is the learning of new possible actions and the corresponding state-action values, while exploitation describes the use of previously learned behaviour (in terms of RL: following the currently optimal policy).

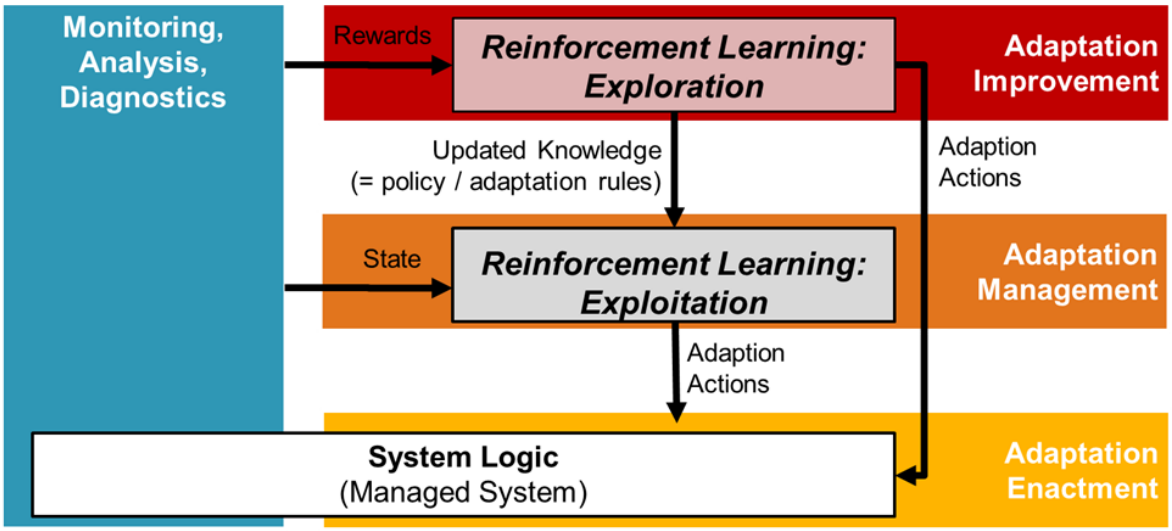

Figure 4: Conceptual design of the Online Learning enabler

As the action-space is what can be seen as possible adaptations of a system, system evolution is reflected directly within the action-space and new actions can be directly explored if an according exploration phase is triggered (cf. R6, upper part of Figure 4).

The knowledge gathered through exploration can be exploited. The optimal policy for solving the underlying MDP can be interpreted as provider of proper adaptation rules for every state the agent encounters. Exploiting the learned adaptation rules means that the system executes the adaptation actions and monitors the environment (cf. lower part of Figure 4).

The changing between exploration and exploitation in combination with the adaptation of the parameters of the underlying algorithm (*e.g.*, the learning rate in case of Q-learning) can be used to handle non-stationarity of the environment. This means that new exploration phases need to be triggered in case of a change in the environment dynamics (Subsection 5.2), like in the case of new action being available (cf. R7, see above).

Finally, the use of additional concept like function approximation might enable existing techniques to deal with very large state and action spaces (cf. R8). This is necessary because limitations in terms of memory might exist which make it more suitable to use functions for the direct computation of values instead of maintaining huge tables with explicit data for every state (of which not all might be visited).

3 Behavioural Drift Analysis of Smart IoT Systems

Online learning is meant to be performed at run-time, considering observations about the physical environment, the state of the system, and so the context. Context-awareness is key to collect sensor data, to understand it and to provide valuable information to reasoning engines. Since the first definition of context [18] a lot of middleware and software frameworks have emerged. Already in 2014, [19] finds 50 context-aware solutions in the scientific literature and today lot of well-known approaches are available [20] to collect sensors and probes data leveraged for modelling various contextual concerns (location, situation, social environment, etc.) and for inferring high level semantic information about context. In ENACT, some partners participated to past European projects that provide some solutions to collect sensors data and enrich them with semantic information (*e.g.*, Tecnalia with SMOOL, Indra with SOFIA[3], and UDE in FI-WARE[4]). Based on these tools, CNRS focuses on SIS, *i.e.*, one can observe and quantify the behavioural evolution of SIS to deal with changing context.

[3] http://sofia2.com/en
[4] https://www.fiware.org/

SIS are computing systems composed with distributed computational elements whose, when embedded in physical things, a.k.a. devices, provide these systems with an interface to the physical world through transducers (sensors and actuators). As such, SIS can be considered as subset of the Cyber-Physical Systems (CPS). These systems pose new challenges. Indeed, as far as physical things are concerned, no guaranty can be made on their availability on the long run. The underlying infrastructure of SIS can thus be volatile. Moreover, the purpose of some of these systems can only be achieved from interactions with the physical environment through actuators (*e.g.*, Heating, Ventilation and Air-Conditioning (HVAC) controllers). In this context, these systems can possibly be affected by unanticipated physical processes over which they have no control leading their behaviour to potentially drift over time in the best case or malfunction in the worst case (here are also considered concurrent SIS sharing the same physical system and possibly the same actuators, *e.g.*, an air conditioner and a heater controlling a given room). Many platforms include context awareness and monitoring mechanisms (SMOOL, SOFIA, and FI-WARE with the Orion Context Broker for instance). However, these platforms do not consider behavioural drift monitoring as an awareness criterion. This is what is addressed in this section.

Behavioural drift is closely related to the dependability of the computing systems [21] [22] [23] where the concept of integrity (*i.e.*, the absence of improper system alterations) characterizes the immunity of computing systems towards uncontrolled physical processes and associated uncertainties (*i.e.*, threats that can affect computing systems operation and undermine their dependability).

The assessment of the dependability of a computing system can be done off-line through analytical metrics using models of the systems and, whenever possible, the known uncertainty spaces[5]. Although necessary, this approach is, however, not sufficient. Indeed, the behaviour of the considered systems has to be monitored in order to ensure that their behaviour respects some properties [24].

Online monitoring involves direct and indirect empirical metrics measuring the system itself through probes and its effects in the physical environment through sensors, respectively. While methodologies involved at design phase (*e.g.*, Model-based design) and at testing phase (*e.g.*, Model checking, simulation, testing) are respectively devoted to off-line fault prevention and fault removal, online monitoring is devoted to automatic fault and anomaly detection [25].

This section discusses existing online anomaly detection techniques and specifically analyses them with respect to their ability to provide a quantitative assessment of the behaviour of the systems against their functional and non-functional requirements. In addition to its ability at providing a feedback for the "Dev" part of the DevOps cycle through dashboards, quantitative assessment also provides a feedback for the "Ops" part that can be leveraged by a self-adaptive mechanism.

3.1 State-of-the-art on Behavioural Drift Analysis of Smart IoT Systems

We do consider SIS as discrete time dynamical stochastic systems whose purpose is achieved through interaction with nonlinear physical systems (*e.g.*, a meeting room) within the physical environment (see Figure 5).

[5] Testing cyber-physical systems under uncertainty: Systematic, extensible, and configurable model-based and search-based testing methodologies (European project id 645463), 2015-2017. http://www.u-test.eu/

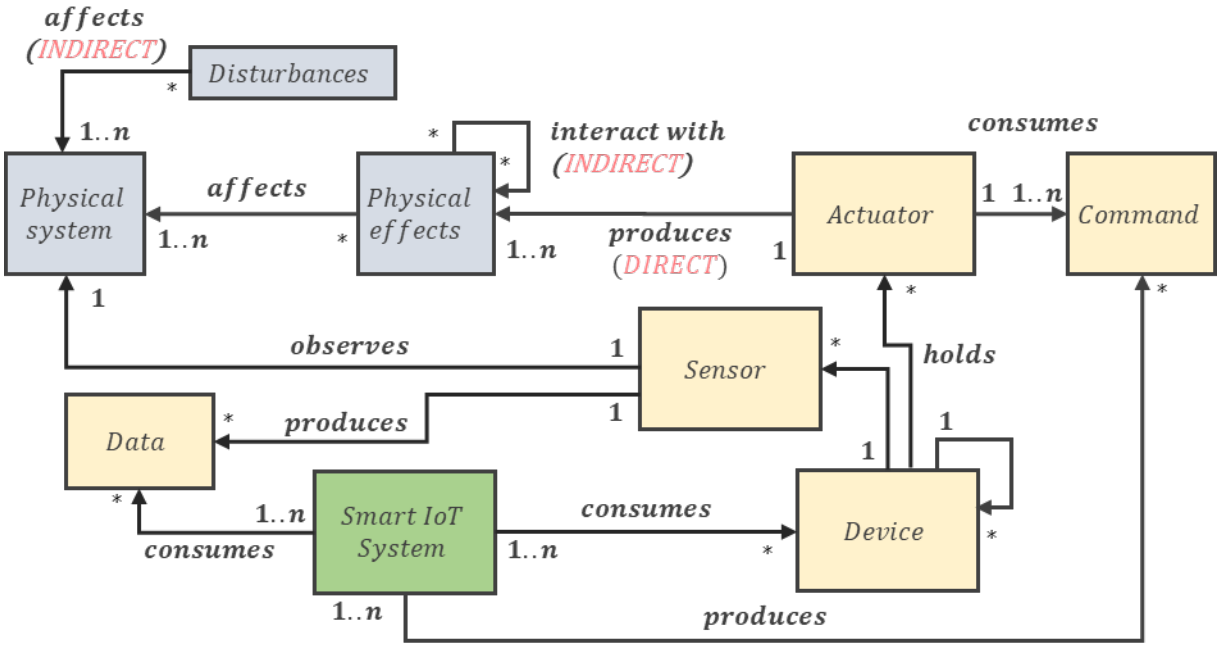

Figure 5: SIS purpose achieved through interactions with a physical system.

The observation of these systems leads to a stream of measures of the direct and indirect physical effects inherent to the commands sent to the actuators (*e.g.*, luminosity in the meeting room set to `300lux` due to `LIGHT_ON` command issued by the system) in response to stimuli (*e.g.*, presence sensor value = `10`).

More formally, observing SIS results in the generation of multivariate discrete-time data stream of continuous behavioural and contextual attributes. Behavioural attributes correspond to the observation of the direct and indirect effects within the physical environment. Contextual attributes are observations allowing to differentiate states of the system (it could be stimuli occurring within the physical environment or the system itself along with time and space information).

3.1.1 Considered anomalies

An anomaly is defined as a pattern that does not conform to the expected normal behaviour of the system [26]. It is worth noting that the anomaly detection problem and associated solutions are somehow close to the novelty detection problem [27]. Novelty detection is the identification of new or unknown data that a machine learning system is not aware of during training. A novel in this case means unusual data that are new and do not occur regularly or are simply different from the others.

In this section, we focus on complex anomalies, namely contextual (a.k.a. conditional) and collective anomalies. When contextual attributes can be associated with behaviours and their behavioural attributes, contextual anomalies are corresponding to behaviours that are valid under some conditions but are abnormal in others. For instance, in European countries, the high temperatures that typically occurs during summer would be considered as contextual anomalies if occurring in winter. Collective anomalies correspond to a collection of consecutive behaviours which are not abnormal by themselves but are abnormal when they occur together as a collection. As an illustrative example let us consider a sequence of events and actions occurring as shown below:

`PRESENCE_ON, LIGHT_ON, PRESENCE_OFF, LIGHT_OFF, **PRESENCE_ON, LIGHT_OFF,** PRESENCE_OFF, LIGHT_OFF, etc.`

The highlighted sequence is a collective anomaly although members of the sequence, taken individually, are not abnormal when they occur in other locations in the sequence.

3.1.2 Anomaly detection problem

The most common formulation of the anomaly detection problem is to determine if a given test sequence is anomalous with respect to normal sequences. More formally:

Definition 1 (Determining if a test sequence (or a subsequence within a test sequence) is anomalous *w.r.t.* the normal sequences): Given a set of n sequences $S = \{S^1_{1:K}, \ldots, S^n_{1:K}\}$ where $S^k_i \in \mathbb{R}^m$, $K, m \in \mathbb{N}^+$, $1 \leq k \leq n$, $1 \leq i \leq K$ and a sequence $S^q_{1:R}$, an anomaly score for $S^q_{1:R}$ with respect to S is computed.

We do not presuppose any synchronization feature for recording the test sequences. Moreover, we do consider that events (stimuli) may occur at any time and may last for an unknown duration. This consideration leads us to assume that test sequences might be misaligned in time and space *w.r.t* the normal sequences.

The Figure 6 provides a summary of the characteristics of the observation sequences further used as input to the anomaly detection techniques.

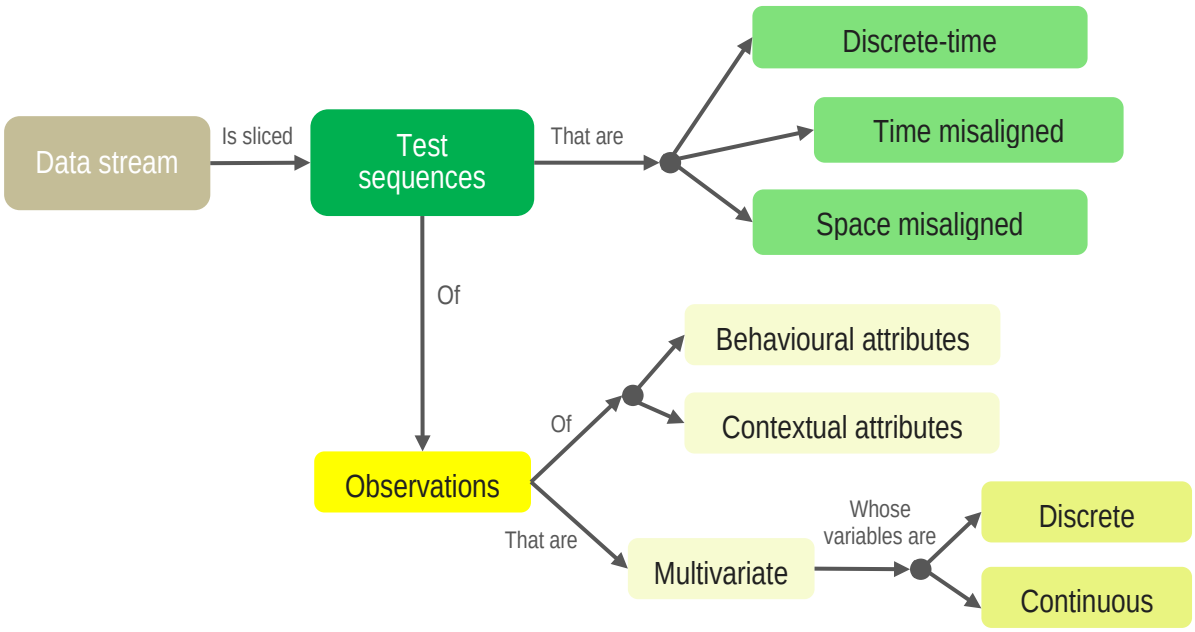

Figure 6: Overview of the characteristics of the test sequences used as input to anomaly detection techniques

Among the different techniques that handle this problem formulation, we do focus on those that can be easily adapted to comply with the characteristics of the test sequences depicted in Figure 6 and the considered anomalies described in section 3.1.1. Anomaly detection problem is widely addressed in the literature and the list of the techniques presented in the sequel has not to be taken as a complete or definitive list. To help us in our approach, we draw on existing surveys addressing anomaly detection techniques [19, 23, 25].

In this study, we disregard shape-based discriminative techniques (similarity, etc.) relying on comparing incoming test sequences with all the possible normal sequences. Indeed, because of the characteristics of the test sequences considered (Figure 6) and the SIS dynamic nature, the set of normal sequences might be infinite.

Thus, in the following, two model-based approaches are investigated:

1. **Static modelling approaches.** A model of the expected behaviour of a SIS is learned from the set S of normal sequences and is not supposed to change over time. The anomaly score is either given by (1) the likelihood of the test sequence to have been produced by the model (*e.g.*, probabilistic models) or (2) the prediction error (predictive models).

2. **Dynamic modelling approaches.** An initial model of a SIS is built from the set *S* of normal sequences. A test model is built as test sequences arrive and is compared with the initial model. The anomaly score corresponds to the dissimilarity value of both models.

3.1.3 Static modelling approaches

For these techniques, a set ***S*** of normal sequences is used to learn predictive and probabilistic parametric models. Figure 7 provides a summary of the techniques reviewed in this section.

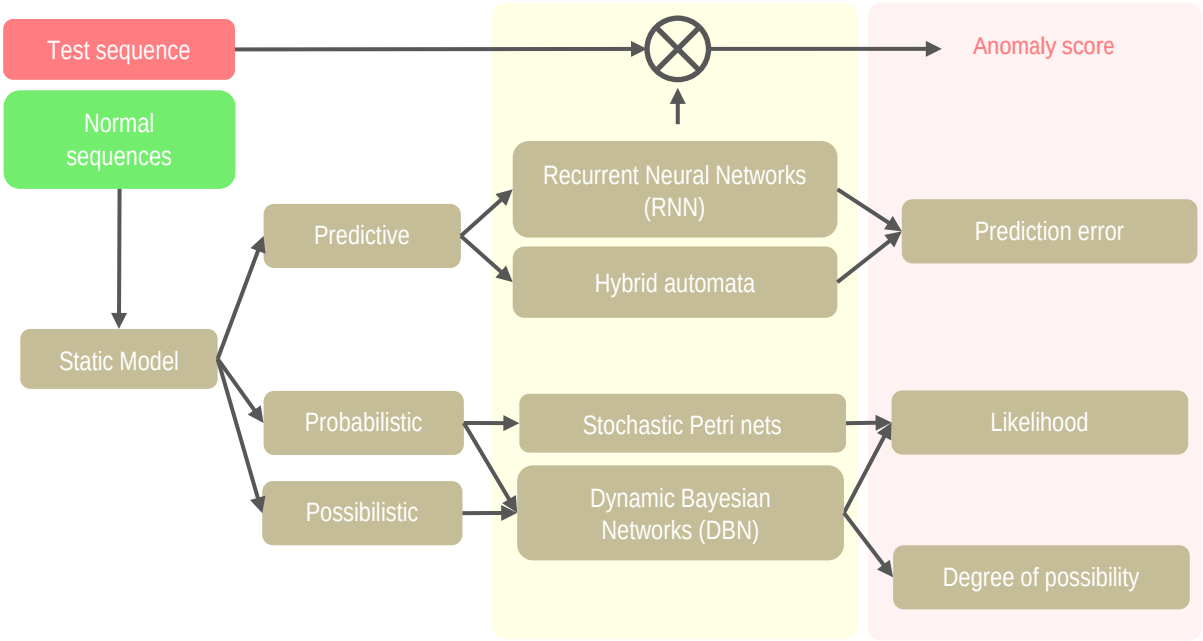

Figure 7: Quantitative anomaly detection based on static modelling approaches

3.1.3.1 Predictive Models

Predictive models consist in modelling normal behaviour through a parametric model used for predicting observation at each time *k*. Abnormal behaviours are those whose real observations differ from the predicted ones.

The main approach in this category is to model test sequences through Neural Networks (NN) and more specifically the extended Recurrent Neural Networks (RNN), *i.e.*, the Long Short-Term Memory (LSTM) approach. In [29] authors use stacked LSTM networks for anomaly/fault detection in time series. A network is trained on non-anomalous data and used as a predictor over a number of time steps. The resulting prediction errors are modelled as a multivariate Gaussian distribution, which is used to assess the likelihood of anomalous behaviour (see Figure 9).

In [30] authors present an unsupervised approach to detect cyber-attacks in Cyber-Physical Systems (CPS). A Recurrent Neural network is used as a time series predictor. The Cumulative Sum method is further used to identify anomalies in a replicate of a water treatment plant (see Figure 8).

In [31] authors provide a procedure for anomaly detection in hybrid systems able to cope with discrete and continuous variables (mainly industrial systems, *i.e.*, Cyber-Physical Production Systems (CPPS)) that uses automatically generated behaviour models (Hybrid automata). Model are learned from logged system's measurements in a hybrid automaton framework (HyBUTLA algorithm). The presented anomaly detection algorithm leverages the learned model for predicting the system behaviour and compares it with the observed behaviour in an online manner (ANODA algorithm). Alarms are raised whenever a discrepancy is found between these two.

Advantages and disadvantages of the predictive modelling techniques.

The main disadvantage of these approaches resides in the fact that they require a high amount of normal sequences for modelling all the possible system behaviours. Unfortunately, with regards to the complexity of the SIS, this is unlikely to happen. **Moreover, these models are generally hardly interpretable, their intrinsic structure and parameters making difficult reverse analysis for anomaly understanding** [32].

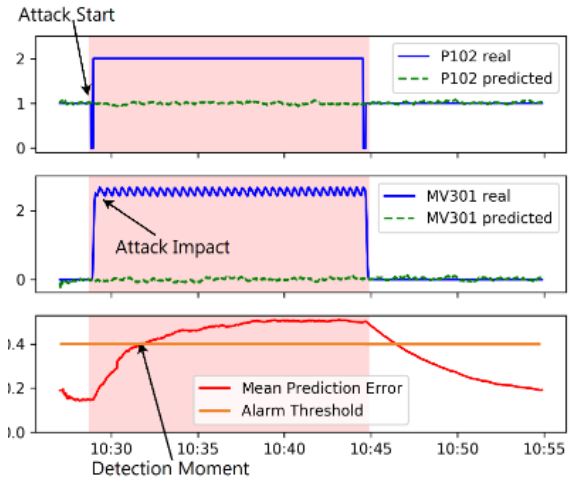

Figure 8: Attack detection on the SWaT dataset [30]

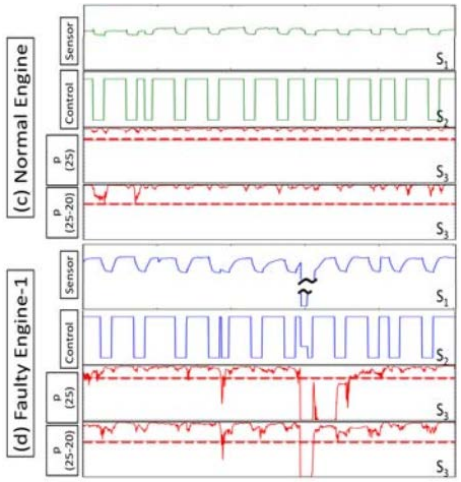

Figure 9: Anomalous sample sequences and corresponding likelihoods [29]

3.1.3.2 Probabilistic Models

Probabilistic models consist in modelling normal behaviour through a parametric model and considering abnormal behaviours as those having low probability to have been generated by the model.

In this category, Bayesian Network (BN) and Dynamic Bayesian networks (DBN) approaches are widely used. Some examples are covered in [33, 34].

In [35] authors propose a graphical model-based approach for profiling normal operational behaviour of an operational Industrial Control System (ICS) referred to as SWaT (Secure Water Treatment). Timed automata are learned as a model of regular behaviours shown in sensors signal like fluctuations of water level in tanks. BNs are learned to discover dependencies between sensors and actuators. The detection results can be quantitatively interpreted and the abnormal sensors or actuators localized thanks to the interpretability of the graphical models.

n-order Markovian models, as special case of DBN, assume that normal and test sequences are generated by a Markovian process, *i.e.*, observations at time k only depend on the state of the hidden process at time $k-1, k-2, \ldots, k-n$. Although this could be a limitation in some cases, this modelling framework is particularly well suited for representing dynamical systems and capture their uncertainties through probabilities [36, 37]. Behavioural uncertainties pertaining the transducers (sensors and actuators) and the physical environment are captured through probability density functions (pdf). In this category, Hidden Markov Models (HMM) and derivatives are widely used as anomaly detection technique [34-37].

In [41], authors propose a fault detection and isolation technique for timed stochastic discrete event systems modelled with partially observed timed Petri nets. The models include the sensors used to measure events and markings along with the temporal constraints to be satisfied by the system operations. The temporal constraints are defined according to tolerance intervals specified for each transition. A fault is an operation that ends too early or too late. The set of trajectories consistent with a given timed measured trajectory is first computed. Then, the probability that the temporal specifications

are unsatisfied is estimated for any sequence of measurements and the probability that a temporal fault has occurred is subsequently obtained.

From Deterministic to probabilistic modelling.

In [42] authors assume that the normal behaviour of SIS can be modelled through a deterministic Finite State Machine (FSM). This *a priori* model is further transformed to its Input/Output HMM counterpart (So as to account for uncertainties pertaining the physical environment) and the likelihood of a test sequence to have been produced by the model used for measuring the effectiveness of the system online (see Figure 10 [42]).

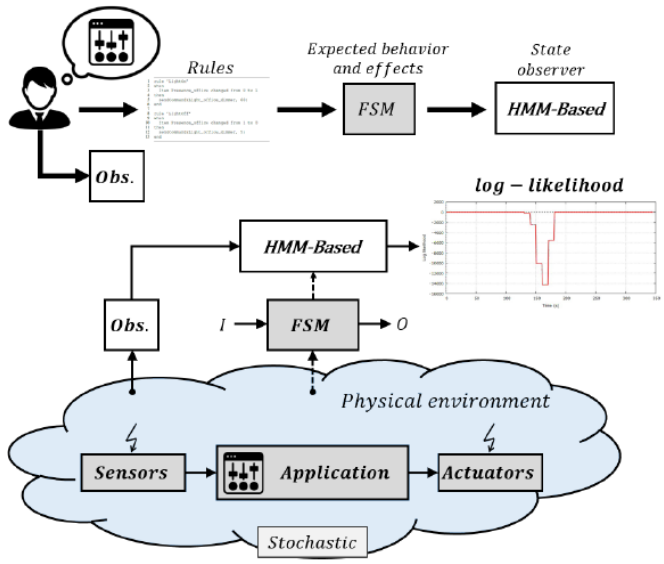

Figure 10: The model of the ideal expected behaviour of an application is specified in terms of the effects it must produce within the physical environment as it executes (Moore FSM). This model is then projected into its probabilistic Input/Output HMM-Based state observer counterpart. The log-likelihood value (]−∞; 0]) computed by the state observer gives direct insight into the behavioural conformity of the application as it executes.

Advantages and disadvantages of the probabilistic modelling techniques.

Interpretability of the models is a key advantage of these techniques. Their main disadvantage is that the interpretation of the likelihood value as a measure of the behavioural drift is difficult. Indeed, given a model, there are numerous observation sequences corresponding to a normal behaviour. Each sequence leads a particular likelihood value which depends on the parameters of the probability density functions it leads to go through in the model. Moreover, the likelihood value highly depends on K (test sequences size) whose configuration is a challenging task (the lower the value, the higher is the probability of occurrence of the subsequence; the higher the value, the lower is the probability of occurrence of the subsequence).

3.1.3.3 Possibilistic Models

Possibilistic models consist in modelling normal behaviour through a parametric model and considering abnormal behaviours as those having low possibility to have been generated by the model.

Compared with the probabilistic approach discussed in section 3.1.3.2, the difference lies in the behavioural uncertainties pertaining to the transducers and the physical environment. Indeed, with possibilistic models, uncertainties are captured through possibility functions (based on fuzzy sets) defining the possibilities for events or physical effects to have a specific value at a given time k.

In [43] authors propose a possibilistic approach for assessing Cyber-Physical Systems (CPS) effectiveness through a Possibilistic Input/Output Hidden Semi-Markov Model PIOHSMM.

Advantages and disadvantages of the possibilistic modelling techniques.

The main advantage of this approach is that the interpretation of the likelihood value as a measure of the behavioural drift is made easier thanks to normalized distributions of possibilities and underlying calculus. Compared with probability distributions, distributions of possibility, through fuzzy sets, are

also more convenient for defining user's preferences. Moreover, the likelihood value (*i.e.*, the degree of possibility) does not depend on the test sequences size K.

3.1.4 Dynamic modelling approaches

An initial model is built from the set $\boldsymbol{S}$ of normal sequences. A test model is further built as test sequences arrive and compared with the initial model. The anomaly score corresponds to the dissimilarity value of both models. Figure 11 provides a summary of the techniques reviewed in this section.

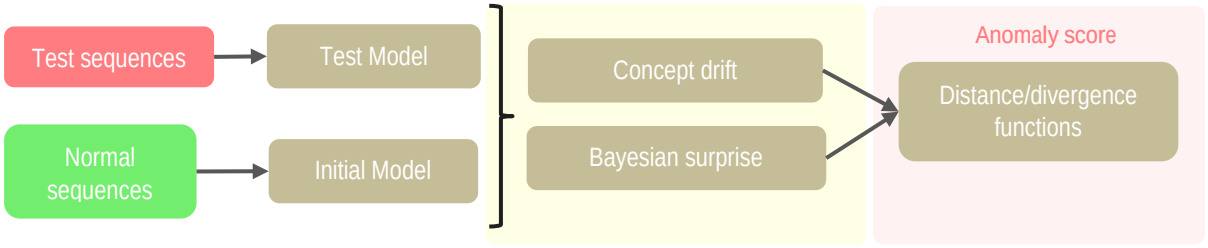

Figure 11: Test sequence anomaly detection based on dynamic modelling approaches

The notion of Concept drift is relative to the unexpected/anomalous evolution of the statistical properties of the model variables. The basic idea is to model the observed behaviour from test sequences as they arrive and compare the obtained test model with the normal behaviour model. Dissimilarities between models give the anomaly score.

Authors in [44] focus on quantitative measure of concept drift and introduce the notion of drift magnitude whose value can be quantified through distance functions such as Kullback-Leibler Divergence or Hellinger Distance. **Here the anomaly is related to the model and not the test sequences** and this approach is used for (1) keeping normal behaviour model up to date and (2) increasing the prediction of the incoming observation values.

Close to the idea of concept drift is **the notion of Bayesian Surprise**. A Surprise [45] quantifies how data affects an observer. It quantifies a mismatch between an expectation and what is actually observed by measuring the difference between posterior and prior beliefs of the observer. In [46] authors propose using Bayesian surprise as a measure of the learning progress of reinforcement learning agents. This measure of surprise produces a "curiosity reward" that spurs agents to behave so as to maintain learning efficiency by favouring regions of their environment leading a surprise (*i.e.*, unknown region leading new information to be acquired) while avoiding "boring" regions (*i.e.*, regions previously sought).

Advantages and disadvantages of the dynamic modelling approaches.

The main disadvantage of the concept drift approach is that the test model accuracy is highly dependent on the amount of observations needed to learn it.

These techniques are depicted in Figure 7 and Figure 11, respectively, and are further described in the sequel. A special attention is put on the interpretability of the techniques and their underlying models, *i.e.*, their ability at making interpretable the behavioural drift value obtained.

3.2 Requirements for Behavioural Drift Analysis enabler

In this section, we present requirements that will drive the development of the behavioural drift analysis associated tools. The distribution of the forecasted tools within the DevOps loop is presented in Figure 12 assuming the following inputs are provided: (1) the model of the physical environment describing dependencies between actuator effects that may hamper the deployed applications expected behaviour; (2) the deterministic model of the SIS expected behaviour and (3) the tolerance model including technological uncertainties pertaining sensors accuracy and epistemic uncertainties relative to how the behaviour of the SIS being observed is judged as being acceptable by the users.

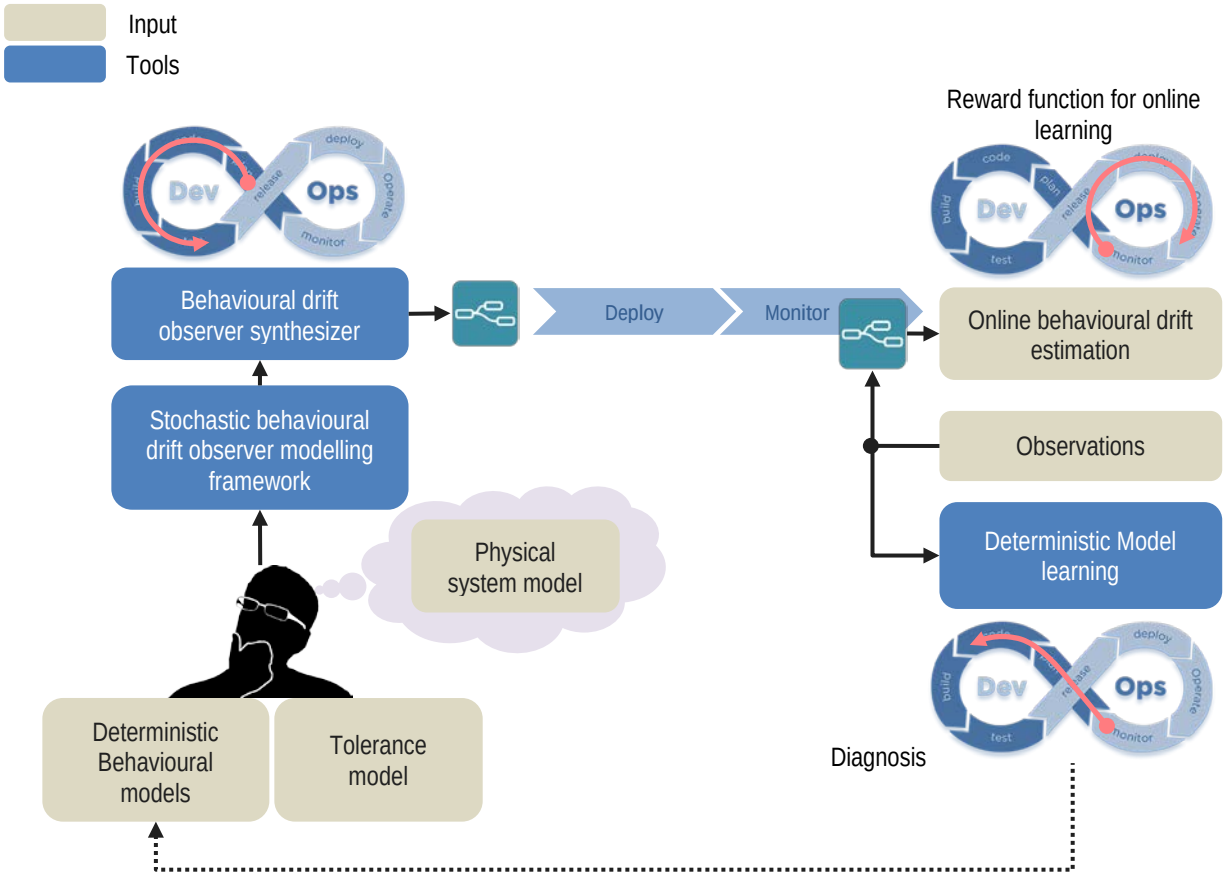

Figure 12: Distribution of the forecasted tools within the DevOps loop.

From the designer perspective, the stochastic behavioural drift observer modelling framework is intended to ease (1) the development of the behavioural drift observer and (2) the interpretation of the quantitative behavioural drift measure.

Table 2: WP3 Behavioural drift analysis requirements

ReqID	Concern	Requirement	Description
R1	Model	Facilitate models' interpretability	The modelling framework underlying the behavioural drift observer shall facilitate models' interpretability
R2	Model	Cope with model variables temporal requirements and transient behaviours	The modelling framework underlying the observer shall cope with model variables temporal requirements and transient behaviours
R3	Model	Facilitate models' elicitation and reusability	The stochastic behavioural drift observer modelling framework shall facilitate models' elicitation and reusability
R4	Model	Define custom tolerances	The designer shall be able to define custom tolerances to cope with natural variability of the sensors readings and uncertainty towards users' satisfaction with regards to SIS behaviour (epistemic).
R5	Operation	Receive feedback from operation	The designer shall receive feedback from operation
R6	Operation	Quantitative measure and normalized, independent of the observer and its underlying behavioural models.	The behavioural drift measure shall be quantitative and normalized ($\in[0,1]$), independent of the observer and its underlying behavioural models.

The following requirements apply:

a. The modelling framework underlying the observer shall:
   i. **Be interpretable by the designer** who needs to understand the relationship between the observed variables and the description of their behavioural requirement within the model,
   ii. **Cope with model variables temporal requirements and transient behaviours**.
b. **The stochastic behavioural drift observer modelling framework shall facilitate models' elicitation and reusability**. One should define SIS expected behaviours from a deterministic perspective, free from any uncertainties (generic behavioural models on the shelf) and obtain their stochastic counterpart by merging the tolerances associated to each defined variable. Sensors to be used (most likely defined in the physical system model) shall be defined in an abstract manner at development time while the concrete sensors shall be defined during operation (reusability, separation of concerns).
c. **The designer shall be able to define custom tolerances** including (1) the natural variability of the physical properties being observed (uncertainties pertaining the physical environment and the sensors), (2) uncertainties (vagueness) relative to how the behaviour of the SIS being observed is judged as being acceptable by the users.

During operation the following requirements apply:

a. **The designer shall receive feedback from operation** helping him (1) to tune the models, (2) to help identifying the underlying root cause of the behavioural drift.
b. Being used as a reward function for online learning**, the behavioural drift measure shall be normalized**, independent of the observer and its underlying behavioural models.

3.3 Conceptual design of Behavioural Drift Analysis enabler

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.

3.3.1 Stochastic behavioural drift observer modelling framework

Table 3 provides a summary of the state-of-the-art underlying modelling approaches used as a basis for estimating an anomaly score (*i.e.*, behavioural drift). Approaches are considered *w.r.t.* the requirements detailed in the previous section.

Table 3: State-of-the-art approaches for behavioural drift score assessment and their compliancy with requirements

<table>
<tr><th colspan="2">Approach</th><th>Interpretability</th><th>Deterministic vs stochastic parameters isolation</th><th>Tolerance</th><th>Discrete & continuous variables</th><th>Normalized score?</th></tr>
<tr><td colspan="2">Recurrent Neural Networks (RNN)</td><td colspan="2">Black box</td><td>No</td><td>Continuous only</td><td rowspan="2">No</td></tr>
<tr><td colspan="2">Hybrid Automata</td><td>Differential equations</td><td colspan="2">Not applicable</td><td>Yes</td></tr>
<tr><td rowspan="2">Dynamic Bayesian networks (DBN)</td><td>Probabilistic</td><td rowspan="4">Good</td><td>Mean/variance* (@state & state-transitions)</td><td>Uncertainty (variability)</td><td rowspan="4">Continuous only</td><td rowspan="2"></td></tr>
<tr><td>Possibilistic</td><td>Core/boundary* (@state& state-transitions)</td><td>Vagueness (preference)</td><td>Yes</td></tr>
<tr><td rowspan="2">Stochastic Petri nets (PN)</td><td>Probabilistic</td><td>Mean/variance* (@state & state-transitions)</td><td>Uncertainty (variability)</td><td>No</td></tr>
<tr><td>Possibilistic</td><td>Core/boundary* (@state & state-transitions)</td><td>Vagueness (preference)</td><td>Yes</td></tr>
</table>

* Probabilistic models are often based on probability density functions (pdf) representing normal probability distributions described through mean and variance/covariance parameters. For their part, possibility distributions are described through core (representing values of the variable for which the possibility value equals to 1) and boundaries (representing values of the variable for which $0 < possibility \leq 1$).

Probabilistic and possibilistic graphical models (DBN and PN) exhibit good compliancy with requirements. There exist many DBN/PN derivatives. Probabilistic/Possibilistic DBN-based behavioural drift analysis are covered in [42] and [43], respectively. We do plan to investigate on stochastic PN [44-46] derivatives for their ability to manage concurrent processes.

3.3.2 Behavioural drift observer synthesizer

The purpose of the synthesizer is to produce the concrete behavioural drift observer to be deployed by GeneSIS. Here, the plan is to rely on the **flow-based programming paradigm (FBP)** [50] where concrete applications are compound of components (black boxes) described by their interface (input ports, output ports and properties). This approach facilitates the separation of concerns, the components connection network being specified externally to the components (see Figure 13). A good representative of this approach in the Internet of Things (IoT) domain is Node-RED where components are nodes.

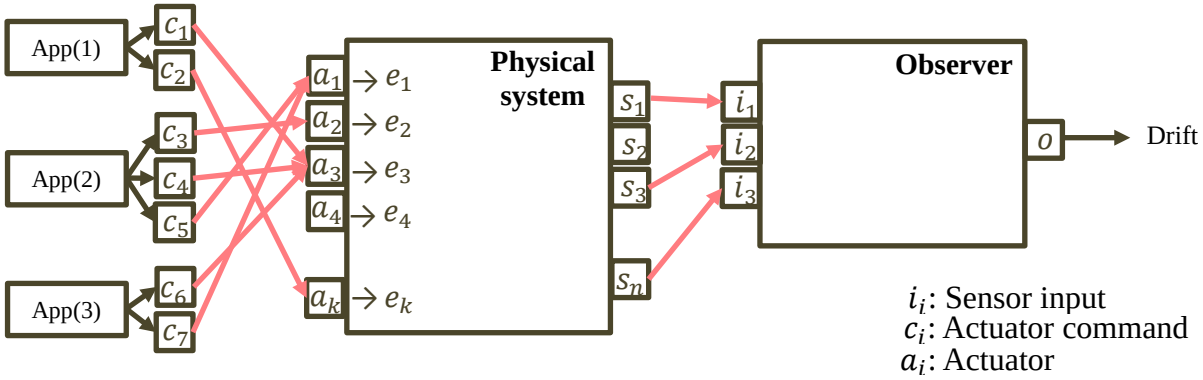

Figure 13: Flow-based programming example

3.3.3 Deterministic Model Learning

To help the designer to facilitate the tuning of the models considered at development time, we do plan to make evolving the observer stochastic model at run-time (test model). For this purpose, Concept drift and Bayesian surprise approaches along with their associated model learning algorithms shall be leveraged. Our objective here is to derive the deterministic counterpart of the stochastic test model (most likely in the form of timed automata, free from uncertainty parameters) and feedback it to the designer. Besides potentially exhibiting parameters drift, it could also make appear unexpected transitions and states, whose identification could (1) help analysing root cause of the drift on the long run, (2) help making the initial model more accurate.

4 Root-Cause Analysis for Smart IoT Systems

Root-cause analysis (RCA) objective is to infer, given a set of observations that contain some types of anomaly, what is causing this observed anomaly by going back through the causal chains governing the system under analysis. It is possible to observe problems like a behavioural drift among others during the operation of a SIS. But to help the developer to fix the problem during development, it is important to pinpoint the root-causes of the problems as much as possible. This is the purpose of the next section.

4.1 State-of-the-art on Root-Cause Analysis

RCA is an old field that predates computer science as the need for systematic troubleshooting is intrinsic to any complex system. Although there is a vast literature on RCA, we will restrict to techniques that can be applied to IT systems. This condensed state-of-the-art will provide a general framework to understand RCA techniques, for more in-detail explanations of each of the methods, other general surveys are available [23] [51]. For specific areas, specific surveys exist for computer networks [52], software [53], industrial systems [22], smart buildings [54], (regular) buildings [55], machinery [56], swarm systems [57], automatic control systems [58] [59], automotive systems [60] [61] and aerospace systems [62].

We will first introduce a general framework to understand any RCA technique that we will use to explain the challenges and how different families of methods address these challenges. Then, in the last part of the section, we will talk about specific challenges and techniques for IoT.

4.1.1 General framework for Root-cause analysis

There are three key ingredients to be able to perform a successful RCA (Figure 14):

- **Domain Knowledge**, the laws that govern the system (*e.g.*, the laws of mechanics for a mechanical device). Obviously, except from mathematical abstractions, anything would have to satisfy the laws of physics, but it is understood that domain knowledge refers to the rules that govern the system at an abstraction level that makes these rules practically applicable (*e.g.*, no one will use quantum mechanics equations to troubleshoot a printer, even though the printer must obey those rules, as an easier and more tractable set of rules is enough to model the device for that particular purpose).
- **System Knowledge**, the elements that comprise the system and their relationships (*e.g.*, the different components of the printer and their relations, like relative position, elements in contact, groups of elements related to the same function, etc.)
- **Observations**, observable data coming from the system (*e.g.*, printed pages from the printer, information from its internal monitoring sensors, history of previous failures, etc.)

These three elements allow making inferences on what could be happening in the unobserved parts of the system, predict future observations and potentially solve the RCA problem, among other things.

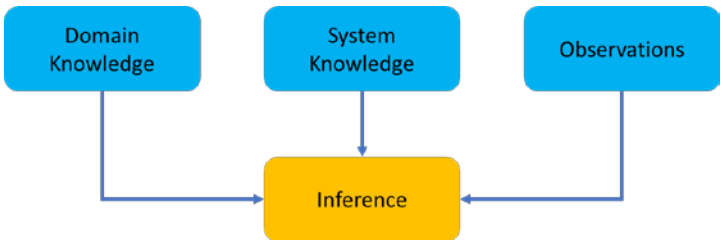

Figure 14: The key ingredients for inference in a system

The different approaches to RCA differ on how they model each of these parts (or merge them in larger models). For instance, observations can be assumed to be noiseless or noisy, giving rise to deterministic RCA methods (*e.g.*, using Propositional logic [63], fault trees [64]) in the former case or to probabilistic methods (*e.g.*, using Bayesian networks [65], Markov logic networks [66]) in the latter.

In a similar fashion, System Knowledge can be perfect or flawed. Assuming one or the other distinguishes between methods that can update (or suggest fixes) to current system knowledge and others that might simply give wrong answers in the presence of incorrect or missing facts. Methods that can handle inexact system knowledge are necessarily more complex and not as popular in research as the ones assuming perfect information, although in many fields this is a frequent source of problems. For instance, in large IT industrial environments it is frequently assumed by administrators that the corresponding Configuration Management Database (CMDB) describing the infrastructure can contain large number of errors or omissions[6].

Finally, Domain Knowledge can be given as an input to the method (this encompasses *model-based* methods as they are sometimes called in the literature) or automatically derived using Machine Learning (ML) techniques. The latter techniques are sometimes called *model-free*, although *automatically generated model methods* would describe more precisely what they actually do. Note that the application of ML can yield models that include both the Domain Knowledge plus the System Knowledge. For instance, an assumption that is sometimes done when diagnosing a black-box system (or a system that people do not understand, *i.e.*, they do not have a model for it), is that similar symptom patterns are related to similar root-causes. In this case, a potential way to proceed is to use a nearest neighbours classifier that maps the symptoms to an N-dimensional space and compares to previously seen symptoms for which we know the root causes. In this case, the model consists of the labelled points, and the N-dimensional space both implicitly contains the information about the domain knowledge and the

[6] https://www.slideshare.net/ITInvolve/overcoming-the-challenge-of-cmdb-inaccuracy

system knowledge (*e.g.*, if the printer has two trays, it might have two variables, one per tray sensor. Should it had only one tray, only one variable will be present).

Model-based methods (*i.e.*, with a human curated input) have many advantages due to the explicit representation of the causal chains, like the capacity to react to previously unseen failures, including black swans (*i.e.*, infrequent rare events). However, they have two major disadvantages: (i) knowledge must be extracted from experts, a process known as *knowledge elicitation*, which is a very costly and slow process [67], and (ii) large models or knowledge bases are difficult to maintain and validate. On the other hand, automatically generated model methods rely only on either large quantities of data and/or that the generalization capabilities of the learnt models are similar to the real causal model in order to deal with unseen cases. There is a whole line of research in ML that tries to incorporate as much domain knowledge available to restrict the models learnt [68], *i.e.*, learning to predict the next position of a ball considering some very basic physic laws that have low cost of elicitation, so that no expert is required and the knowledge base is small enough so that maintainability and validation aspects are easy to solve.

Not all ML techniques necessarily learn the System Knowledge, some of them can work on the Domain Knowledge alone or reduce the amount of System Knowledge by considering a variable length input. For instance, when working with language, the inputs (*e.g.*, sentences) can typically have variable length. Or if one models an IT infrastructure as a graph, graphs can be arbitrarily large. A frequent solution is to work with models that can handle sequences (like the sequence of neighbours of a node in a graph [69]).

Observations in IT come usually from monitoring tools specialised in gathering them and providing unified access to the data. Collected data can be of a wide variety of types, from very structured, like a specific known set of metrics, to almost unstructured such as logs. These observations can be noisy for many factors, including low accuracy of sensors, bad conditions for readings, or simply because of the own particularities of the monitoring system setup: for instance, unsynchronized clocks, different granularities of collection, variable lags, etc., can alter the temporal order of event observation. This is a relevant issue because temporal sequence is often used to restrict feasible causalities (*i.e.*, a consequence cannot precede its cause), and causality discovery is essential for an informed root cause analysis. As a result, some techniques include the possibility to look for correlations between events both in the future and in the past [70].

In complex systems, failures usually propagate through causal chains and produce evolving fingerprints of noisy symptoms. One of the first tasks to accomplish for an automated tool helping humans troubleshoot a system is to group events that are causally connected (and keep unrelated events separated). This might not be straightforward to achieve since components of a system can exhibit symptoms of two unrelated problems. A naïve Bayesian approach is sometimes used, in which problems are assumed to have independent probability of happening (something reasonable to some extent if the problems are not causally related). In such scenario the probability of concurrent problems can be very low. This has been used in many RCA approaches to limit the number of concurrent problems to a single one [71] or a small number of them [72], since it can make the problem more computationally tractable.

In practice, large deployments can exhibit many concurrent problems, and many more apparent problems, especially when noisy sensors (or, equivalently, alarm conditions known to have false positives) are present. It is frequent that some of these problems are recurrent and either the administrators of the system have already some recurrent mitigation action (*e.g.*, reset a particular server every night) or that simply they are not relevant enough to be paid attention. In any case, such situations must be identified, since administrators prefer to focus on the “new” problems than the rather old and already known ones. This fact is frequently overlooked in academic approaches to RCA, since they focus on finding the correct problem or set of problems, not on how to reduce and prioritize these findings when there are lots of problems in the system. This might be caused by experimenting with small systems compared to the sizes of large industrial deployments. This is problematic because the IoT world was born with the promise to connect everything, thus potentially creating huge systems.

In contrast, industrial approaches have favoured solutions with low computational cost that can work at a scale. For instance, decision trees were used for RCA in high-volume production environments at eBay [73]. Obviously, techniques with better running time in general produce less accurate results than more

sophisticated but expensive methods, but in these high-volume massive environments having an approximated indication on what could be the potential causes for most of the failures is better than very precise analyses of each one of the failures. There have been attempts to use more computationally complex models by improving their running times using the usual performance improvement strategies: parallelizing [74] or using hardware acceleration (*e.g.*, using GPUs [75] or FPGAs [76]).

4.1.2 RCA for IoT

Where does IoT fit into this generic framework for RCA? First, it might be a very dynamic environment in some cases, with devices joining/leaving a system in relatively small amounts of time (*e.g.*, mobile devices connecting to a particular antenna). Second, most of the communications are likely to be wireless, introducing a higher degree of unreliability that can be difficult to distinguish from the normal dynamicity mentioned in the previous point (*i.e.*, we are not receiving information from the sensor because the sensor is no longer in range or because communication failed?). This can affect not only the diagnosed system, but the diagnosing system, as observations can get lost and/or affect RCA computation if the RCA is done partly at the edge (using the own diagnosed system). Third, depending on the scenario, the number of components that would have to be considered in the diagnostic could be very large. Finally, battery-powered devices may have a low-activity profile to extend their autonomy of operations, meaning that their inputs might not be synchronized and have the same frequency as other information the RCA can use for diagnosis, worsening the problem of observation noisiness and out of order data.

The first problem, dynamicity, creates a lot of pressure on keeping the system knowledge up-to-date. Considering that in IT environments with less dynamic environments this is already a problem, clearly techniques should adapt to include the possibility to model explicit failure of System Knowledge (*e.g.*, include the possibility in a Bayesian Network that components are no longer present, thus undefining all their variables). The second problem, unreliable communication, has been either modelled by explicitly incorporating the possible misses of messages (particularly in the work of computer networks [77]) and by resilient actor-based mechanisms [78].

To deal with scalability, the third challenge mentioned, there has been work in splitting the computations into components that could be reused (*i.e.*, reusing as much as possible diagnostics from similar devices) [79], or use simpler models, as we have seen before in industrial settings.

Some interesting work focuses in creating self-adaptive decision support trees based on streamed data including change detection, to tackle both scalability and high dynamicity challenges. This work assumes that the amount of information is so large that the data stream can be considered continuous and in general it is not possible to read data more than once. Differently from classical methods such as C4.5, these algorithms do not assume that all training data are available simultaneously in memory and deal with change over time. In particular, different methods have been proposed based on Hoeffding Trees or Very Fast Decision Tree method (VFDT) [80]. For instance, CVFDT is an adaptive variant of VFDT proposed by Hulten et al [81]. Adaptive Hoeffding Trees were later proposed in [82] for the same purpose but detecting change and updating the decision tree based exclusively on data analysis. Attempts to parallelize the creation of a decision tree for heavy streams are implemented in Apache Samoa[7].

Some industrial approaches rely on keeping the tracked relationships in the system knowledge at a very high abstraction level, for instance, tracking only which elements of the system call each other. In this setting, it is possible to identify the likely culprits for performance degradations using simple heuristic rules. For instance, since the running time of a call causally depends in part on the running times of the nested calls, looking for the "deepest" node in the call hierarchy that is exhibiting some anomaly is likely to provide a good culprit candidate. This candidate can be then used as a starting point for a human operator to further investigate. However, the increasing adoption of more decoupled complex architectures with messaging queues, bidirectional communication channels, peer-to-peer protocols, etc. blurs the definition of what a business transaction[8] is and what are its "deepest" nodes (Figure 15). One

[7] Apache Samoa: https://samoa.incubator.apache.org
[8] By *business transaction* we mean the sequence of executed actions in a system in response to a functionality that the business needs to provide. *E.g.*, all the code executed in a distributed system that deals with a "user login", "chart checkout", etc.

possible way to address this issue is to increase the amount of system knowledge: no longer use homogeneous black box nodes to represent the system, but identify the architectural roles they play (*i.e.*, if they are a message queue, a server using sockets to push notifications, …) and then add the corresponding domain knowledge to extend the basic heuristic rules to these more refined scenarios. However, this approach has much more complexity than the simple heuristic method and, thus, a trade-off between scalability and precision of results is established.

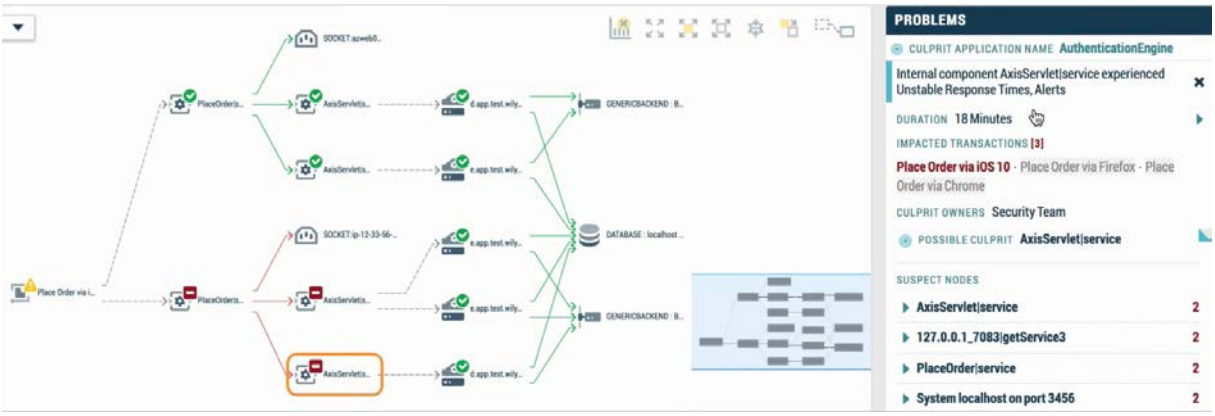

Figure 15: An industrial diagnostic tool, blaming one of the deepest nodes in the call hierarchy for the performance degradation in a business transaction.

4.2 Requirements for Root-Cause Analysis enabler

In terms of capabilities that the RCA enabler must provide, they are linked to the challenges outlined for IoT in the previous section, namely:

Table 4:WP3 RCA enabler requirements

ReqID	Requirement	Description
R1	Being able to cope with highly dynamic environments	Establishing a mechanism that can diagnose regardless of the particular components involved (so, reasoning at a higher abstraction layer in a way that is able to deal with fuzzy similarities), eventually considering changes in the system composition itself as potential root-causes.
R2	Support for Inaccurate System Knowledge	Identify situations where different observed anomalies or problems may be observed in different components in the system that are apparently too far to be part of the same problem (typically because of lack of the system knowledge that would make their relationship explicit, like a shared resource), but that, nonetheless, a historical correlation between them can be established so that they can be merged together even if the System Knowledge necessary is not available.
R3	Explicitly consider communication unreliability	In the most extreme cases subparts of a larger system may become disconnected and still some diagnostic capability has to be retained.
R4	Compensate for noisy and out-of-order data	Observed events in two similar problematic situations may occur in slightly ***different order***. Even if they happen in the same order, the monitoring system may capture them in the ***wrong order***, depending on different sampling granularities for instance. However, both situations may have the same root cause and may end up leaving the system in the same final step. We need to be able to realize that these situations are similar. ***Evidence noise*** also needs to be taken into consideration.
R5	Prediction of problem evolution and its impact	Although this goes beyond the pure RCA, in large systems where many anomalies are happening most of the time, knowing the root-cause of a problem might be less important than determining the potential impact of each one of the current

		problems. Typically, this could be achieved by mining previous problem evolutions considering appropriate impact metrics.

To accomplish these goals the RCA enabler needs to have access to:

- Some basic system knowledge (*i.e.*, what devices are on our IoT environment and how they relate to each other, but where the relationship information may not be completely accurate).
- Monitoring metrics from the IoT devices.
- An anomaly detection mechanism that will flag anomalous situations in the system, *i.e.*, the observable symptoms.

Note that in description of task T3.3 the automatic creations of models for IoT devices is mentioned, but this particular task has been moved inside the simulation efforts of task T2.4.

4.3 Conceptual design of Root-Cause Analysis enabler

The high-level architecture of the RCA enabler is shown in Figure 16. The diagnoser receives monitoring information from the IoT system through one or more data collectors, depending on the number of devices in the system. It then uses this information, together with the historical information of previous and currently open incidents to either create new incidents or enrich/merge/split existing ones. The creation of a new incident occurs when we are observing anomalies that are caused by some problem that was not currently present in the system, although it might have been previously observed. When new anomalies are in fact expected extensions of currently detected problems, then the diagnoser will enrich a currently open incident. While it will merge/split incidents if new evidences suggest that current anomalies could establish a link between currently open incidents or that the observations are better explained by two independent problems.

Figure 16 shows one of the potential ways in which information can reach the data collector. In this particular case an IoT device executes code that has been instrumented with one or more *probes*. Probes are just an abstraction that can include a wide range of forms such as automatically instrumented bytecode, manually inserted log actions in the code, system performance metric capture (*e.g.*, CPU load, memory used), etc. Probes report data back to an agent that runs locally in the device, aggregates all the information of the probes and periodically sends the information to the data collector. Agents may be considered as an example of edge computing, since they may perform some basic computations on the received data (*e.g.*, send aggregate data on windows of fixed size) to reduce the amount of bandwidth required to transfer all the data to the diagnoser.

Finally, the user (typically a System Operator/Administrator) will connect to the Root cause analysis dashboard (the Diagnoser in Figure 16) where the prioritized list of open incidents will be available, with the current likely culprit and potential evolution, based on the incidences observed so far.

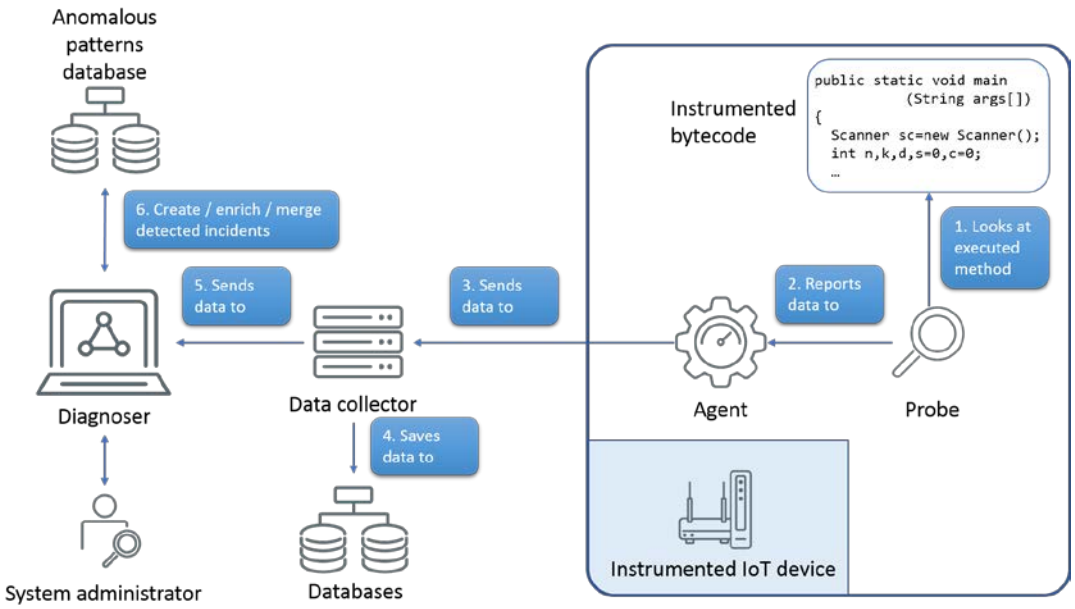

Figure 16: High-level design of the RCA enabler architecture

Section 4 has described the RCA enabler, which is the last of the three enablers explained in this document. In next section we will look at how these enablers can work together.

5 Support and Interrelationships among Techniques for Agile Operation of Smart IoT Systems

Techniques can work in isolation, but we expect enhancements that improve their capabilities by exploiting the interrelationships among them as described below.

5.1 Adaptation Enactment as Support for Self-Adaptation

As depicted in Figure 2, the enactment layer is responsible for adapting the running system based on the actions from the management layer. In turn, it provides the management layer with abstractions (i) to facilitate their understanding and reasoning activities as they can analyse a simplified version of the running system, and (ii) to facilitate the actual adaptations of the system (*i.e.*, trigger high-level adaptation commands, which, in turn, are transformed in atomic modification actions of the system). In addition, the introduction of this layer improves the separation between the running system and the adaptation layer. This separation of concerns brings two main benefits: (i) easier integration of reasoning engines (*i.e.*, the reasoning engines do not need to be tailored to the running system details and implementation), and (ii) easier integration of execution environments (*e.g.*, this layer can abstract and hide differences between the operation and test environments making it easier to add a new one). In the rest of this section we introduce state-of-the-art on mechanisms to enact adaptation before we detail the conceptual design of our Adaptation Enactment layer.

5.1.1 State-of-the-art on Adaptation enactment

For some years now, multiple tools are available on the market to support the continuous deployment of software systems – *e.g.*, Puppet[9], Chef[10], CFEngine[11]. 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 [83] (*i.e.*, systems deployed across multiple cloud solutions from different providers) such as CloudMF [84], OpenTOSCA [85][12], Cloudify[13], Brooklyn[14], Kubernetes[15]. On the other side, few tools such as resin.io[16] and ioFog[17] are specifically designed for the IoT. However, so far, these tools have not specifically considered deployment across the whole IoT, Edge, and cloud space. Even if, some tools are now being extended toward Edge infrastructures– *e.g.*, Kubernetes, OpenTOSCA [85]. 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.

All these platforms provide different levels of support for the adaptation of the system. It is worth noting that such support typically includes two activities: (i) the monitoring of the system, and (ii) the actual enactment of the adaptation of the system. Regarding monitoring, these platforms typically offer mechanisms to monitor the status of the deployment process but very few of them provide mechanisms to monitor the status and execution of the system once deployed. Moreover, these tools do not provide mechanisms to integrate and reflect directly the monitored run-time information into the deployment model (as a way to provide feedback to the developers). This is often due to the fact that the monitoring activity is delegated to a monitoring platform. However, even then, to the best of our knowledge, none of them provide mechanisms to monitor the execution flow of the deployed software components. This is particularly important for the development and maintenance of the software components to be deployed on resource constrained IoT devices, as it is often difficult or even impossible to access and store logs on such devices. Regarding adaptation enactment, there are two main approaches: (i) tools that require the re-deployment of the whole system for each adaptation and (ii) tools that allow re-deploying only the part of the system that accounts for the modification. In addition, main adaptation features include auto-scaling (*i.e.*, scaling in and out) and bursting (*i.e.*, migration from one cloud to another) capabilities.

In parallel, recent research work has also focused on the deployment and orchestration of IoT systems. In the Deliverable 2.1, we have presented the state-of-the-art of the existing deployment and orchestration approaches for IoT (see Appendix A), including our analysis on how deployment and orchestration approaches support the adaptation enactment aspect: monitoring, run-time adaptation, shared access to resources, which we define as follow:

- Monitoring is a key activity to reason on the state of a system and for controlling and managing hardware as well as software infrastructures [86]. Monitoring probes are used to deliver information and indicators characterizing the system and the context in which it is running. The purpose of the monitoring can be multiple. In this study we identify the following: (i) measuring QoS of the system, (ii) capturing the status and health of a deployment, (iii) depicting the state of the environment, and (iv) observing the execution flow of the system, for instance for debugging purpose. On the one side, it is a key tool for controlling and managing hardware and software infrastructures; on the other side, it provides information and Key Performance Indicators (KPIs) for both platforms and applications.

[9] http://puppet.io
[10] http://chef.io
[11] https://cfengine.com
[12] https://www.opentosca.org
[13] https://cloudify.co
[14] https://brooklyn.apache.org
[15] https://kubernetes.io
[16] https://resin.io
[17] https://projects.eclipse.org/proposals/iofog

- Adaptation: there are two main approaches for dynamic adaptations [87]: (i) parametric adaptation and (ii) compositional adaptation. Parametric adaptation allows modifying the system's behaviour by tuning parameter values. This type of adaptation requires the adaptation parameters to be defined at design-time. By contrast with parametric adaptation, compositional adaptation allows the "hot" deployment and binding of software components that were not necessarily foreseen before the initial deployment of the system. It is worth noting that both types of adaptation can be seamlessly combined. Orchestration and deployment tools can support these two types of adaptation for modifying: (i) the application itself (*e.g.*, replacing one software component by another) and (ii) its infrastructure (*e.g.*, bursting from one cloud to another).
- Shared access to resources. Because IoT systems may involve actuators it is important to control the impact these actuators can have on the physical world and to manage conflicting actuation requests. More generally, this applies to the management of shared accesses to resources, which can be of two types: (i) direct concurrent access to resources (*e.g.*, several entities accessing to the same service/actuator) or (ii) indirect shared access to a resource (*e.g.*, actuators from different applications are modifying the temperature with possibly conflicting goals) [88].

As part of the results, Figure 17 shows that few out of 69 primary studies (Appendix A) provide advanced supports such as monitoring, run-time adaptation, shared access to resources. It is worth noting 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 support only deployment (9, 8) or orchestration (6, 7). Shared access to resources is a rare feature in the existing approaches. As discussed in Appendix A , 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. A bootstrap is a basic executable program on a device, or a run-time environment, which the system in charge of the deployment relies on (*e.g.*, Docker). This aspect relates to the specificity of a solution and its dependency to particular technologies [89]. The primary studies are also immature in addressing trustworthy aspects and advanced supports, such as Adaptation enactment, which we aim to address with our Generation and Deployment of Smart IoT Systems (GeneSIS) framework.

Figure 17: Advanced supports mentioned by the existing deployment and orchestration approaches

To analyse deeper in technical details, we selected 17 key studies (see Table 8) out of 69 primary studies that propose both deployment and orchestration for IoT. Table 5 shows the 10 key deployment and orchestration studies for IoT [87-97] that have mentioned supporting for the adaptation of the deployed IoT (***Ada.*** column). After analysing these 10 studies in detail, we find that most of the adaptation in these approaches (six approaches) is of type Compositional. It is worth noting that both types of adaptation can be seamlessly combined. Indeed, we found three studies that have both types of adaptation, *i.e.*, orchestration and deployment tools can support these two types of adaptation for modifying: (i) the application itself (*e.g.*, replacing one software component by another) and (ii) its infrastructure (*e.g.*, bursting from one cloud to another).

Table 5: How the deployment and orchestration approaches for IoT address 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;*

Table 5 also shows that shared access to resources is indeed 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 (addressed in D2.1). The uncertain, dynamic, and partially known nature of the physical environment makes it very difficult to assess at run-time the conformity of the effects of actions in this environment with deterministic models (3.3).

Three studies ARCADIA [S2], Foggy [S16], and WComp [S9] that mention monitoring support also provide adaptation support. ARCADIA [S2] and Foggy [S16] have provided technical details of their monitoring approaches, such as the Resource Monitor module of Foggy can monitor the resource usage and dynamically adapt container placement, whilst ARCADIA [S2] monitors the application to trigger lifecycle management actions (such as relocation, scaling, and termination). WComp [S9] simply monitors the appearance and disappearance of smart things in the environment, and then allows to automatically and dynamically compose multiple applications sharing common services according to the context evolution. Among the approaches that have adaptation support without monitoring, the purposes of adaptation vary. For example, Calvin can update the actor placements and auto-scale to handle a varying workload or to replicate actors onto the available runtimes. TOSCA-BMWi [S17] leverages the OSGi framework Equinox, which allows to add and start new plugins, even during the runtime of the service bus. Some other approaches provide adaptation support for dynamic orchestration such as FogTorch [S1], SoPIoT [S10], Cloud4IoT [S12], BeC3 [S5] and D-LITE [S4]. SAaaS [S6] does not touch trustworthiness aspects and monitoring, adaptation, or shared access to resource at all.

About half of the studies (eight out of 17) have provided tool support for full automation in deployment and orchestration process. Another half have semi-auto level in tool support, *e.g.*, the bootstraps cannot be installed automatically. Only one study (by Verba etal. [S15]) has manual level in tool support, which means the tool has not been implemented but only the tool framework proposed.

5.1.2 Requirements for Adaptation enactment

In this section, we first present the high-level requirements that will drive the development of the mechanism to enact adaptation. Then Table 6 introduces more concrete requirements from WP3 for the GeneSIS run-time engine.

- **Abstraction and Infrastructure independence**: GeneSIS 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.
- **Automatic deployment and dynamic adaptation:** From a GeneSIS deployment model, GeneSIS shall support the fully automatic deployment of SIS over IoT, edge and cloud resources. In addition, the deployment of a system should be dynamically adaptable with minimal impact over the running system (*i.e.*, only the necessary part of the system should be

adapted). The deployment and adaptation API exposed to the users should be technology agnostic and as much as possible device- and platform-independent.

Table 6: WP3 Requirements for GeneSIS

ReqID	Requirement	Description
R1	Scalability	GeneSIS should be able to deploy SIS involving hundred sensors/actuators.
R2	Scope	GeneSIS should be able to deploy SIS involving IoT (*e.g.*, Arduino board), edge (*e.g.*, Raspberry Pi) and cloud infrastructures (*e.g.*, Amazon EC2).
R3	Scope	GeneSIS should be able to deploy SIS on specific hosts using at least SSH connection, container technologies (*e.g.*, Docker), and serial port (*e.g.*, for Arduino boards)
R4	Adaptation/Trustworthiness	GeneSIS should be able to update (install new version of a software node) software components deployed on IoT, edge, and cloud infrastructure.
R5	Adaptation/Trustworthiness	GeneSIS should ensure that the "adapted software" still properly interacts with the rest of the system.
R6	Monitoring	GeneSIS should provide and deploy the necessary mechanisms to monitor the deployment and the health of a SIS. Moreover, when a ThingML program will be deployed, its execution flow should be monitored.
R7	Adaptation/Trustworthiness	GeneSIS should be able to migrate software specified using ThingML from one host to another, including between the IoT, edge and cloud spaces.
R8	Trustworthiness	GeneSIS should be able to deploy trustworthiness mechanisms defined by WP4 enablers.
R9	Scope	GeneSIS should offer an API to trigger a deployment/adaptation from a new GeneSIS deployment model
R10	Scope	GeneSIS should offer an API to trigger high level adaptation actions (*i.e.*, update software, migrate software, deploy software)

In the following section, we first present the current initial conceptual design of the GeneSIS run-time engine before we introduce future work.

5.1.3 Conceptual design of Adaptation enactment

GeneSIS includes: (i) a domain specific modelling language to model the orchestration and deployment of Smart IoT Systems across the IoT, edge and cloud spaces; and (ii) a run-time environment to enact and adapt their actual orchestration and deployment. In the following, we first introduce a simple illustrative example that we use in the rest of the section to detail the conceptual design of the GeneSIS execution engine. Please note that the description of the GeneSIS modelling language can be found in deliverable D2.1.

5.1.3.1 Illustrative example

SensAct is a company delivering application for smart building that connects IoT sensors and actuators to analytics services running in the cloud. SensAct must develop and deploy a new SIS in a house 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 18. 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 locations of the house. This device is plugged to a Raspberry PI (called RaspberryPI2 in Figure 18) 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 RaspberryPi in Figure 18). 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.

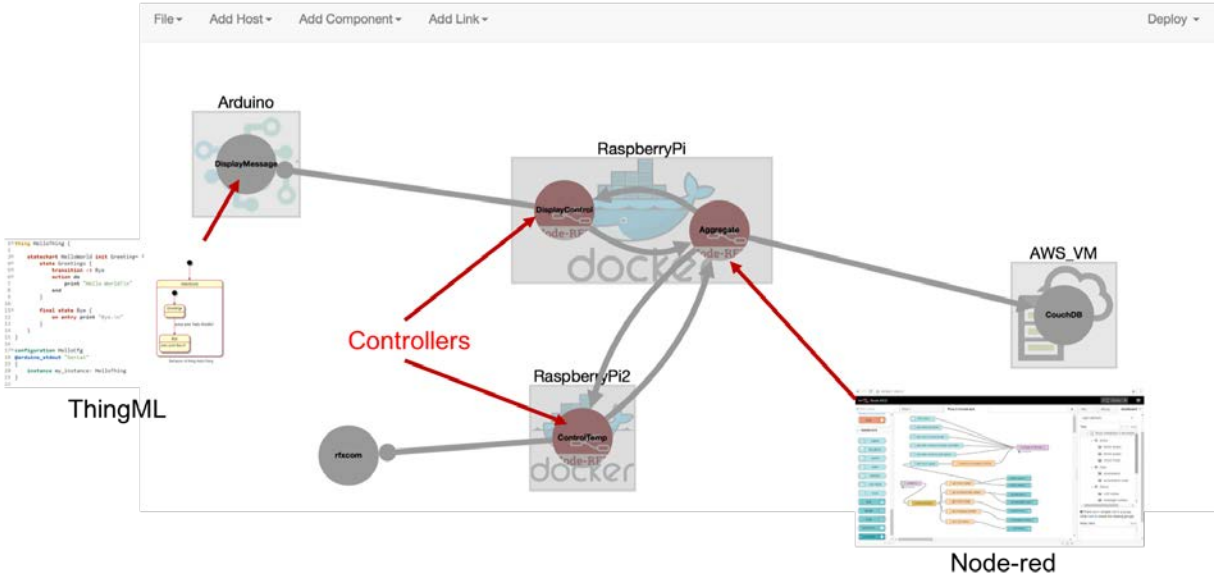

Figure 18: Illustrative example

5.1.3.2 The Models@Runtime pattern

The GeneSIS run-time environment consumes a GeneSIS deployment model and, from it, computes how to deploy or adapt the IoT system before actually enacting this deployment or adaptation. GeneSIS thus leverage model-driven engineering techniques. Model-Driven Engineering (MDE) techniques have shown to be effective in supporting design activities [101]. MDE 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. Models and modelling languages as the main artefacts of the development process enable developers to work at a higher level of abstraction rather than at the level of implementation details.

However, as stated in [102], applying the classical MDE approach for software evolution would be impractical. Indeed, this would typically result in generating the new solution, stopping the running system before replacing it by the new one, this in contrast with common expectations for Cloud services to have more or less 100% up-time. To address this issue, the Models@Runtime approach has emerged.

Models@Runtime [102] is an architectural pattern for dynamic adaptive systems that leverage models as executable artefacts that can be applied to support the execution of the system. Thus, Models@Runtime can be applied to reduce the developer-operator gap by providing a unique model-based representation of the applications that can be applied for both design- and run-time activities. As depicted in Figure 19, Models@Runtime enables to provide abstract representations of the underlying running system, which facilitates reasoning, analysis, simulation, and adaptation. A change in the running system is automatically reflected in the model of the current system. Similarly, a modification to this model is enacted on the running system on demand. This causal connection enables the continuous evolution of the system with no strict boundaries between design-time and run-time activities.

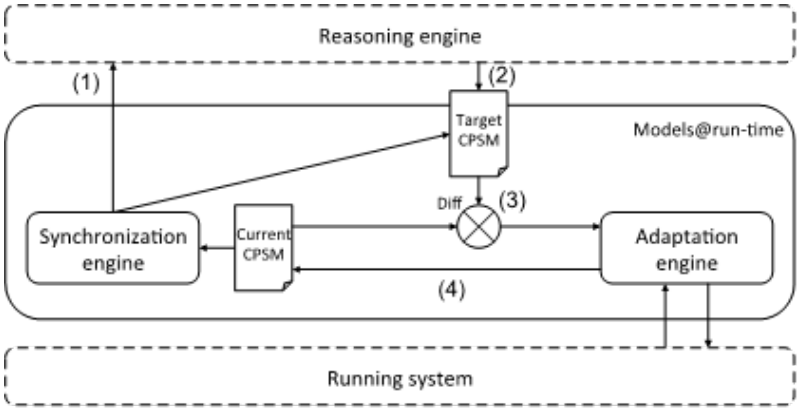

Figure 19: The Models@Runtime pattern

Exploiting Models@Runtime for continuous deployment typically result in the following process. A DevOps team can specify a model of the deployment of its application (typically exploiting a domain-specific language) and thus automatically enact this deployment into a test environment. The team can therefore benefit from this test environment to tune its development and redeploy it automatically. Any change made to the deployment model will be enacted on demand on the running system whilst its status will be reflected in the model providing useful feedback. Once a new release ready, the team can tune this model to maintain and manage the running system. This approach fits well with the DevOps principles as the model used to specify the deployment of the system is enriched after deployment with run-time information thus providing developers with feedback about the operation of the system in the language they are familiar with.

The GeneSIS deployment engine implements the Models@Runtime pattern to support the dynamic adaptation of a deployment with minimal impact on the running system.

5.1.3.3 The GeneSIS run-time environment

From a deployment model specified using the GeneSIS modelling language (see D2.1), the GeneSIS run-time environment 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.

Figure 20 depicts the architecture envisioned for the GeneSIS run-time environment. It can be divided into two main elements: (i) the facade and (ii) the deployment engine. At the time of writing this document only a first version of the deployment engine has been implemented.

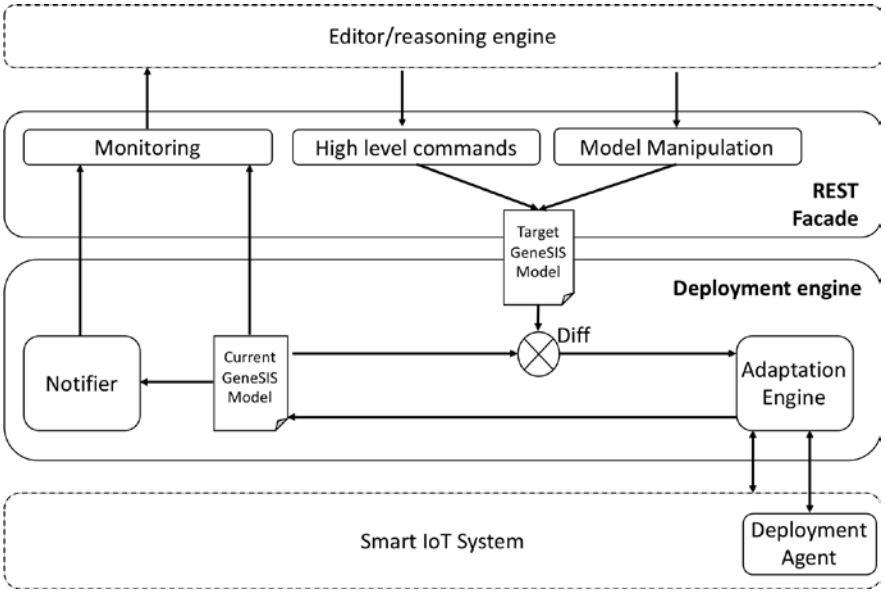

Figure 20: Architecture of the GeneSIS execution environment

The facade provides a common way to programmatically interact with the GeneSIS execution engine via a set of three APIs. The monitoring API offer mechanisms for remote third parties (*e.g.*, reasoning engines) to observe the status of a system. Third parties can either retrieve the whole GeneSIS model of the running system enriched with run-time information or subscribe to a notification mechanism. For the latter, the communication is achieved using the WebSocket protocol[18], which enables light-weight communications. We have defined the standard Meta Object Facility (MOF) reflection modifications as the primitive notification events, which are encoded as plain text.

The high-level commands API exposes a pre-defined set of high-level commands that avoid direct manipulation of the models (*i.e.*, the model is automatically updated when the command is triggered). At the current moment, this API only includes a *Migrate* command that provides initial support for the migration of a software component from one host to another. Finally, the model manipulation API allows atomic MOF-level modifications.

GeneSIS follows a declarative deployment approach. From the specification of the desired system state, which captures the needed system topology, the deployment engine computes how to reach this state. It is worth noting that the deployment engine may not always compute optimal plans. As a result, the interactions between the facade and the deployment engine via GeneSIS models.

Our engine is a typical implementation of the Models@Runtime pattern. When a target model fed the deployment engine, it is compared (see Diff in Figure 20) with the GeneSIS model representing the running system. Finally, the adaptation engine enacts the adaptation (*i.e.*, the deployment) by modifying only the parts of the system necessary to account for the difference and the target GeneSIS model becomes the current GeneSIS model.

A deployment process typically consists in the following steps:

1. **Check infrastructure**: This step consists in checking if the hosts specified in the deployment model are reachable (*e.g.*, is the docker remote API accessible at the address specified in the deployment model). For cloud resources, the objective is to check if the API offered by the cloud provider can be accessed.
2. **Provision and instantiate resource**: In the case of cloud solutions, this step consists in provisioning the cloud resources based on few constraints (*e.g.*, min CPU, min Disk, min RAM) and running the proper execution environment (*i.e.*, virtual machine image) as specified in the deployment model. We use CloudMF [103] for this. For container technologies, this step

[18] http://www.websocket.org/

consists in pulling the image of the container and running it with the set up specified in the deployment model (*e.g.*, access to file system, specifying open ports).

3. **Installation and configuration:** This step consists in running scripts and commands to configure and install extra software on the host. This includes ensuring that the software components that form the deployment topology can communicate with each other.
4. **Start:** This step consists in starting the deployed software.

It is not always possible for the GeneSIS deployment engine to directly deploy software on all hosts. Indeed, tiny devices do not always have access to Internet or even the necessary facilities for remote access (in such case access to Internet is typically granted via a gateway) and for specific reasons (*e.g.*, security) the deployment of software components can only be performed via a physical connection (*e.g.*, serial port). In such case, the actual action of deploying the software on the device has to be delegated to the gateway physically connected to the device. Within our example, this is the case of the *Arduino* device whose software code can only be updated via the *RaspberryPi* gateway. The GeneSIS deployment agent aims at addressing this issue.

In future work the objective is to continue developing GeneSIS following the conceptual design introduced in this deliverable. In addition, we will also consider working on aspects related to the development of a SIS. In particular, the development and operation of applications running on IoT devices such as Arduino boards is typically challenging as it is not always possible to access to the logs or to the systems output. **As future work we plan to extend ThingML and the deployment of ThingML programs via GeneSIS with the necessary mechanisms to enable the remote debugging of ThingML programs as well as the run-time monitoring of its execution flow.** One of the requirements should be to minimize the impact of such monitoring facilities on the performances of the host device, possibly delegating part of the work to more powerful resources.

GeneSIS will provide the link between the adaptation engines and the running system. In addition, some of the other tools developed within WP3 will interact with each other with the aim to provide extended capabilities.

5.2 Behavioural Drift Analysis as input for Online Learning

Reinforcement Learning (RL) techniques are used in the Online-Learning tool. RL field of temporal difference learning works well in known and rather small state spaces: (1) Through “sampling” (experimental behaviour) the agent gathers information about immediate rewards for performing certain actions in certain states; (2) Through bootstrapping the expected future reward can be approximated (based on the values of future states; on- and off-policy needs to be differentiated here).

At some point, the values of states do not change anymore. Convergence is guaranteed for a suitable selection of learning parameters (*e.g.*, in Reinforcement Learning, the e of the e-greedy policy is continuously decreased), so that less experience is explored, and focus lies on exploitation of knowledge (optimal behaviour). So, if the environment is stationary, this means that the learning process converges towards an optimal policy that needs to be exploited.

The main problem is that environment in the setting of IoT systems typically is non-stationarity. The environment in the SIS setting may continuously change over time. Reasons for such changes may include sensors becoming (un-)available during system operation to obtain environment information, the availability of other IoT systems to interact and cooperate with, as well as the amount and quality of data that may be obtained. So, online learning needs to become aware of such changes (drifts) in the environment in order to, for instance, adjust the learning parameters so that more exploration takes place in order to be able to capture the environment changes, such that rewards converge towards the new environmental behaviour. The problem is how to detect such changes in the environment. The behavioural drift analysis tool could be explored as a source to trigger a re-parametrization of the Online-Learning tool. A behavioural drift of the system may indicate that the environment has changed and thereby that this changed environment has to be explored by Online Learning.

5.3 Root-Cause Analysis as input for Online Learning

Root Cause Analysis (RCA) provides evidence about system nodes deviating from expected quality, such as performance. RCA thereby allows pin-pointing the system components or system functions that are the actual culprits for a deviation of system quality.

The information from RCA may serve as additional information to better guide the Online Learning process. As listed above, one of the requirements for the Online Learning tool is to achieve fast convergence. Knowing which of the functions are the culprits in quality problems may help to speed up learning. As an extreme measure, one may, for instance, exclude these culprit functions from the set of actions explored by online learning, thereby avoiding known quality problems. Alternatively, one may prioritize the actions based on the severity of the root cause, thereby providing a more fine-grained input to Online Learning.

5.4 Interrelationships between Root-Cause Analysis and Behavioural Drift Analysis

Behavioural Drift Analysis (BDA) can be used in conjunction with Root Cause Analysis (RCA) in a couple of ways. The first one is as a provider of evidences (an evidence in this case is the terminology used to describe anomalous events or conditions in a monitored component). Since BDA output is the degree of satisfaction of the observed behaviour with respect to the one expected by the model, deviations are potential indicators of anomalous behaviour that can be used by the RCA to enrich its analysis.

The other use is related to explainability and transparency. When evidences come from very simple anomaly detection mechanisms, it is trivial to provide information to the user on why a specific evidence was triggered. For instance, in the most basic anomaly detectors, which are threshold-based, it is easy for the user to understand that the evidence is there because the monitored signal went over/under a given threshold. However, with much more sophisticated techniques like BDA, it is desirable that the user could explore the rationale behind the evidences provided. For that reason, when a user is interested in having an explanation of the evidence while performing RCA, BDA could provide a comparison of the observed behaviour with respect to the expected one that would explain the degree of satisfaction output by the BDA. The BDA could also provide models corresponding to the drifted behaviour which could help developer by comparing the new model to the previous one.

In future work the objective is to strengthen the collaboration and explore other possibilities of exploiting BDA and RCA together.

6 Conclusion and next steps

WP3 aims at developing enablers for the operational part of the DevOps process. WP3 will provide enablers that provide IoT systems with capabilities to (i) monitor their status, (ii) indicate whether they behave as expected or not, (iii) identify the origin of the problem, and (iv) automatically perform typical operation activities (including self-adaptation of the systems). The main focus of the document is the studies performed on the state-of-the-art on the following topics related to the aforementioned capabilities: online learning, behavioural drift analysis, root-cause analysis and the support for self-adaptation of SIS.

For each study, a rigorous analysis of the state-of-the-art is conducted to define the requirements. WP3 will provide the following enablers: online learning, behavioural drift analysis and root-cause analysis. A conceptual design is proposed based on the identified requirements.

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

In the following annexes, further information about the systematic mapping study and a systematic literature review about Orchestration and Deployment are shown. Details about these specific topics are in D2.1.

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 7 gives the full list of the primary studies of the SMS.

Table 7. 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.

Table 8 shows the list of primary studies of the SLR.

Table 8: The primary deployment and orchestration studies of the SLR.

#	Year	Study	Title*	PV
S1	2017	FogTorch [90]	QoS-Aware Deployment of IoT Applications Through the Fog	J
S2	2017	ARCADIA [91]	A Middleware for Mobile Edge Computing	J
S3	2017	Chen et al. [92]	A Dynamic Module Deployment Framework for M2M Platforms	C
S10	2017	SoPIoT [104]	A Novel Service-Oriented Platform for the Internet of Things	C
S11	2017	Calvin [93]	Calvin Constrained: A Framework for IoT Applications in Heterogeneous Environments	C
S14	2017	Niflheim [105]	Niflheim: An end-to-end middleware for applications on a multi-tier IoT infrastructure	C
S15	2017	Verba et al. [106]	Platform-as-a-service gateway for the Fog of Things	J
S16	2017	Foggy [94]	Foggy- A Framework for Continuous Automated IoT Application Deployment in Fog Computing	C
S17	2017	TOSCA-BMWi [95]	A TOSCA-based Programming Model for Interacting Components of Automatically Deployed Cloud and IoT Applications	C
S12	2016	Cloud4IoT [96]	Cloud4IoT: A Heterogeneous, Distributed and Autonomic Cloud Platform for the IoT	C
S7	2015	D-NR [97]	Developing IoT Applications in the Fog: a Distributed Dataflow Approach	C
S9	2015	WComp [98]	A Generic Service Oriented Software Platform to Design Ambient Intelligent Systems	C
S13	2015	xWoT [107]	A component-based approach for the Web of Things	W
S5	2014	BeC3 [100]	BeC3: Behaviour Crowd Centric Composition for IoT applications	J
S8	2014	glue.things [108]	glue.things - a Mashup Platform for wiring the Internet of Things with the Internet of Services	W
S6	2013	SAaaS [109]	Application deployment for IoT: An infrastructure approach	C
S4	2011	D-LITE [99]	D-LITE: Distributed logic for internet of things services	C

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

References

1. Sutton, R.S. and A.G. Barto, *Reinforcement learning: An introduction*. 1998: MIT press.
2. Porter, B. and R. Rodrigues Filho. *Losing Control: The Case for Emergent Software Systems Using Autonomous Assembly, Perception, and Learning*. in *Self-Adaptive and Self-Organizing Systems (SASO), 2016 IEEE 10th International Conference on*. 2016. IEEE.
3. Tesauro, G., et al., *On the use of hybrid reinforcement learning for autonomic resource allocation.* Cluster Computing, 2007. **10**(3): p. 287-299.
4. Amoui, M., et al. *Adaptive action selection in autonomic software using reinforcement learning.* in *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*. 2008. IEEE.
5. Kim, D. and S. Park. *Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software*. in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*. 2009. IEEE.
6. Barrett, E., E. Howley, and J. Duggan, *Applying reinforcement learning towards automating resource allocation and application scalability in the cloud.* Concurrency and Computation: Practice and Experience, 2013. **25**(12): p. 1656-1674.
7. Jamshidi, P., et al. *Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures*. in *Quality of Software Architectures (QoSA), 2016 12th International ACM SIGSOFT Conference on*. 2016. IEEE.
8. Jamshidi, P., et al. *Transfer learning for improving model predictions in highly configurable software*. in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2017. IEEE Press.
9. Porter, B., *Defining emergent software using continuous self-assembly, perception, and learning.* ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2017. **12**(3): p. 16.
10. Sharifloo, A.M., et al. *Learning and evolution in dynamic software product lines*. in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2016 IEEE/ACM 11th International Symposium on*. 2016. IEEE.
11. Richter, M.M. and R.O. Weber, *Case-based reasoning*. 2016: Springer.
12. Qian, W., et al., *Rationalism with a dose of empiricism: combining goal reasoning and case-based reasoning for self-adaptive software systems.* Requirements Engineering, 2015. **20**(3): p. 233-252.
13. Zhao, T., et al. *A Reinforcement Learning-based Framework for the Generation and Evolution of Adaptation Rules*. in *Autonomic Computing (ICAC), 2017 IEEE International Conference on*. 2017. IEEE.
14. Ramirez, A.J., et al., *Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems.* Cluster Computing, 2011. **14**(3): p. 229-244.
15. Moustafa, A. and M. Zhang. *Learning efficient compositions for QoS-aware service provisioning*. in *2014 IEEE International Conference on Web Services (ICWS)*. 2014. IEEE.
16. Wang, H., et al., *Integrating reinforcement learning with multi-agent techniques for adaptive service composition.* ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2017. **12**(2): p. 8.
17. van Otterlo, M. and M. Wiering, *Reinforcement learning and markov decision processes*, in *Reinforcement Learning*. 2012, Springer. p. 3-42.
18. Laprie, J.-C., *Dependability: Basic concepts and terminology*, in *Dependability: Basic Concepts and Terminology*. 1992, Springer. p. 3-245.
19. Gao, Z., C. Cecati, and S.X. Ding, *A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches.* IEEE Transactions on Industrial Electronics, 2015. **62**(6): p. 3757-3767.
20. Kavulya, S.P., et al., *Failure diagnosis of complex systems*, in *Resilience assessment and evaluation of computing systems*. 2012, Springer. p. 239-261.

21. Bertolino, A., et al. *Dependability and performance assessment of dynamic connected systems*. in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. 2011. Springer.
22. Delgado, N., A.Q. Gates, and S. Roach, *A taxonomy and catalog of runtime software-fault monitoring tools.* IEEE Transactions on software Engineering, 2004. **30**(12): p. 859-872.
23. Chandola, V., A. Banerjee, and V. Kumar, *Anomaly detection: A survey*. ACM computing surveys (CSUR), 2009. **41**(3): p. 15.
24. Pimentel, M.A., et al., *A review of novelty detection.* Signal Processing, 2014. **99**: p. 215-249.
25. Gupta, M., et al., *Outlier detection for temporal data.* Synthesis Lectures on Data Mining and Knowledge Discovery, 2014. **5**(1): p. 1-129.
26. Malhotra, P., et al. *Long short term memory networks for anomaly detection in time series*. in *Proceedings*. 2015. Presses universitaires de Louvain.
27. Goh, J., et al. *Anomaly detection in cyber physical systems using recurrent neural networks*. in *High Assurance Systems Engineering (HASE), 2017 IEEE 18th International Symposium on*. 2017. IEEE.
28. Vodenčarević, A., et al. *Using behavior models for anomaly detection in hybrid systems*. in *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on*. 2011. IEEE.
29. Lipton, Z.C., *The mythos of model interpretability*. arXiv preprint arXiv:1606.03490, 2016.
30. Mascaro, S., A. Nicholson, and K. Korb. *Anomaly detection in vessel tracks using Bayesian networks*. in *Proceedings of the 8th Bayesian Modeling Applications Workshop*. 2011. Citeseer.
31. Loy, C.C., T. Xiang, and S. Gong, *Detecting and discriminating behavioural anomalies.* Pattern Recognition, 2011. **44**(1): p. 117-132.
32. Lin, Q., et al. *TABOR: A Graphical Model-based Approach for Anomaly Detection in Industrial Control Systems*. in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 2018. ACM.
33. Fraser, A.M., *Hidden Markov models and dynamical systems*. Vol. 107. 2008: Siam.
34. Al-ani, T., *Hidden Markov models in dynamic system modelling and diagnosis*, in *Hidden Markov models, theory and applications*. 2011, InTech.
35. Park, D., et al. *A multimodal execution monitor with anomaly classification for robot-assisted feeding*. in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. 2017. IEEE.
36. Li, J., W. Pedrycz, and I. Jamal, *Multivariate time series anomaly detection: A framework of Hidden Markov Models.* Applied Soft Computing, 2017. **60**: p. 229-240.
37. Stefanidis, K. and A.G. Voyiatzis. *An HMM-based anomaly detection approach for SCADA systems*. in *IFIP International Conference on Information Security Theory and Practice*. 2016. Springer.
38. Lefebvre, D., *Detection of Temporal Anomalies for Partially Observed Timed PNs.* Mathematical Problems in Engineering, 2017. **2017**.
39. Rocher, G., J.-Y. Tigli, and S. Lavirotte, *Probabilistic Models Toward Controlling Smart-* Environments.* IEEE Access, 2017. **5**: p. 12338-12352.
40. Rocher, G., et al. *A Possibilistic I/O Hidden Semi-Markov Model For Assessing Cyber-Physical Systems Effectiveness*. in *International Conference on Fuzzy Systems*. 2018.
41. Webb, G.I., et al., *Characterizing concept drift.* Data Mining and Knowledge Discovery, 2016. **30**(4): p. 964-994.
42. Itti, L. and P. Baldi, *Bayesian surprise attracts human attention.* Vision research, 2009. **49**(10): p. 1295-1306.
43. Storck, J., S. Hochreiter, and J. Schmidhuber. *Reinforcement driven information acquisition in non-deterministic environments*. in *Proceedings of the international conference on artificial neural networks, Paris*. 1995. Citeseer.
44. Bause, F. and P. Kritzinger, *Stochastic Petri Nets.* Verlag Vieweg, Wiesbaden, 1996. **26**.
45. Benveniste, A., E. Fabre, and S. Haar, *Markov nets: probabilistic models for distributed and concurrent systems.* IEEE Transactions on Automatic Control, 2003. **48**(11): p. 1936-1950.
46. Cardoso, J., R. Valette, and D. Dubois, *Possibilistic petri nets.* IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 1999. **29**(5): p. 573-582.

47. Morrison, J.P. *Flow-based programming*. in *Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems*. 1994.
48. Solé, M., et al., *Survey on models and techniques for root-cause analysis.* arXiv preprint arXiv:1701.08546, 2017.
49. łgorzata Steinder, M. and A.S. Sethi, *A survey of fault localization techniques in computer networks.* Science of computer programming, 2004. **53**(2): p. 165-194.
50. Agarwal, P. and A.P. Agrawal, *Fault-localization techniques for software systems: A literature review.* ACM SIGSOFT Software Engineering Notes, 2014. **39**(5): p. 1-8.
51. Lazarova-Molnar, S., H.R. Shaker, and N. Mohamed. *Fault detection and diagnosis for smart buildings: State of the art, trends and challenges*. in *Big Data and Smart City (ICBDSC), 2016 3rd MEC International Conference on*. 2016. IEEE.
52. Katipamula, S. and M.R. Brambley, *Methods for fault detection, diagnostics, and prognostics for building systems—a review, part I.* Hvac&R Research, 2005. **11**(1): p. 3-25.
53. Feng, Z., M. Liang, and F. Chu, *Recent advances in time–frequency analysis methods for machinery fault diagnosis: A review with application examples.* Mechanical Systems and Signal Processing, 2013. **38**(1): p. 165-205.
54. Qin, L., X. He, and D. Zhou, *A survey of fault diagnosis for swarm systems.* Systems Science & Control Engineering: An Open Access Journal, 2014. **2**(1): p. 13-23.
55. Isermann, R., *Model-based fault-detection and diagnosis–status and applications.* Annual Reviews in control, 2005. **29**(1): p. 71-85.
56. Hwang, I., et al., *A survey of fault detection, isolation, and reconfiguration methods.* IEEE transactions on control systems technology, 2010. **18**(3): p. 636-653.
57. Lanigan, P.E., et al., *Diagnosis in automotive systems: A survey*. Last accessed Sept, 2011. **10**: p. 2011.
58. Mohammadpour, J., M. Franchek, and K. Grigoriadis, *A survey on diagnostic methods for automotive engines.* International Journal of Engine Research, 2012. **13**(1): p. 41-64.
59. Patton, R., *Fault detection and diagnosis in aerospace systems using analytical redundancy.* Computing & Control Engineering Journal, 1991. **2**(3): p. 127-136.
60. Bruce, G., B. Buchanan, and E. Shortliffe, *Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project*. 1984, Reading, Mass: Addison-Wesley.
61. Mahandeka, D.S. and D.M. Rosyid, *Fault Tree Analysis for Investigation on the Causes of Project Problems.* Procedia Earth and Planetary Science, 2015. **14**: p. 213-219.
62. Alaeddini, A. and I. Dogan, *Using Bayesian networks for root cause analysis in statistical process control.* Expert Systems with Applications, 2011. **38**(9): p. 11230-11243.
63. Schoenfisch, J., et al., *Using abduction in markov logic networks for root cause analysis.* arXiv preprint arXiv:1511.05719, 2015.
64. Korb, K.B. and A.E. Nicholson, *Bayesian artificial intelligence*. 2010: CRC press.
65. Stewart, R. and S. Ermon. *Label-free supervision of neural networks with physics and domain knowledge*. in *AAAI*. 2017.
66. Velickovic, P., et al., *Graph attention networks.* arXiv preprint arXiv:1710.10903, 2017.
67. Yan, H., et al., *G-rca: a generic root cause analysis platform for service quality management in large ip networks.* IEEE/ACM Transactions on Networking (TON), 2012. **20**(6): p. 1734-1747.
68. Yemini, S.A., et al., *High speed and robust event correlation.* IEEE communications Magazine, 1996. **34**(5): p. 82-90.
69. Monacelli, L. and G. Reali, *Evolution of the codebook technique for automatic fault localization.* IEEE Communications Letters, 2011. **15**(4): p. 464-466.
70. Zheng, A.X., J. Lloyd, and E. Brewer, *Failure Diagnosis Using Decision Trees*, in *Proceedings of the First International Conference on Autonomic Computing*. 2004, IEEE Computer Society. p. 36-43.
71. Alves, J., et al., *Brief survey on computational solutions for Bayesian inference.* 2015.
72. Zheng, L., O. Mengshoel, and J. Chong, *Belief propagation by message passing in junction trees: Computing each message faster using gpu parallelization.* arXiv preprint arXiv:1202.3777, 2012.

73. Lin, M., I. Lebedev, and J. Wawrzynek. *High-throughput bayesian computing machine with reconfigurable hardware*. in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. 2010. ACM.
74. Kompella, R.R., et al. *IP fault localization via risk modeling*. in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. 2005. USENIX Association.
75. Zasadzinski, M., V. Muntés-Mulero, and M.S. Simo. *Actor based root cause analysis in a distributed environment*. in *Software Engineering for Smart Cyber-Physical Systems (SEsCPS), 2017 IEEE/ACM 3rd International Workshop on*. 2017. IEEE.
76. Zasadziński, M., et al., *Fast root cause analysis on distributed systems by composing precompiled bayesian networks.* Proc. World Congr. on Engineering and Computer Science, 2016. **1**: p. 464-469.
77. Domingos, P. and G. Hulten. *Mining high-speed data streams*. in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2000. ACM.
78. Hulten, G., L. Spencer, and P. Domingos. *Mining time-changing data streams*. in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001. ACM.
79. Bifet, A. and R. Gavaldà. *Adaptive learning from evolving data streams*. in *International Symposium on Intelligent Data Analysis*. 2009. Springer.
80. Petcu, D. *Multi-Cloud: expectations and current approaches*. in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. 2013. ACM.
81. Ferry, N., et al., *CloudMF: Model-Driven Management of Multi-Cloud Applications.* ACM Transactions on Internet Technology (TOIT), 2018. **18**(2): p. 16.
82. 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.
83. 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.
84. Carrega, A., et al., *A middleware for mobile edge computing.* IEEE Cloud Computing, 2017. **4**(4): p. 26-37.
85. 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.
86. 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.
87. 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.
88. 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.
89. 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.
90. 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.
91. 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.
92. 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.
93. Cherrier, S., et al., *BeC 3: behaviour crowd centric composition for IoT applications.* Mobile Networks and Applications, 2014. **19**(1): p. 18-32.

94. McKinley, P.K., et al., *A taxonomy of compositional adaptation.* Rapport Technique numéroMSU-CSE-04-17, 2004.
95. Ruscio, D.D., R.F. Paige, and A. Pierantonio, *Guest editorial to the special issue on Success Stories in Model Driven Engineering.* Sci. Comput. Program., 2014. **89**(PB): p. 69-70.
96. Blair, G., N. Bencomo, and R.B. France, *Models@ run. time.* Computer, 2009. **42**(10).
97. Ferry, N., et al. *CloudMF: applying MDE to tame the complexity of managing multi-cloud applications*. in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*. 2014. IEEE.
98. O'Leary, N. and D. Conway-Jones, *Node red-a visual tool for wiring the internet of things*. 2017.
99. 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.
100. 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.
101. 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.
102. Verba, N., et al., *Platform as a service gateway for the Fog of Things.* Advanced Engineering Informatics, 2017. **33**: p. 243-257.
103. 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.
104. 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.
105. Distefano, S., G. Merlino, and A. Puliafito. *Application deployment for IoT: An infrastructure approach*. in *Global Communications Conference (GLOBECOM), 2013 IEEE*. 2013. IEEE.