

Chapter 5

A DevOps Toolchain for Managing Actuation Conflicts in Smart IoT Systems

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

5.1 Introduction

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

5.1.1 SIS Actuation Challenges

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

5.1.2 DevOps Still Lacks the Tools to Meet These Challenges

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

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

5.1.3 An End-to-end DevOps Toolchain

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

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

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

5.2 Overview of the SIS Actuation Conflict Management Toolset

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

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

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

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

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

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

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

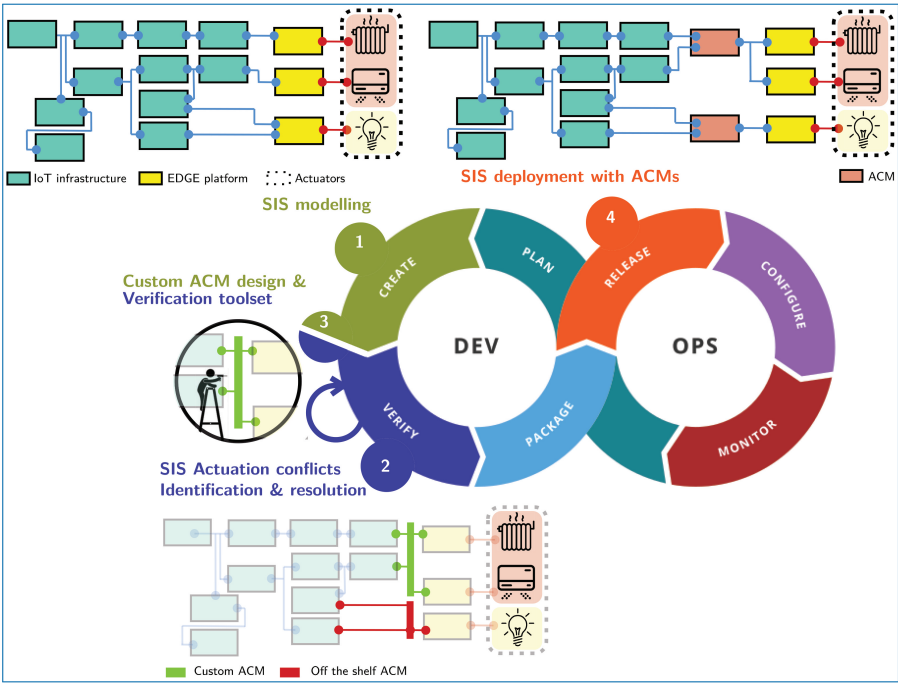


Figure 5.1. Usage of Actuation Conflict Management (ACM) tool-set throughout the DevOps life-cycle.

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

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

5.2.1 Beyond the State of the Art

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

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

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

5.2.2 Actuation Conflict Management Workflow

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

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

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

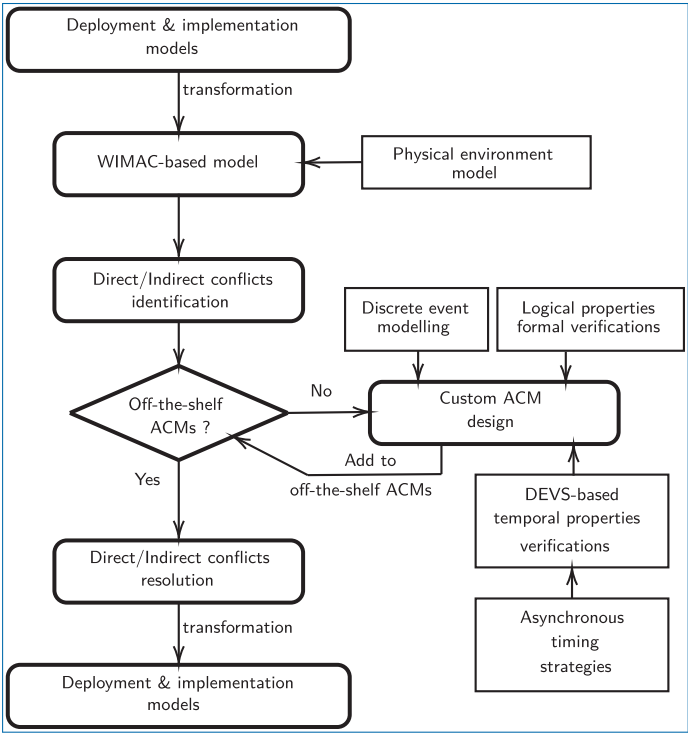


Figure 5.2. Actuation conflicts identification and resolution workflow.

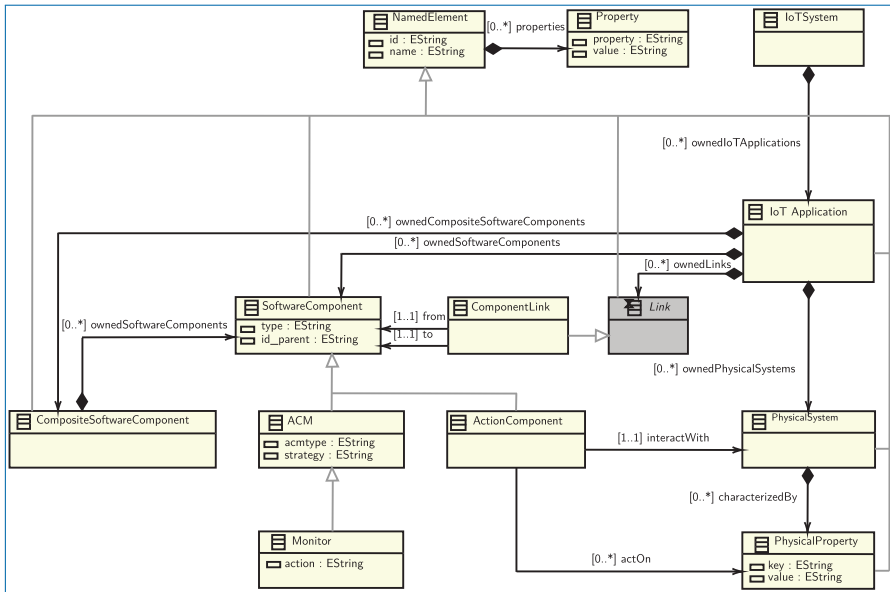


Figure 5.3. WIMAC meta-model.

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

1. **SoftwareComponents** are black-box components. They conceptualize single applications or composite applications described through implementation models (e.g., Node-RED¹ flow). The actuation conflict management solution proposed in the sequel is based solely on the structural links that exist between software components for inferring their interrelationships, identifying and resolving potential conflicts.
2. **ActionComponents** are SoftwareComponents controlling actuators,
3. **PhysicalSystems** are spatially delimited physical entities whose properties can be changed by ActionComponents (e.g. the temperature in the kitchen).

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

1. <https://nodered.org>

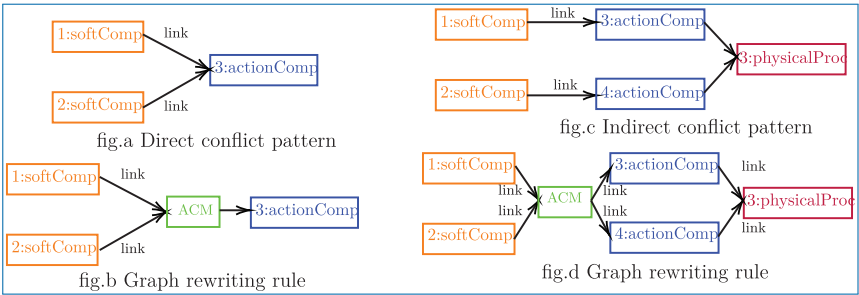


Figure 5.4. Example of AGG identification and rewriting rules for direct (left) and indirect (right) actuation conflicts.

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

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

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

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

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

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

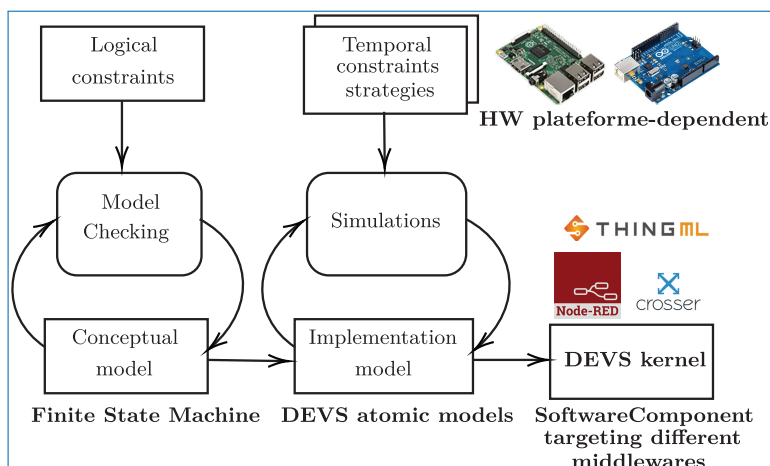


Figure 5.5. Custom ACM design workflow.

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

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

5.3 Overview of the SIS Behaviour Monitoring and Analysis Toolset

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

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

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

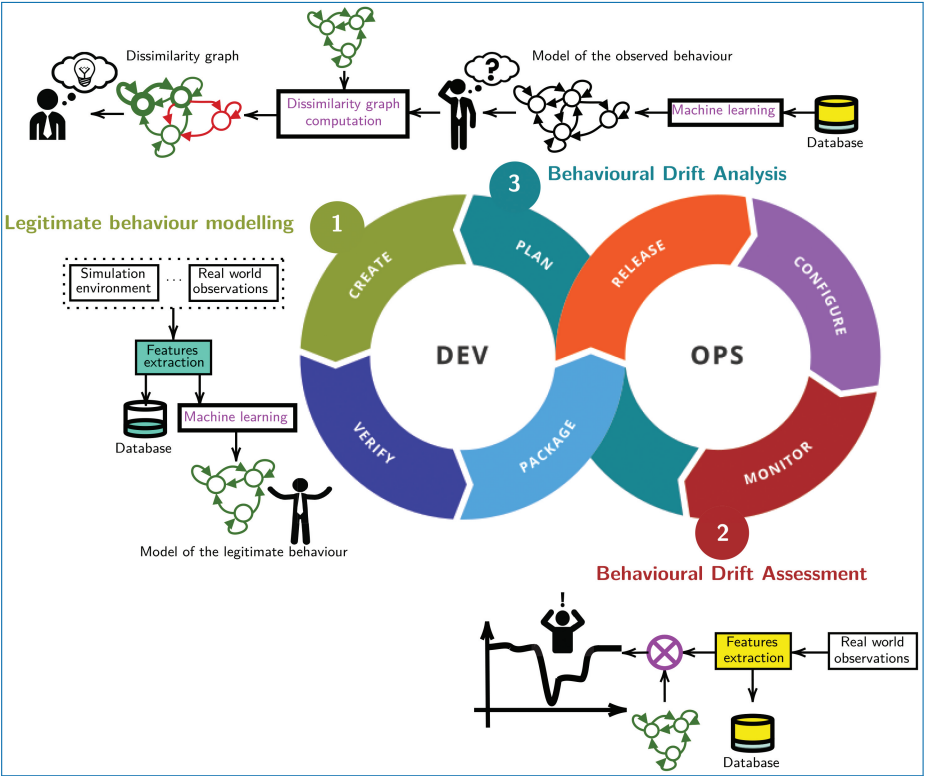


Figure 5.6. Usage of the Behavioural Drift Analysis (BDA) tool-set throughout the DevOps life-cycle.

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

5.3.1 Beyond the State of the Art

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

5.3.2 Behavioural Drift Assessment Tool

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

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

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

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

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

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

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

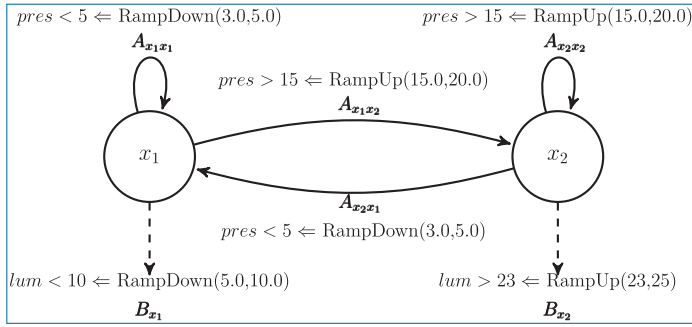


Figure 5.7. Possibilistic IOHMM model describing the legitimate effects that a luminosity control system must produce.

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

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

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

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

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

3. **Termination**

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

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

5.3.3 Behavioural Drift Analysis Tool

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

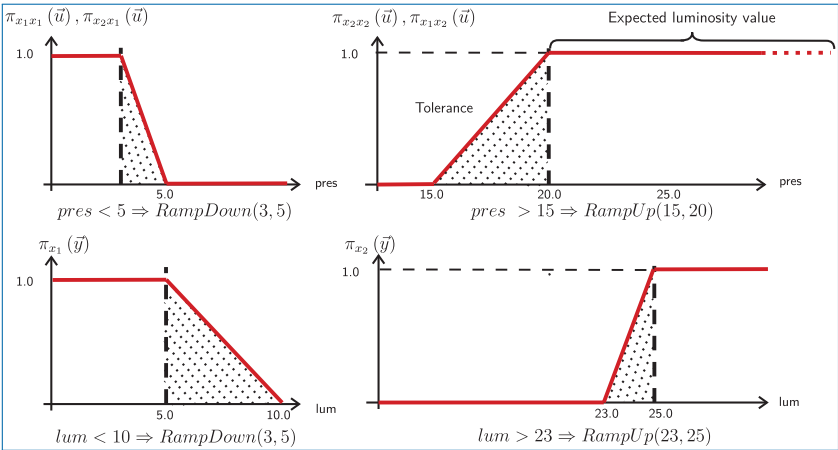


Figure 5.8. Distributions of possibility defining tolerances on expectations of the model depicted in Fig. 5.7.

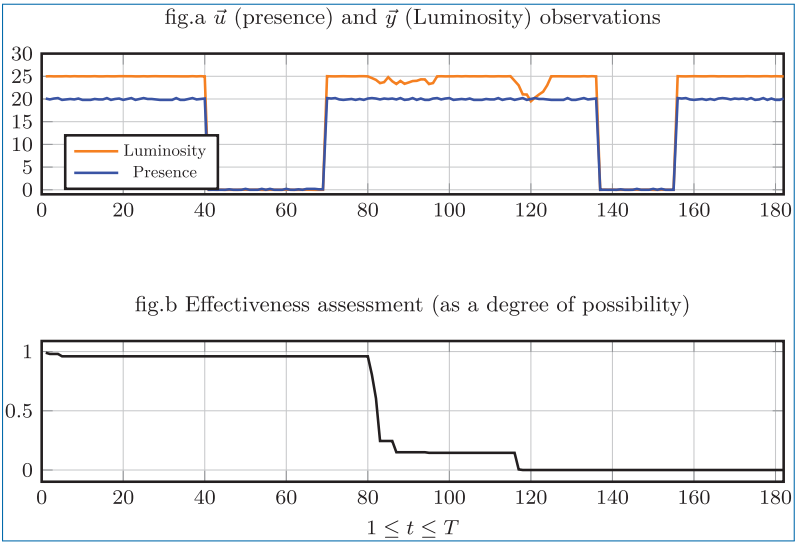


Figure 5.9. Effectiveness assessment results obtained on the model depicted in Fig. 5.7 from synthetic observations applied to the forward algorithm.

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

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

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

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

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

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

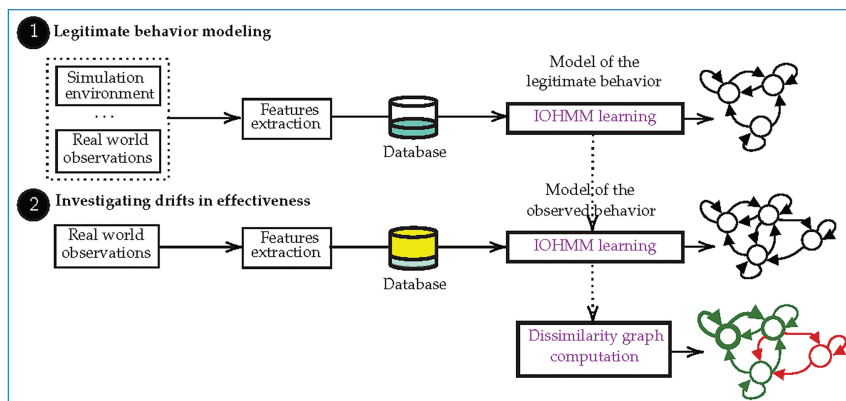


Figure 5.10. Steps to implement the proposed approach.

- ② This model is complemented with real-world observations, leading to the learning of a model of the observed behaviour. Then, on the basis of these models, an algorithm is developed to build a directed dissimilarity graph that makes clear the differences between both models, thereby helping designers direct their research and identify the possible causes of drifts in effectiveness.

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

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

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

5.4 Smart Home Use-Case and Illustration

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

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

5.4.1 Smart Home use Case Description

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

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

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

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

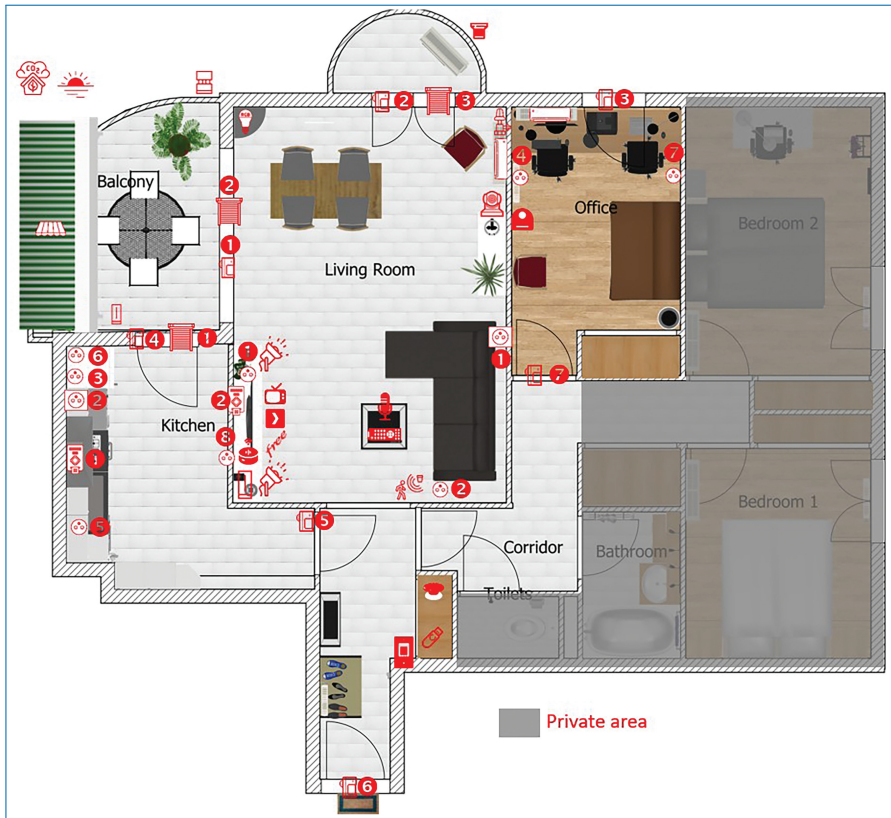


Figure 5.11. Smart Home use case setup.

5.4.2 Software Development (Devs, Cycle 1)

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

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

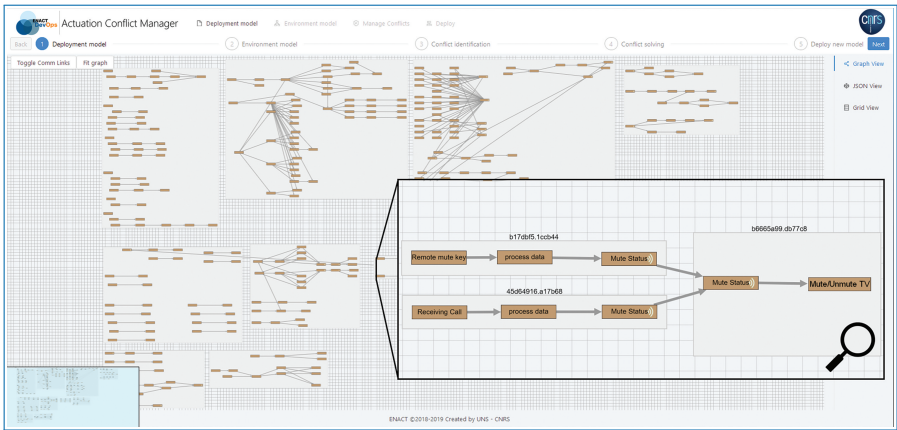


Figure 5.12. Excerpt of *App_Phone_TV* and *App_RC_TV* structural inter-relationships, as described in the WIMAC model, without environment model description.

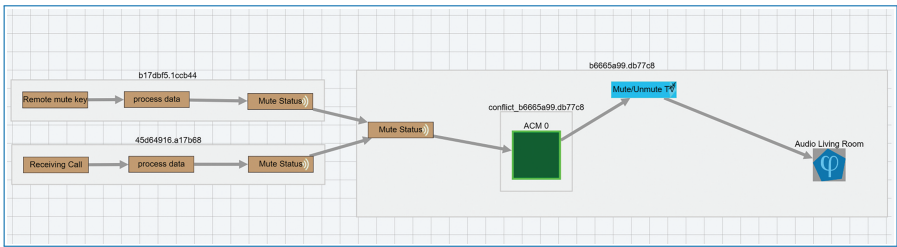


Figure 5.13. Excerpt of *App_Phone_TV* and *App_RC_TV* structural inter-relationships, as described in the WIMAC model with environment model description and the direct ACM instantiated.

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

5.4.3 System Operations (Ops, Cycle 1)

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

The model of the legitimate behaviour is built upon a set of sound features computed from a microphone signal. Output observations (expected effects to be produced by the SIS) are then characterized by a Mel-Frequency Cepstral Coefficient³

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

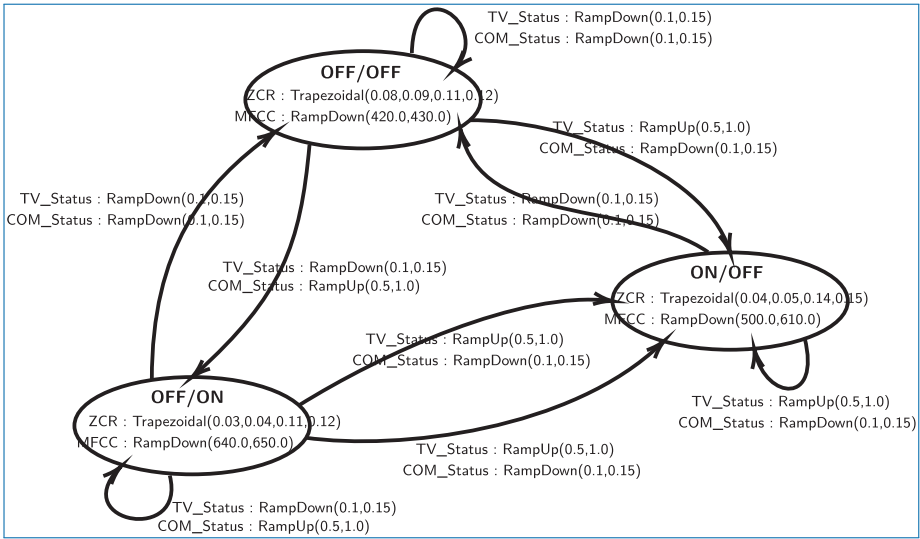


Figure 5.14. Possibilistic model of the legitimate behaviour. States are defined by the operating status of the TV and the communication status (i.e., OFF/ON means TV:ON, COM:OFF). The configuration ON/ON is not legitimate, the direct ACM instantiated is supposed to mute the TV while a communication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

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

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

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

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

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

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

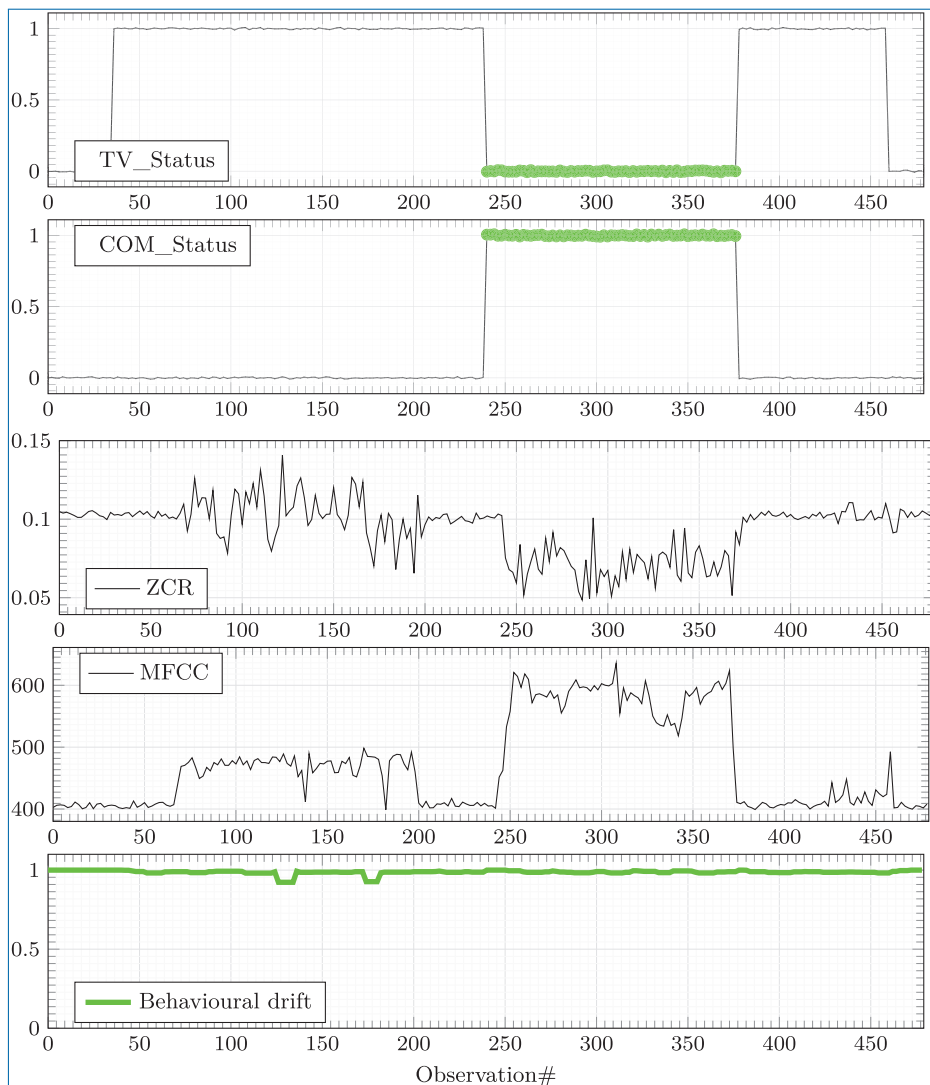


Figure 5.15. Observations from the field corresponding to the legitimate behaviour. The ACM instantiated during the development phase fulfils its role (green part of the graph) and no drift is reported here, the SIS behaves as expected.

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

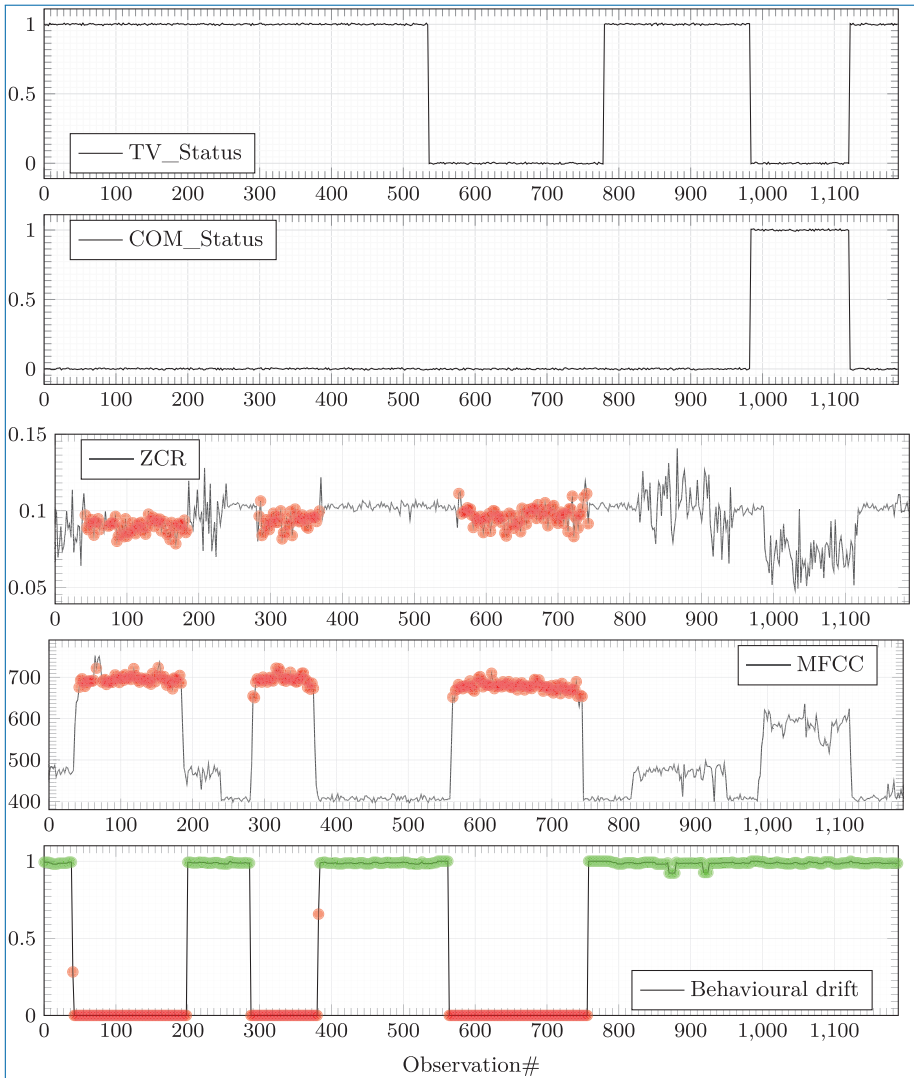


Figure 5.16. Observations from the field coloured in red denote the presence of the smart vacuum cleaner not anticipated/foreseen in the model of the legitimate behaviour (leading behavioural drifts to occur).

5.4.4 Software Development (Devs, Cycle 2)

The behavioural drift reported at run-time suggests that the model of the physical environment is incomplete. Indeed, the indirect conflict on the sound physical property should have been identified during the first phase of development. A new development cycle is therefore necessary to correct the model of the physical environment. The updated model is depicted in Fig. 5.18; an indirect

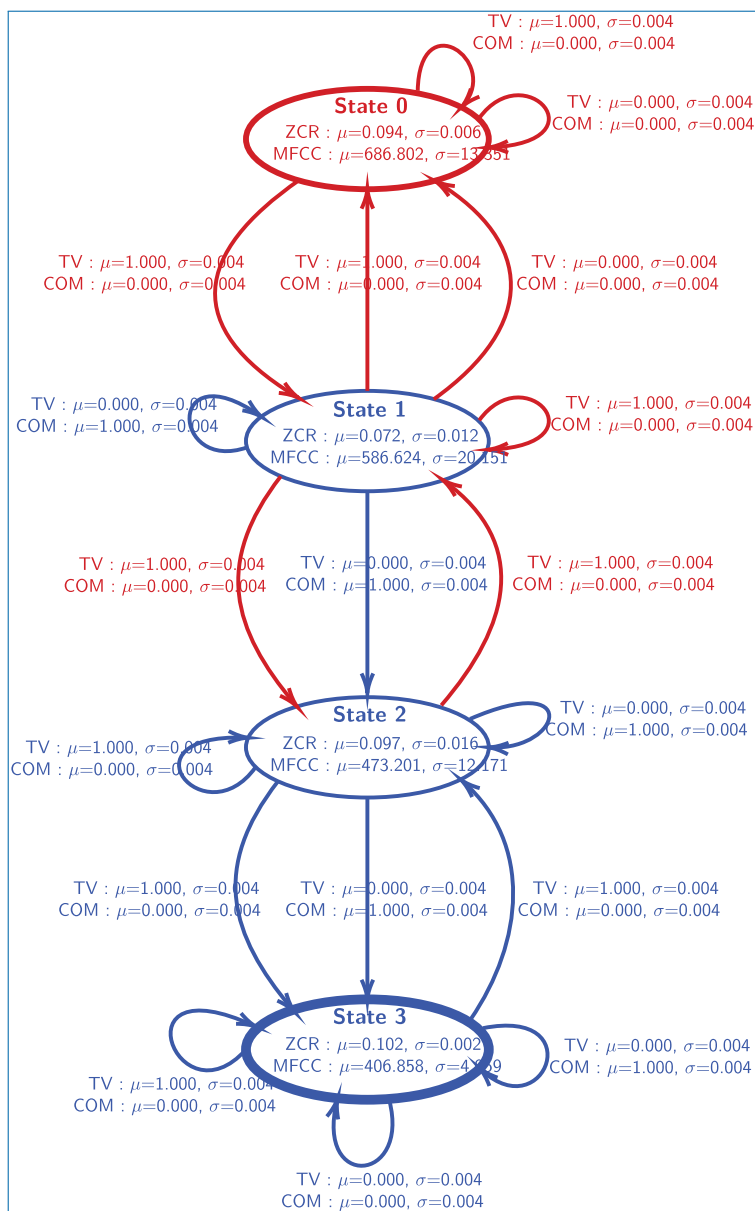


Figure 5.17. Dissimilarity graph. Nodes and edges coloured in blue correspond to the legitimate behaviour, those in red correspond to unexpected behaviour resulting from the appearance of the smart vacuum cleaner in the living-room.

actuation conflict is now detected between the *App_RC_TV*, *App_Phone_TV* and the *App_Vacuum_Cleaner* applications and a dummy *ACM* has been instantiated.

The management of this indirect actuation conflict implies semantic concerns that cannot be managed through generic off-the-shelf *ACMs*. A custom *ACM*

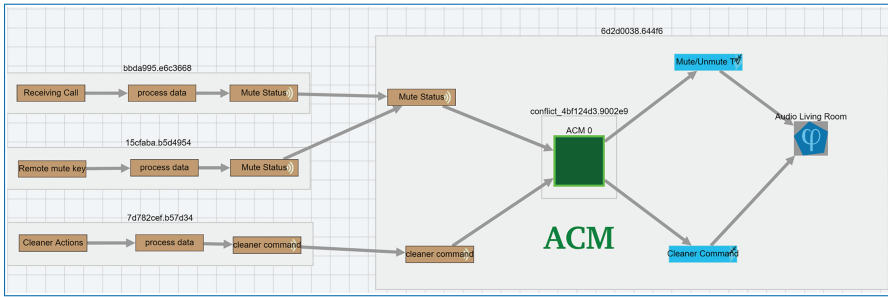


Figure 5.18. Excerpt of indirect conflict detection and management.

has to be developed. This task relies on “ECA” rules that define the ACM logic, further transformed into an FSM. More specifically, designers have to describe a control strategy that prevent the TV and the smart vacuum cleaner robot to produce sound simultaneously. The custom ACM thus receives commands from *App_RC_TV*), *App_Phone_TV* and *App_Vaccum_Cleaner*. On the basis of these input, the ACM control logic defines a strategy resulting in sending commands to the TV and the vacuum cleaner such that it is stopped while a communication is in progress or while inhabitants watch TV. The ACM control strategy defines some prioritization whose semantic has to be defined by designers.

Along with the logical strategy of the custom ACM, some logical and temporal properties to be formally verified have to be specified. To this end, ECA rules are extended into ECA+ rules (more details about this language are given in [7]) to provide the model with the properties to be verified by state-of-the-art MDE verification tools.

Logical properties are verified through NuSMV [8], a state of the art model checker. Model checkers are meant to ensure that logical properties are always verified, regardless of the state sequence being executed. Two main types of logical properties are formally verified: *safety* (i.e., something bad will never occur) and *Liveness* (i.e., something good will eventually happen). In the context of ACM design, safety properties ensure that conflicting states will never occur (e.g., the vacuum cleaner will never operate while a communication is in progress).

Temporal properties are verified through DEVS formalism. FSMs are defined by two functions: (i) the state-transition function computes the new state given the previous state $x_{(k-1)}$ at time $k - 1$ and the current input $\vec{u}_{(k)}$ at time k , (ii) the output function computes the outputs $\vec{y}_{(k)}$ that solely depend on the state $x_{(k)}$ at time k . DEVS formalism allows an FSM to be encapsulated into DEVS atomic model coupled with a synchronizer managing asynchronous timings on the inputs of the FSM. The ECA+ language provides a syntax to specify asynchronous timings management strategies for the synchronizer. Temporal properties can then be

verified for each strategy within a DEVS simulation environment. It is worth noting that asynchronous timing strategies are meant to reproduce the asynchronous timings that govern the different hardware platforms **ACMs** are supposed to be deployed to, at the edge of the **IoT** infrastructure.

Once **ACMs** temporal properties have been verified in DEVS simulated operational contexts, associated DEVS Atomic models (embedding the synchronizer and the logical behaviour) can be directly used for implementation. Indeed, DEVS Atomic models can be translated into high level programming languages (C, C++, C#, Node.js, etc.), embedding a lightweight execution engine (DEVS kernel). This makes **ACMs** completely portable on the lightweight hardware platforms available at the edge of the **IoT** infrastructure.

Here again, the WIMAC model modified with the custom **ACM** is sent to Genesis for the deployment on the platforms, thus completing the second development cycle.

5.4.5 System Operations (Ops, Cycle 2)

Finally, **SIS** is deployed and the physical effects it produces in the physical environment are observed. The model of the legitimate behaviour is modified to take into account the autonomous vacuum cleaner, as learned during the first cycle. No behavioural drift is reported anymore. However, as the physical environment is complex, there are many reasons why the behaviour of the **SIS** may drift again and trigger a new development cycle:

- the introduction of a new software component that drives an actuator not correctly defined in the model of the physical environment,
- the introduction of a new device producing physical effects in the environment (as the vacuum cleaner in the use-case),
- unexpected changes in the physical environment in which the **SIS** operate (e.g., a tree growing in front of a window is likely to have an impact on the luminosity of the room in the long term).

5.5 Conclusion and Future Works

This chapter has introduced two innovative toolsets designed to enrich the **DevOps** eco-system and meant to address an issue that has been poorly addressed, to date, in the development of trustworthy **SIS**: the management of applications that can interact with their physical environment through actuators.

The first toolset, called “Actuation Conflict Management” toolset (**ACM**), is to secure the design of these applications at Devs time. The objective is to identify

and resolve, through local structural transformations of the **SIS**, the actuation conflicts that might arise as a result of the actuation effects produced in the physical environment a priori known and limited to its model. While such precautions at Devs time are necessary, they are not sufficient to guarantee that the **SIS** will always operate as expected in the physical environment. There is a risk of behavioural drift which is observed and quantified at Ops time, thanks to the second toolset called “Behavioural Drift Assessment & Analysis” toolset (**BDA**). This toolset makes it possible to assess and analyse the differences between the expected and observed effects of the **SIS** in real-world environment.

Beyond the interest of these toolsets as part of the **DevOps** methodology, their complementarity represents a major contribution in the realm of trustworthy **SIS**. Indeed, it is during consecutive **DevOps** cycles that the contribution of the couple **BDA-ACM** can be fully appreciated. The **BDA** toolset provides **DevOps** team with information on the observed behavioural drifts, thus motivating new development cycles which then results in changes to the model of the physical environment and/or corrections of the applications carried out in and with the **ACM** toolset. This complementarity is highlighted in this chapter throughout a smart-home use-case which involves two consecutive **DevOps** cycles to converge towards a **SIS** with satisfactory behaviour.

It is within the framework of this complementarity that two possible lines of work are foreseen. The first axis aims to reinforce the added-value of the information obtained from the **BDA** so as to accelerate the **SIS/ACMs** re-design cycle. Indeed, while the **BDA** toolset allows to obtain a model of the effects observed in the field and the symptoms of their dissimilarities with the legitimate ones, it is neither informative on the root-causes underlying these differences nor it provides insights to infer the corrections to be made to the model of the physical environment. The second axis is about leveraging behavioural drift measures as rewards towards self-adaptive **ACMs** automatically containing/mitigating behavioural drifts at run-time.

References

- [1] A. Metzger (Ed.). *Software Continuum: Recommendations for ICT Work Programme 2018–2019*. NESSI White Paper, February, 2016.
- [2] Abdullah Al Farooq *et al.* “Iotc 2: A formal method approach for detecting conflicts in large scale IoT systems”. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE. 2019, pp. 442–447.
- [3] Robert C. Armstrong *et al.* “Survey of existing tools for formal verification”. In: *SANDIA REPORT SAND2014-20533* (2014).

- [4] Yoshua Bengio and Paolo Frasconi. “An input output hmm architecture”. In: *Advances in neural information processing systems*, pp. 427–434, 1995.
- [5] James Bruce. *Getting Started with OpenHAB Home Automation on Raspberry Pi*. 2015.
- [6] Laurent Capocchi *et al.* “DEVSimPy: A collaborative python software for modeling and simulation of DEVS systems”. In: *2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE. 2011, pp. 170–175.
- [7] Franck Chauvel *et al.* *Risk-driven Continuous delivery of Trustworthy Smart IoT Systems*. 2020. URL: <https://enact-project.eu/deliverables/D2.3.pdf>.
- [8] Alessandro Cimatti *et al.* “Nusmv 2: An opensource tool for symbolic model checking”. In: *International Conference on Computer Aided Verification*. Springer. 2002, pp. 359–364.
- [9] European Commission. *Horizon 2020 Work Programme 2016–2017 – Cross-cutting activities (Focus Areas), Call IoT-03-2017 : R&I on IoT integration and platforms*. Oct. 2015. URL: http://ec.europa.eu/newsroom/dae/document.cfm?%5C&doc%5C_id=11867%7D.
- [10] Muhammad Fahim and Alberto Sillitti. “Anomaly detection, analysis and prediction techniques in IoT environment: A systematic literature review”. In: *IEEE Access* 7 (2019), pp. 81664–81681.
- [11] G. Rocher, J.-Y. Tigli, S. Lavirotte and N. Le Thanh. *Effectiveness assessment of Cyber-Physical Systems*. 2020.
- [12] Thibaut Gonnin *et al.* “Actuation Conflict Management Enabler for DevOps in IoT”. In: *10th International Conference on the Internet of Things Companion, IoT '20 Companion*. Malmö, Sweden: Association for Computing Machinery, 2020. ISBN: 9781450388207. DOI: 10.1145/3423423.3423474. URL: <https://doi.org/10.1145/3423423.3423474>.
- [13] Edward R. Griffor *et al.* “Framework for cyber-physical systems: Volume 2, working group reports”. In: (2017).
- [14] Andrew Guillory *et al.* “Learning executable agent behaviors from observation”. In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. 2006, pp. 795–797.
- [15] Hani Hagrais. “Toward human-understandable, explainable AI”. In: *Computer* 51.9 (2018), pp. 28–36.
- [16] Andrew KS Jardine, Daming Lin, and Dragan Banjevic. “A review on machinery diagnostics and prognostics implementing condition-based maintenance”. In: *Mechanical systems and signal processing* 20.7. (2006), pp. 1483–1510.

- [17] Mengda Jia *et al.* “Adopting Internet of Things for the development of smart buildings: A review of enabling technologies and applications”. In: *Automation in Construction* 101 (2019) 111–126. ISSN: 0926-5805. DOI: <https://doi.org/10.1016/j.autcon.2019.01.023>. URL: <http://www.sciencedirect.com/science/article/pii/S0926580518307064>.
- [18] Chris Jones. “Attributed graphs, graph-grammars, and structured modeling”. In: *Annals of Operations Research* 38.1 (1992), pp. 281–324.
- [19] Stéphane Lavirotte *et al.* “IoT-based Systems Actuation Conflicts Management Towards DevOps: A Systematic Mapping Study”. In: *IoTbDS*. 2020, pp. 227–234.
- [20] Jean Louis Le Moigne. *La modélisation des systèmes complexes*. Bordas Paris, 1990.
- [21] Renju Liu *et al.* “RemedIoT: Remedial actions for Internet-of-Things conflicts”. In: *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 2019, pp. 101–110.
- [22] Meiyi Ma *et al.* “Detection of runtime conflicts among services in smart cities”. In: *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2016, pp. 1–10.
- [23] Mohammad Nasar and Mohammad Abu Kausar. “Suitability of influxdb database for iot applications”. In: *International Journal of Innovative Technology and Exploring Engineering* 8.10 (2019), pp. 1850–1857.
- [24] Dang Tu Nguyen *et al.* “IoTsan: Fortifying the safety of IoT systems”. In: *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. 2018, pp. 191–203.
- [25] Daniella Niyonkuru and Gabriel Wainer. “Towards a DEVs-based operating system”. In: *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 2015, pp. 101–112.
- [26] Lawrence R. Rabiner. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.
- [27] Gerald Rocher *et al.* “An Actuation Conflicts Management Flow for Smart IoT-based Systems”. In: *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. Paris, France: IEEE, Dec. 2020, pp. 1–8. ISBN: 978-0-7381-2460-5. DOI: 10.1109/IOTSMS52051.2020.9340196. URL: <https://ieeexplore.ieee.org/document/9340196/> (visited on 02/05/2021).
- [28] Gérald Rocher *et al.* “A Possibilistic I/O Hidden Semi-Markov Model for Assessing Cyber-Physical Systems Effectiveness”. In: *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE. 2018, pp. 1–9.

- [29] Gérald Rocher *et al.* “An IOHMM-Based Framework to Investigate Drift in Effectiveness of IoT-Based Systems”. In: *Sensors* 21.2 (2021), p. 527.
- [30] Gérald Rocher *et al.* “Overview and Challenges of Ambient Systems, Towards a Constructivist Approach to their Modelling”. In: *arXiv preprint arXiv:2001.09770* (2020).
- [31] Claude Rochet. “Public Management as a moral science”. Habilitation à diriger des recherches, Université de droit, d’économie et des sciences – Aix-Marseille III, Dec. 2007.
- [32] Alexander Schliep. “A bayesian approach to learning hidden markov model topology with applications to biological sequence analysis”. PhD thesis, Universität zu Köln, 2001.
- [33] Trusit Shah *et al.* “Conflict detection in rule based IoT systems”. In: *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2019, pp. 0276–0284.
- [34] Antero Taivalsaari and Tommi Mikkonen. “A roadmap to the programmable world: software challenges in the IoT era”. In: *IEEE Software* 34.1 (2017), pp. 72–80.
- [35] Arjan J. Van Der Schaft and Johannes Maria Schumacher. *An introduction to hybrid dynamical systems*. Vol. 251. Springer London, 2000.
- [36] Yentl Van Tendeloo and Hans Vangheluwe. “An evaluation of DEVS simulation tools”. In: *Simulation* 93.2 (2017), pp. 103–121.
- [37] Philippe Weber and Christophe Simon. *Benefits of Bayesian network models*. John Wiley & Sons, 2016.
- [38] Muneer Bani Yassein, Wail Mardini, and Ashwaq Khalil. “Smart homes automation using Z-wave protocol”. In: *2016 International Conference on Engineering & MIS (ICEMIS)*. IEEE. 2016, pp. 1–6.
- [39] Bernard P Zeigler, Alexandre Muzy, and Ernesto Kofman. *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press, 2018.
- [40] Xiangrui Zeng and Junmin Wang. “A stochastic driver pedal behavior model incorporating road information”. In: *IEEE Transactions on Human-Machine Systems* 47.5 (2017), pp. 614–624.
- [41] Tao Zheng and Gabriel A Wainer. “Implementing finite state machines using the CD++ toolkit”. In: *Proceedings of the SCS Summer Computer Simulation Conference, 2003. atomic model, 1 CD++, 1 coupled model, 1 DEVS, 1 DEVS Graph, 1 discrete-event modeling*. Citeseer, 2003.