

THESE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE UNIVERSITE BRETAGNE LOIRE

L'UNIVERSITE DU LUXEMBOURG

ECOLE DOCTORALE N° 601
*Mathématiques et Sciences et Technologies
de l'Information et de la Communication*
Spécialité : *Informatique*
Par

Ludovic MOULINE

TOWARDS A MODELLING FRAMEWORK WITH TEMPORAL AND UNCERTAIN DATA FOR ADAPTIVE SYSTEMS

Thèse présentée et soutenue à Luxembourg, le 29/11/2019
Unité de recherche : Équipe DiverSE, IRISA, Rennes et Équipe SerVal, SnT, Luxembourg

Rapporteurs avant soutenance :

Philippe COLLET	Professeur, Université Côte d'Azur, Nice, France
Manuel WIMMER	Professeur, JKU Linz, Linz, Autriche

Composition du Jury :

Président :	Prénom Nom	Fonction et établissement d'exercice (9)(à préciser après la soutenance)
Examineurs :	Nicolas NAVET	Professeur, Université du Luxembourg, Luxembourg, Luxembourg
	Manuel WIMMER	Professeur, JKU Linz, Linz, Autriche
	Philippe COLLET	Professeur, Université Côte d'Azur, Nice, France
	Ada DIACONESCU	Maître de conférence, Telecom ParisTech, Paris, France
Dir. de thèse :	Olivier BARAIS	Professeur, Université de Rennes 1, Rennes, France
Co-dir. de thèse :	Yves LE TRAON	Professeur, Université du Luxembourg, Luxembourg, Luxembourg

Invité(s)

Johann BOURCIER	Maître de conférence, Université de Rennes 1, Rennes, France
François FOUQUET	CTO, DataThings, Luxembourg, Luxembourg

Abstract

Vision: As state-of-the-art techniques fail to model efficiently the evolution and the uncertainty existing in dynamically adaptive systems, the adaptation process makes suboptimal decisions. To tackle this challenge, modern modelling frameworks should efficiently encapsulate time and uncertainty as first-class concepts.

Context Smart grid approach introduces information and communication technologies into traditional power grid to cope with new challenges of electricity distribution. Among them, one challenge is the resiliency of the grid: how to automatically recover from any incident such as overload? These systems therefore need a deep understanding of the ongoing situation which enables reasoning tasks for healing operations. **Abstraction** is a key technique that provides a description of systems, their behaviours, and/or their environments alleviating their complexity. **Adaptation** is a cornerstone feature that enables reconfiguration at runtime for optimising software to the current and/or future situation.

The model-driven engineering (MDE) methodology promotes the use of abstraction in software engineering. However, information concerning the grid, such as loads, is not always known with absolute confidence. Through the thesis, this lack of confidence about data is referred to as **data uncertainty**. They are approximated from the measured consumption and the grid topology. This topology is inferred from fuse states, which are set by technicians after their services on the grid. As humans are not error-free, the topology is therefore not known with absolute confidence. This data uncertainty is propagated to the load through the computation made. If it is neither present in the model nor considered by the adaptation process, then the adaptation process may make suboptimal reconfiguration decisions.

1 The literature refers to systems which provide adaptation capabilities as dynam-
2 ically adaptive systems (DAS). One challenge in the grid is the phase difference
3 between the monitoring frequency and the time for actions to have measurable
4 effects. Actions with no immediate measurable effects are named **long-term ac-**
5 **tions**. On the one hand, an incident should be detected in the next minutes. On
6 the other hand, a reconfiguration action can take up to several hours. For example,
7 when a tree falls on a cable and cuts it during a storm, the grid manager should be
8 noticed in real time. The reconfiguration of the grid, to reconnect as many people
9 as possible before replacing the cable, is done by technicians who need to use their
10 cars to go on the reconfiguration places. In a fully autonomous adaptive system, the
11 reasoning process should consider the ongoing actions to avoid repeating decisions.

12 *Problematic* **Data uncertainty and long-term actions are not specific to**
13 **smart grids.**

14 First, data are, almost by definition, uncertain and developers work with
15 estimates in most cases. Hardware sensors have by construction a precision that
16 can vary according to the current environment in which they are deployed. A simple
17 example is the temperature sensor that provides a temperature with precision to the
18 nearest degree. Software sensors approximate also values from these physical sensors,
19 which increases the uncertainty. For example, CPU usage is computed counting
20 the cycle used by a program. As stated by Intel, this counter is error-prone¹.

21 Second, it always exists a delay between the moment where a suboptimal state
22 is detected by the adaptation process and the moment where the effects of decisions
23 taken are measured. This delay is due to the time needed by a computer to process
24 data and, eventually, to send orders or data through networks. For example,
25 migrating a virtual machine from a server to another one can take several minutes.

26 **Through this thesis, we argue that this data uncertainty and this**
27 **delay cannot be ignored for all dynamic adaptive systems.** To know if
28 the data uncertainty should be considered, stakeholders should wonder **if this**
29 **data uncertainty affects the result of their reasoning process, like adap-**
30 **tation.** Regarding long-term actions, they should verify **if the frequency of the**

¹<https://software.intel.com/en-us/itc-user-and-reference-guide-cpu-cycle-counter>

1 **monitoring stage is lower than the time of action effects to be measur-**
2 **able.** These characteristics are common to smart grids, cloud infrastructure or
3 cyber-physical systems in general.

4 *Challenge* These problematics come with different challenges concerning the
5 representation of the knowledge for DAS. The global challenge addresses by this
6 thesis is: **how to represent an uncertain knowledge that can be efficiently**
7 **queried with ongoing actions in order to improve adaptation processes?**

8 *Vision* **This thesis defends the need for a unified modelling framework**
9 **which includes, besides all traditional elements, temporal and uncer-**
10 **tainty as first-class concepts.** Therefore, a developer will be able to abstract
11 information related to the adaptation process, the environment as well as the
12 system itself.

13 Concerning the adaptation process, the framework should enable abstraction
14 of the actions, their context, their impact, and the specification of this process
15 (requirements and constraints). It should also enable the abstraction of the system
16 environment and its behaviour. Finally, the framework should represent the
17 structure, behaviour and specification of the system itself as well as the actuators
18 and sensors. All these representations should integrate the data uncertainty.

19 *Contributions* Towards this vision, this document presents two contributions: a
20 temporal context model and a language for uncertain data.

21 The temporal context model allows abstracting past, ongoing and future actions
22 with their impacts and context. First, a developer can use this model to know
23 what the ongoing actions, with their expected future impacts on the system, are.
24 Second, she/he can navigate through past decisions when they led to a sub-optimal
25 state to understand why they have been made .

26 The language, named Ain'tea, integrates data uncertainty as a first-class citizen.
27 It allows developers to attach data with a probability distribution which represents
28 their uncertainty. Besides, it maps all arithmetic and boolean operators to un-
29 certainty propagation operations. Thus, developers can automatically propagate
30 the uncertainty of data without additional effort, compared to an algorithm which
31 manipulates certain data.

1 *Validation* Each contribution has been evaluated separately. First, the context
2 model has been evaluated through the performance axis. The dissertation shows
3 that it can be used to represent the Luxembourg smart grid. The model also
4 provides an API which enables the execution of query for diagnosis purpose. In
5 order to show the feasibility of the solution, it has also been applied to the use case
6 provided by the industrial partner.

7 Second, the language has been evaluated through two axes: its ability to
8 detect errors at development time and its expressiveness. Ain'tea can detect
9 errors in the combination of uncertain data earlier than state-of-the-art approaches.
10 The language is also as expressive as current approaches found in the literature.
11 Moreover, we use this language to implement the load approximation of a smart
12 grid furnished by an industrial partner, Creos S.A.².

13
14 **Keywords:** dynamically adaptive systems, knowledge representation, model-
driven engineering, uncertainty modelling, time modelling

²Creos S.A. is the power grid manager of Luxembourg. <https://www.creos-net.lu>

Contents

I Context and challenges in modelling adaptive systems

1

1 Introduction

3

1.1	Context	4
1.2	Challenges	5
1.2.1	Engineering uncertainty-aware software	5
1.2.2	Reasoning over long-term actions	7
1.2.3	Diagnosing the adaptation process	8
1.2.4	Modelling inconsistent states of systems	9
1.2.5	Modelling temporal and interconnected data	10
1.3	Scope of the thesis	10
1.4	Contribution & validation	11
1.5	Structure of the document	13

2 Background

15

2.1	Adaptive systems	16
2.1.1	Principles and vision	16
2.1.2	Models@run.time	19
2.1.3	Characterisation of information of the knowledge	20
2.1.4	Key concepts for this thesis	22
2.2	Model-Driven Engineering	23
2.2.1	Principles and vision	23
2.2.2	Metamodel, model	25

1	2.2.3	Tooling	26
2	2.2.4	Concepts used in this thesis	28
3	2.3	Software Language Engineering	28
4	2.3.1	Software Languages	28
5	2.3.2	SLE in this thesis	30
6	2.4	Probability theory	30
7	2.4.1	Random variables	30
8	2.4.2	Distribution	31
9	2.4.3	Distribution used in this thesis	32
10	3	Motivating example: smart grid	35
11	3.1	Smart grid overview	36
12	3.2	Data uncertainty	38
13	3.2.1	Impacts of ignoring data uncertainty	38
14	3.2.2	Managing uncertainty is not effortless	38
15	3.3	Long-term actions	43
16	3.3.1	Examples	43
17	3.3.2	Use case scenario	48
18	4	State of the art	51
19	4.1	Review methodology	52
20	4.2	Results RQ1: long-term actions	54
21	4.2.1	Modelling the evolution of system's context, structure, or	
22		behaviour	54
23	4.2.2	Modelling actions, their circumstances, and their effects . . .	60
24	4.2.3	Reasoning over evolving context or behaviour	63
25	4.2.4	Modelling and reasoning over long-term actions	67
26	4.3	Results RQ2: data uncertainty	67
27	4.3.1	Categories of data uncertainty	67
28	4.3.2	Modelling data uncertainty	69
29	4.3.3	Propagation and reasoning over uncertainty	73
30	4.3.4	Modelling of data uncertainty and its manipulation	75
31	4.4	Threat to validity	75

1	4.5	Conclusion	76
2	II	Towards a modelling frameworks for adaptive sys-	
3		tems	79
4	5	Uncertainty and time in modeling frameworks	81
5	5.1	Summary of previous chapter	82
6	5.2	Vision	82
7	6	Ain’tea: managing data uncertainty at the language level	85
8	6.1	Uncertainty as a first-class language citizen	87
9	6.1.1	Language overview	87
10	6.1.2	Uncertain boolean	88
11	6.1.3	Uncertain number	94
12	6.1.4	Uncertain references	102
13	6.1.5	Static semantic: typing rules	104
14	6.2	Evaluation	105
15	6.2.1	Ain’tea: our implementation	106
16	6.2.2	Conciseness	109
17	6.2.3	Error handling at development time	112
18	6.2.4	Discussion	114
19	6.3	Conclusion	115
20	7	A temporal knowledge metamodel of adaptive systems	117
21	7.1	Knowledge formalization	118
22	7.1.1	Formalization of the temporal axis	119
23	7.1.2	Formalism of the knowledge	120
24	7.1.3	Application on the use case	124
25	7.2	Modelling the knowledge	128
26	7.2.1	Parent element: <i>TimedElement</i> class	129
27	7.2.2	Knowledge metamodel	129
28	7.2.3	Context metamodel	130
29	7.2.4	Requirement metamodel	131

1	7.2.5	Action metamodel	132
2	7.3	Validation	133
3	7.3.1	Diagnostic: implementation of the use case	134
4	7.3.2	Reasoning over unfinished actions and their expected effects	137
5	7.3.3	Performance evaluation	139
6	7.3.4	Discussion	142
7	7.4	Conclusion	144
8	III	Conclusion and future work	145
9	8	Conclusion	147
10	8.1	Summary	148
11	8.2	Future work	150
12	8.2.1	Software language and uncertain data	150
13	8.2.2	(Self-)Explanation of adaptive systems	155
14	Glossary		i
15	Abbreviations		v
16	List of publications and tools		vi
17	List of figures		viii
18	List of tables		x
19	French summary - Résumé en français		xiii
20	8.3	Contexte	xiv
21	8.4	Définitions	xv
22	8.4.1	Ingénierie de logiciels sensibles à l'incertitude	xvi
23	8.4.2	Raisonnement sur les actions à long terme	xvii
24	8.4.3	Diagnostic du processus d'adaptation	xix
25	8.4.4	Modélisation des états incohérents des systèmes	xx
26	8.4.5	Modélisation des données temporelles et interconnectées	xxi
27	8.5	Présentation de la thèse	xxii

1	8.6 Contribution & validation	xxiii
1	Bibliography	xxvii

2

Part I

3

Context and challenges in modelling adaptive systems

1

3 Introduction

4 Contents

5	1.1	Context	4
6	1.2	Challenges	5
7	1.3	Scope of the thesis	10
8	1.4	Contribution & validation	11
9	1.5	Structure of the document	13

10
11
12
13

14 *Model-driven engineering methodology and dynamically adaptive systems ap-*
15 *proach are combined to tackle challenges brought by systems nowadays. After*
16 *introducing these two software engineering techniques, we describe five problems*
17 *that we identified for such systems: data uncertainty, actions with long-term effects,*
18 *emergent behaviours of such systems, different evolution paces of the subparts,*
19 *and the temporal dimension in their structures and behaviours. We present the*
20 *challenges that come with these problems. Before describing the two contributions*
1 *of this thesis, we scope to the addressed sub-challenges tackled.*

2 1.1 Context

3 Self-Adaptive Systems (SAS) optimize their behaviours or configurations at run-
4 time in response to a modification of their environments or their behaviours [CdLG⁺09].
5 Kephart and Chess [KC03] laid the groundwork for this approach, based on an
6 IBM white paper [Com⁺06]. Since then, practitioners have applied it to differ-
7 ent domains [GVD19] such as cloud infrastructure [JG17; Tea15; BKF⁺17] or
8 Cyber-Physical System (CPS) [LGC17; FMF⁺12; HFK⁺14a]. One example of such
9 a system is a smart grid, which employs the adaptation capacity to heal itself
10 autonomously.

11 A smart grid is a power grid in which utilities introduce Information and
12 Communication Technology (ICT) to face the new challenges of electricity sup-
13 ply [Far10; IA09; FMX⁺12]. One of the required feature is the self-healing capacity.
14 A self-healing system can automatically repair any incident, software or hardware,
15 at runtime [KC03]. For example, a smart grid can optimise the power flow to deal
16 with failures of transformers¹ [FMX⁺12].

17 The adaptation process can be performed only if the system has a deep under-
18 standing of the situation and the problem. In this case, the situation comprises the
19 structure (elements that compose the system), the behaviour (the set of possible
20 executions of the system) and the environment (where it is executed) of the system.
21 This understanding can be extracted from an, or a set of, **abstraction**(s) of these
22 elements. Abstractions provide a description of systems, their behaviours, or their
23 environments. For example, Hartmann *et al.*, [HFK⁺14b] provide a class diagram
24 that describes the smart grid topology, when it uses power lines communications².

25 **Model-Driven Engineering (MDE)** defenders argue for using the abstrac-
26 tion mechanism to facilitate the development of current software [Sch06; Ken02;
27 BCW17]. This methodology can be applied to different stages of software develop-
28 ment. In this thesis, we focus on one of its paradigms: **models@run.time** [BBF09;
29 MBJ⁺09]. As we depict in Figure 1.1, using this paradigm, the adaptation process
30 relies on a model for analysing the situation and triggering the adaptation. In
1 this document, we say that the model represents the knowledge of the adaptation

¹Transformers change the voltage in the cables.

²Data are sent through cables that also distribute electricity.

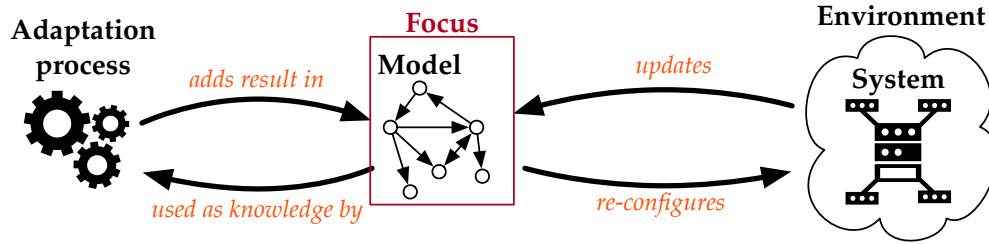


Figure 1.1: Overview of the models@run.time and focus of the thesis

process. Developers can use this paradigm to implement adaptive systems [MBJ⁺09; HFK⁺14a]. This dissertation contributes to this modelling layer.

1.2 Challenges

During our study, we have identified five characteristics of adaptive systems that bring challenges to the software engineering research community. First, information gathered is not always known with absolute confidence. Second, reconfigurations may not be immediate, and their effects are not instantaneously measured. Third, system behaviour may be emergent [ZA11], *i.e.*, it cannot be entirely known at design time. Four, the different sub-parts of the system do not evolve at the same rate. Five, structure and behaviour of systems have a time dimension. The last one has been published in our vision paper regarding time awareness in MDE [BHM⁺17]. We detailed them in this section.

1.2.1 Engineering uncertainty-aware software

Most fuses are manually opened and closed by technicians rather than automatically modified. Then, technicians manually report the modifications done on the grid. Due to human mistakes, this results in errors. The grid topology is thus uncertain. This uncertainty is propagated to the load approximation, used to detect overloads in the grid. Wrong reconfigurations might be triggered, which could be even worse than if no change would have been applied.

More generally, **data are, almost by definition, uncertain and developers work with estimates in most cases** [BMM14; Met08; AY09]. The uncertainty may be explained by how data are collected. We can distinguish three categories: sensors, humans, and results of computations. Sensors (software or

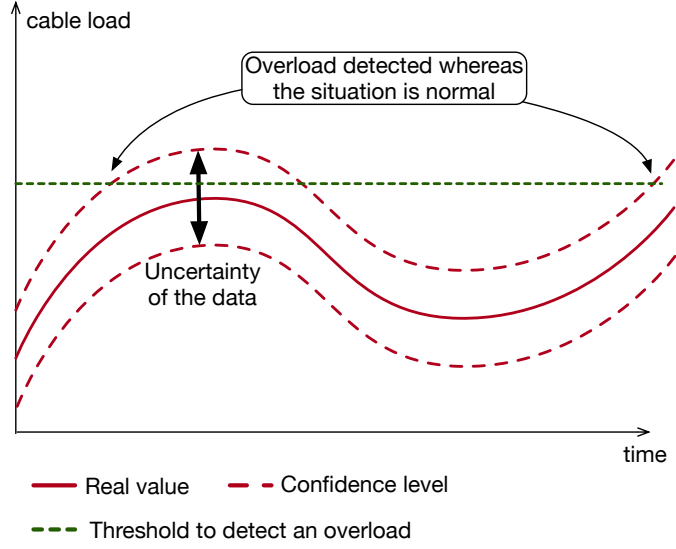


Figure 1.2: Illustration of the problem due to data uncertainty

hardware) always estimate the value and have a precision value due to the method of measurement [Met08; BMM14]. Humans are error-prone. Computations can either give an approximation or be based on uncertain data. This uncertainty is then propagated through all steps until the final result.

For a specific domain, this uncertainty may impact the understanding of the real situation as depicted in Figure 1.2. For example, the uncertainty of the Central Processing Unit (CPU) clock is too low to damage the percentage load of the processor. However, the uncertainty of the cable load in a smart grid may trigger false detection of an overload, as depicted in Figure 1.2. **If the data uncertainty can mislead the understanding of a system behaviour or state, then developers should implement an uncertainty-aware system.** For adaptive systems, this lack of confidence may trigger suboptimal adaptations.

Therefore, we argue that data uncertainty impacts all the development stages of software, from the design to the execution. Among the different stages, in this thesis we focus on the design one. We firmly think that design techniques should provide mechanisms to help developers abstract and manipulating uncertain data.

The literature provides approaches to help engineers reason or manipulate data uncertainty, or at least probability distributions. For example, believe functions [Sha76] help to reduce this uncertainty by combining several sources of data.

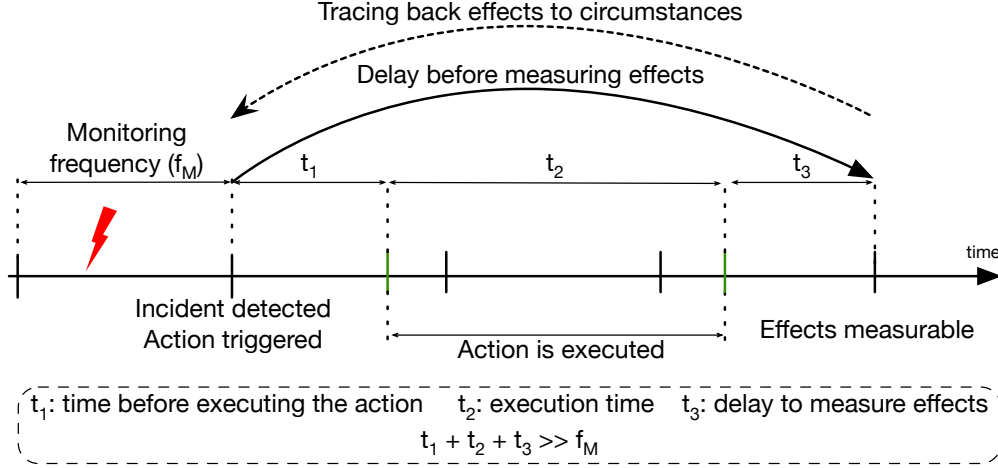


Figure 1.3: Illustration of a long-term action

The probabilistic programming [GHN⁺14] community provide frameworks and languages [MWG⁺18; BDI⁺17] to propagate probabilities through computations.

However, from the best of our knowledge, no study has been done to evaluate the impact of data uncertainty on the development of software. The following challenge still remains an open question for the software engineering community:

How to engineer uncertainty-aware software (design, implement, test, and validate)?

1.2.2 Reasoning over long-term actions

Reconfiguring a smart grid implies to change the power flow by opening or closing fuses. As said before, technicians need to drive physically to fuse locations to modify their states. In the case of the Luxembourg smart grid, meters send energy measurement every 15 min, non-synchronously. Therefore, between the time a reconfiguration of the smart grid is decided, and the time the effects are measured, a delay of at least 15 min occurs. On the other hand, an incident should be detected in the next minutes. If the adaptation process does not consider this difference of rates, it can cause repeated decisions.

More generally, a difference may exist between the monitoring frequency and the time for action effects to be measured. One cause of this is what we call long-term actions in this document, illustrated in Figure 1.3. A long-term action is defined as an action that takes time to be executed (delay to be executed and execution time)

2 or that have long-term effects. A second cause is an impossibility to reduce the
3 monitoring frequency since systems must be reactive in some cases. This difference
4 in rates may damage the decision process.

5 Therefore, we argue that **decision-making processes should consider this**
6 **delay if the frequency of the monitoring stage is lower than the time of**
7 **action effects to be measurable**. From the best of our knowledge, none of the
8 approaches allows developers implementing such tools. One open challenge for the
9 research community is thus:

10
11

How to model, store, and query long-term actions with their effects?
--

12 1.2.3 Diagnosing the adaptation process

13 Smart grid behaviour is affected by several factors that cannot be controlled
14 by the grid manager. One example is weather conditions. Smart grids rely on an
15 energy production distributed over several actors. For instance, users, who were
16 mainly consumers before, can now produce energy by adding solar panels on the
17 roof of their houses. The production of such energy depends on the weather, and
18 even on the season³. Despite this stochasticity of the behaviour, engineers need to
19 implement an adaptation process, that can lead to suboptimal grid configuration.

20 Faced with growingly complex and large-scale software systems (e.g. smart
21 grid systems), we can all agree that the presence of residual defects becomes
22 unavoidable [BdMM⁺17; MPS15; HBB15]. Even with a meticulous verification
23 or validation process, it is very likely to run into an unexpected behaviour that
24 was not foreseen at design time. Alone, existing formal modelling and verification
25 approaches may not be sufficient to anticipate these failures [TOH17]. As such,
26 complementary techniques need to be proposed to locate the anomalous behaviour
27 and its origin in order to handle it in a safe way.

28 Bencomo *et al.*, [BWS⁺12] argue that comprehensive explanation about the sys-
29 tem behaviour contributes drastically to the quality of the diagnosis, and eases the
30 task of troubleshooting the system behaviour. To enable this, as shown in Figure 1.3,
31 we believe that adaptive software systems should be equipped with traceability
1 management facilities to link the decisions made to their **(i) circumstances, that**

³The angle of the sun has an impact on the amount of energy produced by solar panels. This angle varies according to the season.

is to say, the history of the system states and the targeted requirements, and (ii) the performed actions with their impact(s) on the system. In particular, an adaptive system should keep a trace of the relevant historical events. Additionally, it should be able to trace the goals intended to be achieved by the system to the adaptations and the decisions that have been made, and vice versa. Finally, in order to enable developers to interact with the system in a clear and understandable way, appropriate abstraction to enable the navigation of the traces and their history should also be provided. In other words, one global challenge that remains unaddressed is:

How to trace back adaptation decision effects to their circumstances?

1.2.4 Modelling inconsistent states of systems

Every meter sends consumption and production data every 15 min. However, this collection is not synchronous. That is, not all meters send their data at the same timestamp. The global system, which receives all data, has not thus a global vision with the same freshness for all the part of the grid. Electricity data are volatile: a peak or a drop may happen in less than a minute due to, for instance, the starting or the finishing of a washing machine. Reconfiguration of the grid may thus be suboptimal due to outdated information.

Different parts of a system may evolve at different rates. Some systems are heterogeneous in terms of hardware and software. This diversity results in different evolution or reaction rates. For example, if some components are working on batteries, they will have a sleep cycle to save energy. Contrary, if some others are running connected directly to a power source, they can react faster.

Despite this difference of rates, a global vision of a system at a precise time point may be still required. The vision should deal with data that have different freshness. For example, the adaptation process may require a global view of the system. In the worst case, some data would be outdated and cannot be used.

When designing the adaptation process, engineers need thus solutions to deal with an inconsistent system state. One solution can, for example, seamlessly estimate what should be the current value of outdated data. One global challenge for the software engineering community is therefore:

How to represent, query, and store inconsistent system states and behaviours?

2 1.2.5 Modelling temporal and interconnected data

3 Power flow is impacted by consumption and production of users, and by the
4 modifications of the topology. Knowing the last status of the grid is as important as
5 knowing how it evolves. Based on the evolution, the grid operator can predict any
6 future incidents, like overloads. It could also compare this evolution of behaviour
7 with a normal one to detect, for example, any malicious behaviour.

8 **Evolution of systems is inherently linked with a time dimension.** Evo-
9 lution and time are two related concepts. For some systems, not only the last states
10 are important but also how they evolve. Then, analysis processes will investigate
11 this evolution to know if it is a normal one or not. They can also use this evolution
12 to predict how systems will evolve. Based on these predictions, they can proact on
13 future incidents in the system.

14 Decisions are not made upon the last state of the system but how it evolves.
15 The analysis process should thus navigate both in the structure of the system
16 and its behaviour over time. **Engineers need efficient tooling to structure,**
17 **represent, query, and store temporal and interconnected data on a large**
18 **scale.**

19 Time in software engineering is not a new challenge. For example, Riviera *et*
20 *al.*, [RRV08] have already identified time as a challenge for the MDE community.
21 Different approaches have been defined [BCC⁺15; KT12; KH10; HFN⁺14b].

22 However, we notice that research efforts made by the MDE community did
23 not focus on the modelling, persistence, and processing of evolving data. Thomas
24 Hartmann started addressing these challenging in his PhD thesis [Har16]. The final
25 global challenge, not fully addressed, is thus:

26

27 How to structure, represent, query, and store efficiently temporal data on a large scale?

28 1.3 Scope of the thesis

29 Among all the challenges described in the previous section, this thesis focuses on
30 three of them: data uncertainty (Section 1.2.1), long-term actions (Section 1.2.2),
31 and error-prone adaptation process (Section 1.2.3). More precisely, we address
1 three sub-problems of these challenges.

Managing uncertainty requires significant expertise in probability and statistics theory. The literature provides different solutions to manage uncertainty [Zad; Met08; Sha76]. The application of these techniques requires a deep understanding of the underlying theories and is a time-consuming task [VMO16]. Moreover, it is hard to test and perhaps most importantly, very error-prone. In this thesis, we address thus the following problem:

Sub-challenge #1: How to ease the manipulation of data uncertainty for software engineers?

Adaptation processes may rely on long-term action like resource migration in cloud infrastructure. Due to the lack of information about unfinished actions and their expected effects on the system, the reasoning component may take repeated or sub-optimal decisions. One step for enabling this reasoning mechanism is to have an abstraction layer which can represent these long-term actions efficiently. In this thesis, we, therefore, cope with the following challenge:

Sub-challenge #2: How to enable reasoning over unfinished actions and their expected effects?

Due to the increasing complexity of systems, developers have difficulties in delivering error-free software [BdMM⁺17; MPS15; HBB15]. Moreover, complex systems or large-scale systems may have emergent behaviours. Systems very likely have an abnormal behaviour that was not foreseen at design time. Existing formal modelling and verification approaches may not be sufficient to verify and validate such processes [TOH17]. In such situations, developers usually apply diagnosis routines to identify the causes of the failures. During our studies, we tackle the following challenge:

Sub-challenge #3: How to model the decisions of an adaptation process to diagnose it?

1.4 Contribution & validation

In this thesis, we argue that modern modelling frameworks should consider uncertainty and time as first-class concepts. In this dissertation, I present two contributions that support this vision. First, we define a language with uncertainty at a first-class citizen: Ain'tea. We detail this contribution in Chapter 6. Second, we define a metamodel, and we formalise it, of the knowledge of adaptive systems. We present this contribution in Chapter 7.

```

Gaussian<double> average(Gaussian<double>[] arrays) {
    Gaussian<int> sum = arrays[0];
    for(int i = 0; i < arrays.length; i++) {
        sum = sum + arrays[i];
    }
    return sum / arrays.length;
}

```

Gaussian distribution can only be applied on (float, double). Actual: int

Figure 1.4: Overview of the language proposed, Ain'tea

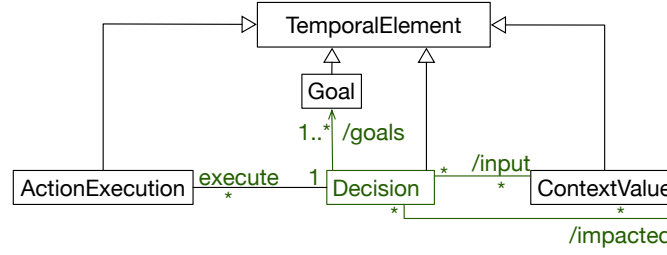


Figure 1.5: Overview of the temporal knowledge model

2 **Ain'tea: Managing Data Uncertainty at the Language Level** This con-
3 tribution addresses the challenge of the manipulation of uncertain data (cf. Sub-
4 Challenge #1). We propose Ain'tea, a language able to represent uncertain data
5 as built-in language types along with their supported operations. An overview
6 of the language is depicted in Figure 1.4. It contains a sampling of distributions
7 (Gaussian, Bernoulli, binomial, Dirac delta function, and Rayleigh) that covers
8 the different data types (booleans, numbers, and references). We implement a
9 prototype of the language, publicly available on GitHub⁴. We use a real-world case
10 study based on smart grid, built with our partner Creos S.A.. It shows first that
11 our approach does not impact the conciseness of the language. Second, it highlights
12 the feasibility and the advantages of uncertainty-aware type checking systems on
13 the language level.

14 This contribution is under submission at the JOT Journal⁵:

- 15 • “Ain'tea: Managing Data Uncertainty at the Language Level”, Mouline,
1 Benelallam, Hartmann, Bourcier, Barais, and Cordy

⁴<https://github.com/lmouline/aintea/>

⁵<http://www.jot.fm/>

A temporal knowledge metamodel This contribution addresses the challenge of reasoning over unfinished actions, and understanding of adaptive system behaviour (cf. Sub-Challenge #2 and #3). First, we formalise the common core concepts implied in adaptation processes, also referred to as knowledge. The formalisation is based on temporal graphs and a set of relations that trace decision impacts to circumstances. Second, we propose a framework to structure and store the state and behaviour of a running adaptive system, together with a high-level Application Programming Interface (API) to efficiently perform diagnosis routines. Our framework relies on a temporal model-based solution that efficiently abstracts decisions, their corresponding circumstances, and their effects. We give an overview of the metamodel in Figure 1.5. We demonstrate the applicability of our approach by applying it to a smart grid based example. We also show that our approach can be used to diagnose the behaviour of at most the last five days of a district in the Luxembourg smart grid in ~ 2.4 seconds.

Part of this contribution has been published at the IEEE International Conference on Autonomic Computing⁶ (ICAC) and at the ACM/SIGAPP Symposium On Applied Computing⁷ (SAC):

- “Enabling temporal-aware contexts for adaptative distributed systems”, Mouline, Benelallam, Hartmann, Fouquet, Bourcier, Morin, and Barais
- “A Temporal Model for Interactive Diagnosis of Adaptive Systems”, Mouline, Benelallam, Fouquet, Bourcier, and Barais

1.5 Structure of the document

We split the remaining part of this document into seven chapters, as shown in Figure 1.6. First, Chapter 2 describes the necessary background of the thesis. Then, Chapter 3 describes a motivating example, based on a smart grid system. We present concepts related to MDE and adaptive systems. Based on this background, we show the gap of the current state of the art in Chapter 4. Chapter 5 explains our vision. Chapter 6 and Chapter 7 describe our two contributions. The former details our language, Ain’t tea, that integrates uncertainty as a first-class citizen. The latter explains our temporal metamodel that can represent past and ongoing

⁶<http://icac2018.informatik.uni-wuerzburg.de/>

⁷<http://www.sigapp.org/sac/sac2018/>

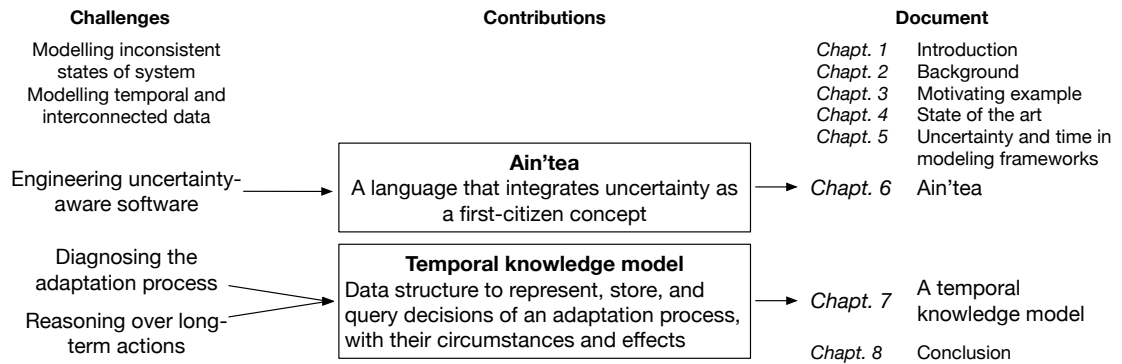


Figure 1.6: Structure of the document

2 actions with their circumstances and effects. Finally, we conclude in Chapter 8,
 1 and we present future work.

2

3 Background

4 Contents

5	2.1 Adaptive systems	16
6	2.2 Model-Driven Engineering	23
7	2.3 Software Language Engineering	28
8	2.4 Probability theory	30

9

10

11

12

13 *In this chapter, we describe the principles of the different software engineering*
14 *approaches related to this thesis: adaptive systems, Model-Driven Engineering*
15 *(MDE), and Software Language Engineering (SLE). Moreover, we introduce the*
16 *elements of the probability theory used in this thesis. For each of them, we also*
17 *detail the concepts used in our work, and we explicitly link them to our vision and*
1 *our contributions.*

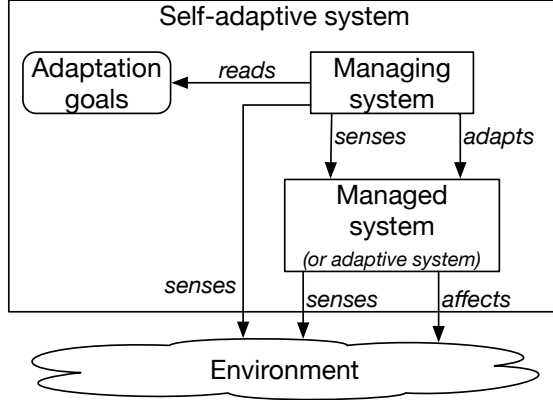


Figure 2.1: Conceptual vision of adaptive system (based on [Wey19])

2.1 Adaptive systems

This section introduces the background to understand adaptive systems. First, we describe the principles and vision of adaptive systems. Before we characterise the information used by an adaptation process, we detail a model-based technique to implement them: models@run.time. Finally, we highlight the key concepts used in this thesis and link them to the contributions.

2.1.1 Principles and vision

Adaptive systems take their origins in the vision paper of Kephart and Chess [KC03], which is based on the autonomic computing vision pushed by IBM engineers [Com⁺06]. These systems are recognised by their ability to have their behaviour or structure adapted automatically in response to changes in their environment or of the systems themselves. This adaptation helps them to achieve their goals based on high-level objectives [CdLG⁺09]. If a system performs itself this adaptation mechanism with minimal interference, the literature refers to it as a SAS [BSG⁺09].

Danny Weyns identified two principles for adaptive systems [Wey19]: the internal and the external principles. The former one is based on the “discipline split” defined by Andersson *et al.*, [AdLM⁺09]: each adaptive system can be split into two concerns. First, the domain concern categorises the part of the system that deals with the purpose for which the system has been built. Second, the adaptation concern handles the adaptation mechanism and interacts with the first

one without interfering with it [KM90]. The external principle says that adaptive systems should handle changes and uncertainties in their environment, the managed systems, and the goal autonomously.

In addition to these principles, the literature has defined four adaptation goals, usually called the self-* features [Com⁺06]: self-healing, self-optimising, self-configuring, and self-protecting. First, the healing capacity, defined when the failures in the system can be automatically discovered, diagnosed, and repaired. Second, the adaptation mechanism can be used to optimise the system by tuning resources and balancing workloads. Third, the system can be autonomously configured for dynamically adapting to the changes in the environment. Four, threats can be anticipated, detected, and identified by the adaptation process to protect the managed system. Besides, we can add the self-organisation feature [Dem98]: adaptive systems can “acquire and maintain themselves, without external control.” [WH04]. It is mainly discussed for distributed systems, where local rules are applied to adjust their interactions and act co-operatively for adaptation. However, this mechanism can lead to emergent behaviour [WH04].

As depicted in Figure 2.1, each adaptive system is composed of four elements: the environment, the managed system, adaptation goals, and the managing system [Wey19]. The environment includes all external entities, virtual or physical, with which the adaptive system interacts on each it effects [Jac97]. Only the elements that are monitored are part of the system. One may distinguish the environment to the adaptive system as, contrary to the element of the adaptive system, it cannot be directly impacted by the engineer. The managed system evolves in the environment and covers all the part of the system that implements the domain concern. In the literature, researchers use different names to refer to it: managed element [KC03], system layer [GCH⁺04], core function [ST09], base-level subsystem [WMA12], or controllable plant [FHM14]. To enable the adaptation process, the managed system should contain sensors, for monitoring, and actuators, for modifications. This adaptation process needs adaptation goals to perform. They are related to the managed system and mainly concern its software quality metrics [WA13]. At the roots of the self-* features, Kephart and Chess have defined four families of goals: configuration, optimisation, healing, and protection [KC03]. Engineers can redefine these goals over time, which should consider the uncertainty

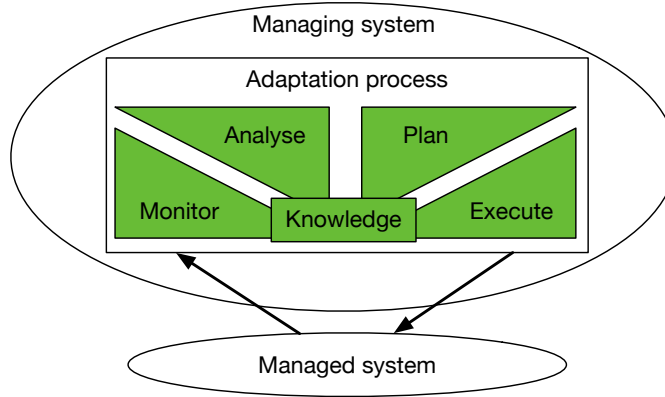


Figure 2.2: MAPE-k loop (based on [KC03])

2 of the environment or the system. To express such goals, different approaches have
3 been defined, such as probabilistic temporal logic [CGK⁺] or fuzzy goals [BPS10].
4 Finally, the managing system will use these goals to drive the adaptation of the
5 managed system in response to changes in the environment. It thus continuously
6 monitor the environment and the managing system. Researchers use different names
7 to refer to this element: autonomic manager [KC03], architecture layer [GCH⁺04],
8 adaptation engine [ST09], reflective subsystem [WMA12], controller [FHM14].

9 In the literature, we can find different approaches to engineer adaptive sys-
10 tems [GCH⁺04]. Among them, the most used one took its inspiration from control
11 theory [BSG⁺09]: the feedback control loop. The common implementation is
12 the Monitor, Analyse, Plan, and Execute over knowledge (MAPE-k) loop [KC03;
13 Com⁺06], shown in Figure 2.2. This loop is split into four phases: monitoring,
14 analyse, planning, and executing. During the monitoring phase, all information of
15 the managed system and the environment are put into the knowledge. Based on
16 the updated knowledge, the analyse phase detects any need for adaptation using
17 the adaptation goals. If any, the planning phase computes the set of actions to
18 adjust the managing system structure or behaviour. Finally, the executing phase
19 completes the plan. From the MDE community, one approach to implement such
20 a feedback look is to use the models@run.time paradigm, explained in the next
1 section.

Element	Description
Context	Context data are characterised by: their volatility, their temporality, their uncertainty, their source, or their connection.
Requirement	Three kinds of requirements: performance, specific quality, and constraint.
Action	Four pieces of information characterise an action: requirement(s) to achieve, initial context, effects, and execution information (start time, end time, status).

Table 2.1: Characterisation of information of the knowledge

2.1.2 Models@run.time

The adaptation process needs to have a deep understanding of the system and its environment. Following the MDE methodology¹, research efforts have led to the models@run.time paradigm [MBJ⁺09; BBF09]. It stresses the use of a model layer, causally connected to the system, and used by the adaptation process. The causal connection encompasses two features of the model. First, the model reflects the up-to-date state of the system (structure and behaviour) and its environment. Second, any modification of the model triggers a modification of the system. In this way, the model can be used as an interface between the system and the adaptation process, as shown in Figure 1.1. The layer can contain several models to represent different aspect of the system at runtime [MBJ⁺09]: structure, behaviour, environment, *etc.* Moreover, it can contains representation that guide the adaptation process [CGF⁺09; HHP⁺08] such as configurations point.

Following the MAPE-k, the model layer should structure the knowledge. In this thesis, we define a metamodel of this knowledge to enable developers diagnosing, understanding, and reasoning over long-term action (*cf.* Chapter 7). In the next section, we thus characterise the information that composes the knowledge.

2.1.3 Characterisation of information of the knowledge

General concepts of adaptation processes

Similar to the definition provided by Kephart [KC03], IBM defines adaptive systems as “a computing environment with the ability to manage itself and **dynamically adapt** to change in accordance with **business policies and objectives**. [These systems] can perform such activities based on **situations they observe or sense in the IT environment** [...]” [Com⁺06].

Based on this definition, we can identify three principal concepts involved in adaptation processes. The first concept is **actions**. They are executed to perform a dynamic adaptation through actuators. The second concept is **business policies and objectives**, which is also referred to as the **system requirements** in the domain of (self-)adaptive systems. The last concept is the observed or sensed **situation**, also known as the **context**. The following subsections provide more details about these concepts. Table 2.1 gives an overview of these different elements.

Context

In this thesis, we use the widely accepted definition of context provided by Dey [Dey01]: “Context is **any information that can be used to characterize** the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and [the system], including the user and [the system] themselves”. In this section, we list the characteristics of this information based on several works found in the literature [HIR02; HFN⁺14b; BBH⁺10; PZC⁺14]. We use them to drive our design choices of our *Knowledge* metamodel (*cf.* Section 7.2.2).

Volatility Data can be either **static** or **dynamic**. Static data, also called frozen, are data that will not be modified, over time, after their creation [HIR02; MSS13; BBH⁺10]. For example, the location of a machine, the first name or birth date of a user can be identified as static data. Dynamic data, also referred to as volatile data, are data that will be modified over time.

¹MDE is a methodology that promotes the usage of models for software engineering activities (*cf.* Section 2.2)

Temporality In dynamic data, we may be interested not only in storing the latest value, but also the previous ones [HFN⁺14b; HIR02]. We refer to these data as **historical** data. Temporal data is not only about past values, but also future ones. Two kinds of future values can be identified, **predicted** and **planned**. Thanks to machine learning or statistical methods, dynamic data values can be **predicted**. **Planned** data are set by a system or a human to specify planned modification on the data.

Uncertainty One of the recurrent problems facing context-aware applications is the data uncertainty [dLGM⁺10; HIR02; MSS13; BBH⁺10]. Uncertain data are not likely to represent reality. They contain noise that makes it deviate from its real value. This noise is mainly due to the inaccuracy and imprecision of sensors. Another source of uncertainty is the behaviour of the environment, which can be unpredictable. All the computations that use uncertain data are also uncertain by propagation.

Source In this dissertation, we distinguish three main categories of data source: sensed, computed, profiled, and given. Sensed data are those measured by a sensor (hardware or software). Computed data result from a process that combine different data. And profiled data are those that are learned. Finally, given data are static data manually set in a software.

Connection Context data entities are usually linked using three kinds of connections: conceptual, computational, and consistency [HIR02; BBH⁺10]. The conceptual connection relates to (direct) relationships between entities in the real world (e.g. smart meter and concentrator). The computational connection is set up when the state of an entity can be linked to another one by a computation process (derived, predicted). Finally, the consistency connection relates to entities that should have consistent values. For instance, temperature sensors belonging to the same geographical area.

Requirement

Adaptation processes aim at modifying the system state to reach an optimal one. All along this process, the system should respect the **system requirements** established ahead. Through this thesis, we use the definition provided by IEEE [III17]:

2 “(1) Statement that translates or expresses a need and its associated **constraints**
3 and **conditions**, (2) **Condition or capability that must be met or possessed**
4 by a system [...] to satisfy an agreement, standard, specification, or other formally
5 imposed documents”.

6 Although in the literature, requirements are categorised as functional or non-
7 functional, in this document we use a more elaborate taxonomy introduced by
8 Glinz [Gli07]. It classifies requirements in four categories: functional, performance,
9 specific quality, and constraint. All these categories share a common feature: they
10 are all temporal. During the life-cycle of an adaptive system, a stakeholder can
11 update, add or remove some requirements [CA07; PSR10].

12 **Action**

13 In the IEEE Standards [III17], an action is defined as: “**process of trans-**
14 **formation** that **operates upon data** or other types of inputs to create data,
15 produce outputs, or **change the state** or condition of the subject software”.

16 In the context of adaptive systems, we can define an action as a process that,
17 given the context and requirements as input, adjusts the system behaviour. This
18 modification will then create new data that correspond to an output context. In
19 the remainder of this document, we refer to the output context as impacted context,
20 or effect(s). Whereas requirements are used to add preconditions to the actions,
21 context information is used to drive the modifications. Actions executions have a
22 start time and a finish time. They can either succeed, fail, or be cancelled by an
23 internal or external actor.

24 **2.1.4 Key concepts for this thesis**

25 Adaptive systems have been proposed to tackle the growing complexity of
26 systems (structure and behaviour) and their environment. One common model for
27 implementing them is the well-known MAPE-k loop, a feedback control loop based
28 on shared knowledge. Applying the models@run.time paradigm, this knowledge
29 can be structured by a model, which is causally connected to the system. This
30 model should represent the uncertain and time dimension of the knowledge. In
31 this thesis, we consider that the knowledge comprises information related to the
1 context, the actions, and the requirements. In this thesis, we propose a meta-

model² to represent the decisions made by the adaptation process over time (*cf.* Chapter 7). Additionally, we define a language, Ain'tea, to propagate uncertainty in the computation made during the adaptation process (*cf.* Chapter 6).

2.2 Model-Driven Engineering

Abstraction, also called modelling, is the heart of all scientific discipline, including computer science [Kra07]. One will abstract a system, computer or not, a problem, or a solution to reason about it for a specific purpose. Abstraction reduces the scope to relevant parts, removing the extra that complexify the understanding. For example, a climatologist models all elements that impact the global climate (wind, ocean current, temperature, ...), ignoring local information, like the temperature under a forest, or global ones, like the solar system. In contrast, astronomers model the solar system, ignoring all information regarding the Earth climate.

In computer science, different modelling formalisms have been proposed. We can cite the entity-relationship [Che76] that is used to describe the data model for relational databases. In the web community, they use the ontology [Gru95] formalism to define the semantic web³ [BHL⁺01]. The software engineering community applies the Unified Modelling Language (UML) [OMG17] to formalism software system structure or behaviour.

Extending this need for abstraction to all software engineering activities, practitioners have proposed the MDE methodology [Sch06; Ken02]. This methodology advocates the use of models, or abstractions, as primary software artefacts [WHR]. The contributions proposed in this thesis are in line with this vision. In the following sections, we give an overview of the MDE methodology and how we will use it.

2.2.1 Principles and vision

Global overview Software systems tend to be more and more complex. To tame this complexity [FR07; Sch06], the MDE methodology suggests to use models for all the steps of software development and maintenance [Sch06; BCW17; HRW11; BLW05; HWR⁺11; AK03]: design, evolution, validation, etc. The core

²A metamodel is a model that defines another model (*cf.* Section 2.2).

³The Semantic Web is defined as an extension of the Web to enable the processing by machines.

idea of this approach is two reduce the gap between the problem and the solution space [Sch06]. Two main mechanisms have been defined: Domain Specific Modelling Language (DSML) and model transformation. The former is based on the separation of concern principle. Each concern⁴ should be addressed with a specific language, which manipulates concepts, has a type system, and a semantics dedicated to this concern. These languages allow to create and manipulate models, specific for a domain. The latter enables engineers to generate automatically software artefacts, such as documentation, source code, or test cases. Using these mechanisms, stakeholders can focus on their problem keeping in mind the big picture. A well-known example is the Structured Query Language (SQL) [fSta16] Using this language, engineers can query a relational database, the data model being the model. They don't have to consider indexes (hidden behind the concept of, for example, primary keys) or all the mechanisms to persist and retrieve data from the disk.

Advantages and disadvantages Defenders of the MDE approach mainly highlight the benefits of abstraction in software engineering [Sch06; Ken02; BLW05]. First, using the same model, engineers can target different specific platforms. For example, the ThingML [HFM⁺16] language allow specifying the behaviour of the system through state machines. The same ThingML code can be deployed on different platform such as Arduino, Raspberry Pi. Second, thanks to the transformation engine, the productivity and efficiency of developers is improved. Third, models allow engineers to implement verification and validation techniques, like model checking [BK08], which will enhance the software quality. Finally, the models enable the separation of application and infrastructure code and the reusability of models.

However, the literature has also identified some drawbacks of the MDE approach [Ken02; BLW05; WHR⁺13; HRW11]. First, it requires a significant initial effort when the DSML needs to be defined. Second, current approaches do not allow the definition of very large models. This drawback can be mitigated with new methods such as NeoEMF [BGS⁺14; DSB⁺17], which enable the storage of large models, and MogwaËr [DSC16], a query engine for large models. Third, this

⁴The definition of concern is intentionally left undefined as it is domain-specific.

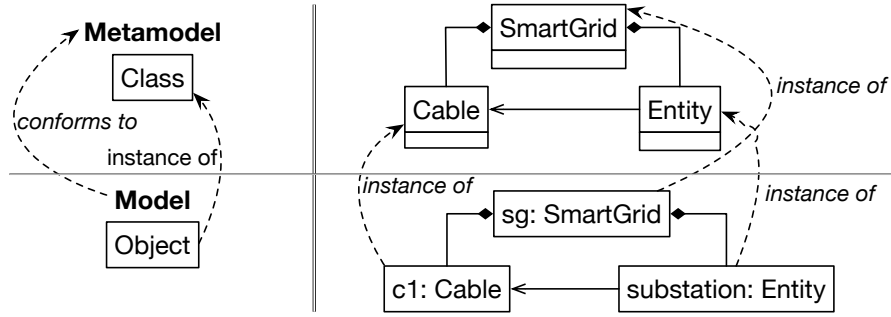


Figure 2.3: Difference and relation between metamodel and model

approach suffers from poor tooling support, as they should be reimplemented for each model. As for the second drawback, recent contributions try to remove this limitation. For example, we can cite the work of Bousse *et al.*, [BLC⁺18] that define a generic omniscient debugger⁵ for DSML. Third, introducing MDE in a software development team presents organisational challenges. It changes the way developers interact and work together. Finally, abstraction is a two edges sword. Indeed, reasoning at an abstract level may be more complex as some prefer to work with concrete examples and based on simulation.

Fundamentals concepts MDE is based on three fundamentals concepts: meta-model, model, and model transformation. In this thesis, we do not use any transformation technique. In the next section, we thus detail the concept of metamodel and model.

2.2.2 Metamodel, model

Metamodel Metamodels the different concepts in a domain and their relationships. They represent the knowledge of a domain, for a specific purpose [BJT05]. Also, they define the semantics rules and constraints to apply [Sch06]. They can be seen as models of models. In the MDE community, they are generally defined using the class diagram of the UML specification [OMG17], as shown in Figure 2.3. They, therefore, contain classes, named metaclass, and properties (attributes or references). In the language engineering domain, metamodels are used to define the

⁵Debuggers, generally, allow engineers to execute step-by-step programs, that is, forward. An omniscient debugger is also able to go backwards: to navigate back in the previous states of a program [Lew03].

2 concepts that a language can manipulate (*cf.* Section 2.3). Object Management
3 Group (OMG)⁶ define a standard metamodeling architecture: Meta Object Facility
4 (MOF) [Gro16a]. It is set aside with Object Constraint Language (OCL) [Gro14],
5 the standard constraint language to define constraints that cannot be specified
6 by a class diagram, and XML Metadata Interchange (XMI) [Gro15], the standard
7 data format to persist (meta)models.

8 **Model** Models capture some of the system characteristics into an abstraction
9 that can be understood, manipulated, or processed by engineers or another system.
10 They are linked to their metamodels through the conformance relationship as
11 depicted in Figure 2.3. A model is conformed to exactly one metamodel if and
12 only if it satisfies all the rules of the metamodel. That is, each element of a
13 model should instantiate at least one element of the metamodel and respect all
14 the semantics rules and constraints, that can be defined using OCL. Based on
15 these models, stakeholders can apply verification and validation techniques, such
16 as simulation and model checking, or model transformation. France *et al.*, have
17 identified two classes of models [FR07]: development and runtime models. The
18 former are abstraction above the code level. It regroups requirements, architectural,
19 or deployment models. The latter abstract runtime behaviour or status of systems.
20 They are the basis of the models@run.time paradigm, explained in Section 2.1.2.
21 The models@run.time paradigm uses metamodels to define the domain concepts
22 of a real system, together with its surrounding environment. Consequently, the
23 runtime model depicts an abstract and yet rich representation of the system context
24 that conforms to (is an instance of) its metamodel.

25 2.2.3 Tooling

26 Tooling is an essential aspect of every approach to be adopted. Development
27 platforms allow developers to create, manipulate, and persist (meta)models through
28 high or low-level programming interfaces. For example, a graphical language can
29 be used for this purpose. Additionally, these tools should embed transformation
30 engines such as a code generator.

1 In the MDE community, the standard tool is the Eclipse Modelling Frame-

⁶<https://www.omg.org/>

work (EMF) [SBM⁺08]. It is the defacto baseline framework to build modelling tools within the Eclipse ecosystem. It embeds its metamodeling language, ECore [SBM⁺08; Fou10]. ECore is thus the standard implementation of Essential MOF (EMOF) [Gro16a], a subset of MOF that corresponds to facilities found in object-oriented languages. As written on the EMF-website⁷, this modelling framework “provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor”.

However, as highlighted by Fouquet *et al.*, [FNM⁺14; FNM⁺12], models generated by EMF have some limitations, which prevent their use for the models@run.time paradigm. The models@run.time can be used to implement an adaptive Internet of Things (IoT) systems⁸. These systems contain small devices, like micro-controllers, that have limited memory, disk space, and process capacity. If one wants to deploy a model on it, thus it should have a low memory footprint, a low dependency size, a thread safety capacity, an efficient model (un)marshalling and cloning, a lazy loading mechanism, and compatibility with a standard design tool, here EMF. However, the approaches at this time failed to tame the first four requirements. They, therefore, define Kevoree Modelling Framework (KMF) [FNM⁺14; FNM⁺12], a modelling framework specific to the models@run.time paradigm.

Hartmann extended this work and created the GreyCat Modelling Environment (GCM)⁹. Using this environment, a developer can create high-scalable models, designed for models@run.time [HFN⁺14b; MHF⁺15], with time as a first-class concept. All metaclasses have a default time attribute to represent the lifetime of the model element that instantiates it. The created models are object graphs stored on a temporal graph database, called GreyCat¹⁰ [HFM⁺19; Har16]. In addition to time, metamodels defined by GCM can have an attribute with a value computed from a machine learning algorithm [HMF⁺19]. Based on the metamodel definition, a Java and a Javascript API are generated, to manipulate the model, *i.e.*, the

⁷<https://www.eclipse.org/modeling/emf/>

⁸Definition of IoT by Gubbi *et al.*, [GBM⁺13]: “Interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications.”

⁹<https://github.com/datathings/greycat/tree/master/modeling>

¹⁰<https://greycat.ai/>

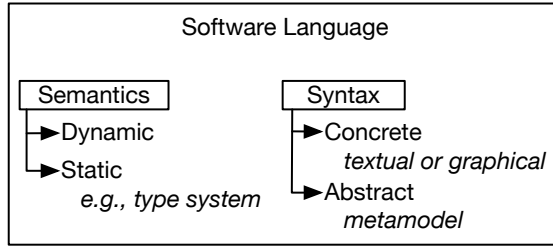


Figure 2.4: Composition of a software language

2 temporal object graph.

3 2.2.4 Concepts used in this thesis

4 The models@run.time paradigm is a well-known approach to handle the chal-
5 lenges faced in adaptive system development. Plus, the GCM has been designed to
6 design a data model, with time as the first-class concept. In this thesis, we will use
7 this tool to define a metamodel to abstract the knowledge of adaptive systems (cf.
8 Chapter 7).

9 2.3 Software Language Engineering

10 As stated by Favre *et al.*, [FGL⁺10], software languages are software. Like
11 traditional software, they need to be designed, tested, deployed, and maintained.
12 These activities are grouped under the term Software Language Engineering (SLE)
13 [Kle08]. Before explaining the role of software languages in this thesis, we will first
14 define them in this section.

15 2.3.1 Software Languages

16 Most of, not to say all of, developers have used, at least once, a programming
17 language to develop software. For example, one may use JavaScript to implement
18 client-side behaviour of a web site and another one C to implement a driver. We
19 can distinguish another kind of language, named modelling languages. Those
20 models allow developers to implement a model. For example, we can argue that
1 many developers have already used the HTML language to implement a Document

Object Model (DOM)¹¹ With the emergence of executable models¹², the difference between models and programs are more and more blurry. So it is for the difference between programming and modelling language. Hence, Annake Kleppe uses the term software language to combine both kinds of languages.

Another way to classify software languages is by their scope [vDKV00]. General Purpose Language (GPL) are languages that can be used for any domain whereas Domain Specific Language (DSL)¹³ that are restricted to a specific domain. Using a GPL, a developer can use it to implement a full software benefit from great tooling support. However, she may manipulate concepts that are different from those of the problem space. For example, implementing an automatic coffee machine in Java, she will have to manipulate the concept of class, object, functions. In contrary, DSLs are close to their problem domain [vDK98] but might suffer from poor tooling support [Voe14]. As for MDE, there are some research efforts to remove this disadvantage [BLC⁺18]. Using DSLs, developers can have simpler code, easier to understand and maintain [vDKV00; vDK98].

As depicted in Figure 2.4, Software languages are composed of two parts [HR04]: syntax and semantics. The syntax defines the element allowed in the language and the semantics their meaning. We can distinguish two kinds of syntax: the abstract one and the concrete one. The abstract one defines the different concepts manipulated with the language and their relationships. A metamodel can express it. The concrete abstract describes how these concepts are represented. It exists two categories of concrete syntax: graphical and textual. As for the semantics, there are two kinds of semantics: the static and the dynamic. The static semantics defines the constraints of the abstract syntax that cannot be directly expressed in the formalism chosen. For example, the static semantics will define uniqueness constraints for some elements or will forbid any cycle in dependencies (if any). The type system usually goes in part of the static semantics. The dynamic semantics defines the behaviour of the language.

When designing a language, engineers can start with the abstract or with the concrete abstract. In the first case, we say that they define the language in a

¹¹“The DOM is a platform -and language- neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of [web] documents.” [W3C05]

¹²Executable models are models that have a semantics attached to their concepts.

¹³In this thesis, we do not make the difference between DSML and DSL

2 model-first way. Others will prefer to start by the concrete syntax, and thus they
3 are using a grammar-first approach. Some tools, like XText [EB10], can generate
4 the model automatically from the grammar.

5 **2.3.2 SLE in this thesis**

6 In our vision, we argue that uncertainty should be considered as a first-class
7 citizen for modelling frameworks. This modelling framework should allow the
8 definition of metamodels with uncertainty management capacities. As described in
9 the previous sections, these metamodels can correspond to the abstract syntax of a
10 language. As a contribution, we, therefore, propose a language with uncertainty as
11 a first-class citizen: Ain'tea(*cf.* Chapter 6).

12 **2.4 Probability theory**

13 Data uncertainty, and more generally uncertainty, is linked with confidence.
14 Indeed, the confidence level that one can give to data depends on its uncertainty.
15 The more uncertain value is, the less trust it can be placed in it. However, it is
16 rather difficult to put exact numbers on this confidence level. A strategy used
17 by experts to formalise this confidence level, and thus the uncertainty, is to use
18 probability distributions.

19 In this section, we thus introduce the necessary concepts of these distributions.
20 We first describe the concept of random variables in probability theory. Then we
21 describe what a probability distribution is and the properties that will be used in
22 this thesis. Finally, we will explain the distributions used in this thesis.

23 **2.4.1 Random variables**

24 **Definition** The three base elements of the probability theory are: an outcome, an
25 event, and a sample space. An outcome is the occurrence of a random phenomenon.
26 An event is a set of outcomes, and a sample space S is the set of all possible
27 outcomes. Let us take an example: the rolling of two dice. i denotes the result of
28 one die and j the result of the other. An example of an outcome is the result of a
29 roll, like the pair $(2, 6)$. An event could contain all outcomes where both dice have
30 an even result: $E = \{(i, j) \mid i, j \in \{2, 4, 6\}\}$. The sample space of our example is
1 this equals to: $S_1 = \{(i, j) \mid i, j \in \{1, 2, \dots, 6\}\}$. We thus have:

A random variable is a function from the sample space S to \mathbb{R} . In our example, we can define two random variables X and Y that represent, respectively, the minimum of the two dice and the sum of the two. We have thus: $X = \min(i, j)$ and $Y = i + j$.

Discrete and continuous Random variable can be either discrete or continuous. For discrete random variables, we can list all the events of the sample space, whereas we cannot for continuous ones. For example, it is possible to list all results of the rolling of n dice, $n \in \mathbb{N}$. But we cannot list all possible temperatures of a room (if we consider that we have an infinite precision).

Independence and disjoint Independence and joint are defined at the event level. Two events are independent if the probability of occurrence of one does not impact the probability of occurrence of the others. Two events are disjoint if and only if they do not share any outcome. Mathematically speaking, events A and B are independent if and only if $P(A \cap B) = P(A) * P(B)$, and they are disjoint if and only if $P(A \cap B) = 0$.

2.4.2 Distribution

A probability distribution of a random variable X is a function that gives the probability that X takes on the values x , for all possible values of the sample space. Here, we can distinguish two cases: discrete and continuous distributions. A distribution is said discrete when it is based on a discrete random variable, and continuous when the random variable is continuous.

The difference that will interest us in this thesis between the two is how they compute the probability, and thus the confidence level. For discrete distributions, the probability that $X = x$ corresponds to the result of the function. By definition, the probability that the random variable is inferior to a value x is thus equal to the sum of probability for $X < x$.

Contrary, for continuous distribution, the probability is mapped to the area under the function that defines the probability distribution, *i.e.*, the integral of the function. For example, the confidence that a given uncertain value is greater than zero is the surface under the function $f(x)$ for any $f(x) \mid f(x) > 0$. As the area under a precise point is null, the probability that a continuous random variable

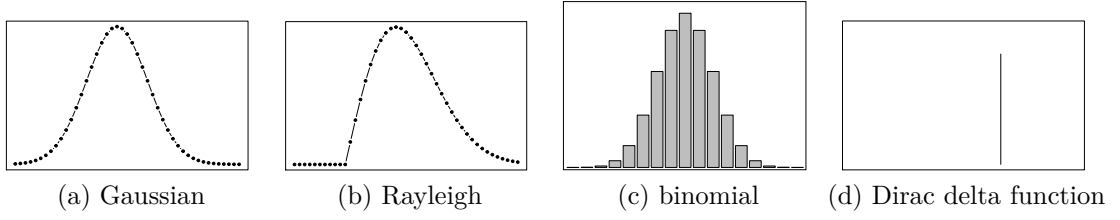


Figure 2.5: Probability distributions used in this thesis

2 equals x is zero.

3 2.4.3 Distribution used in this thesis

4 Probability distributions have proven their ability to represent uncertain data.
5 Among the existing distributions, in this thesis, we use on five of them: Gaussian
6 (or Normal) distribution, Rayleigh distribution, binomial distribution, Bernoulli
7 distribution and the Dirac delta function. We give an example of these distributions
8 in Figure 2.5. The Gaussian and the Rayleigh distributions will represent continuous
9 distributions. The Bernoulli and the binomial one represent discrete distributions.
10 Finally, the Dirac delta function can represent the confidence level of exactly one
11 value.

12 **Bernoulli distribution** Bernoulli distribution represents the distribution of a
13 binary phenomenon. One well-known example is the flip of a fair coin. Bernoulli(p)
14 denotes a random variable that equals 1 with a probability of p and 0 with a
15 probability of $(1 - p)$. Following the coin example, 1 can represent the head and 0
16 tail.

17 **Binomial distribution** The binomial distribution represents the probability of
18 success of a binary phenomenon over a set of trials. Figure 2.5c illustrates an
19 example of a Binomial distribution. It is defined by two parameters: n and p . n
20 is the number of trials done and p the probability of success. By definition, it is
21 defined over a discrete domain, more specifically on the set of natural number \mathbb{N} .
22 In this case, the confidence level corresponds to the values of the function.

23 It has been proven that this distribution is similar to a Gaussian distribution
24 in certain cases [BHH05]. If the domain definition can be changed from discrete to
1 continuous, a binomial distribution can be approximated by a Gaussian distribution.

Gaussian (or Normal) distribution The Gaussian distribution, commonly referred to as normal distribution, is the most general probability distribution. For example, the International Bureau of Weights and Measures encourages the use of this distribution to quantify the uncertainty of measured values [Met08]. It is defined by two parameters: a mean and a variance. The distribution is defined on a continuous domain: $]-\infty; +\infty[$. An example of this distribution is illustrated in Figure 2.5a.

Rayleigh distribution Another distribution defined on a continuous domain $([0; +\infty[)$ is the Rayleigh distribution. It is often used for GPS positions [Bor13]. This distribution is defined using a unique parameter: a variance. We depict an example of this distribution in Figure 2.5b.

Dirac delta function The Dirac delta function is defined as a probability function with $f(x) = +\infty$ for $x = 0$ and $f(x) = 0$ for all the other points. To represent other values than zero, the following variable substitution can be used $x = x - a$, where. We call a the shifting value. By definition, the integral of this function on the whole domain definition is equal to 1. As it is considered as a continuous distribution, confidence is mapped to the integral of the function. By applying a coefficient, we can modify the value of this integral, *e.g.*, to have 0.8 as the integral. We define this probability distribution using two parameters a coefficient and a shifting value. An example of this probability distribution is shown in Figure 2.5d. Conventionally, the Dirac function is represented as a vertical line that stops at the coefficient. The figure shows a Dirac function with a coefficient of 0.8 and a shifting value of 3. As it is defined on a single value, this distribution can be used for any numbers. This distribution can be used to represent uncertainty that is due to human errors.

3 Motivating example: smart grid

4 Contents

5	3.1 Smart grid overview	36
6		
7	3.2 Data uncertainty	38
8	3.3 Long-term actions	43

12 *In this chapter, we present our motivating example, based on the Luxembourg*
13 *Smart Grid. This example has been built in collaboration with our partner, Creos*
14 *S.A., the Luxembourgish grid manager. We first several examples of long-term*
15 *action and then we motivate the need for a language with uncertainty as a first-class*
1 *citizen with code samples.*

2 In this chapter, we exemplify the impacts of uncertain data and long-term
3 actions on a smart grid system. We built these example with Creos Luxembourg,
4 our partner¹. We first give an overview of a smart grid. Then, we focus on cable
5 load estimation and the impacts of not taking uncertainty into consideration. Then,
6 we concentrate on the reconfiguration feature of smart grids and the effect of
7 long-term action. We also extend this example with other examples from other
8 domain such as cloud infrastructure.

9 3.1 Smart grid overview

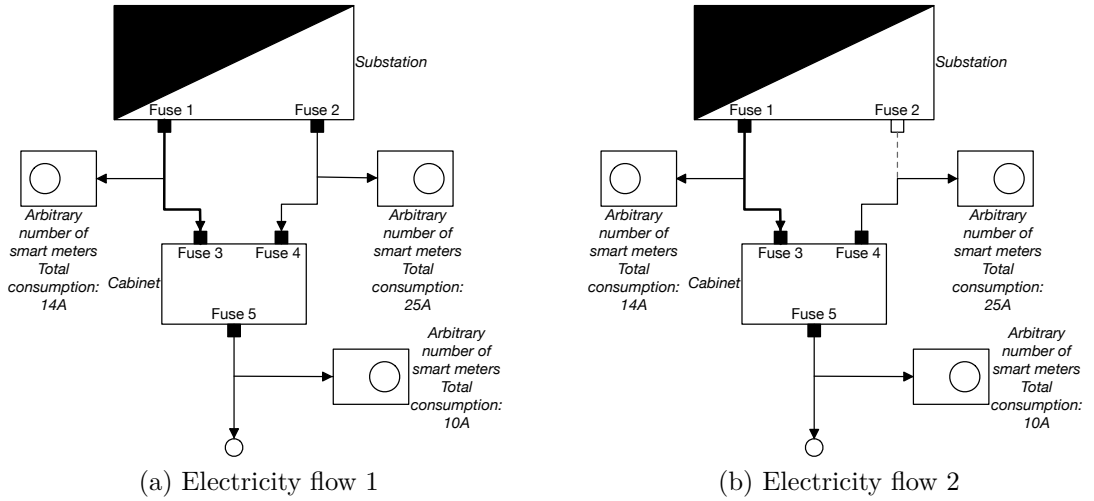


Figure 3.1: Two different electric flows for a same grid topology

10 The National Institute of Standards and Technology (NIST) defines a smart
11 grid as “a planned nationwide network that uses information technology to deliver
12 electricity efficiently, reliably, and securely”². Conceptually, a smart grid is com-
13 posed of different entities, like smart meters, cabinets (connection points of cables)
14 and power substations. These entities are connected, forming a network, and able
15 to exchange information using different technologies [HFK⁺14b; HMF⁺16].

16 The network is the connection linking the smart grid entities by means of
1 physical cables. Every cable has a maximum load depending on its diameter and

¹Creos Luxembourg is the main grid operator in the country

²<https://www.nist.gov/engineering-laboratory/smart-grid/smart-grid-beginners-guide>

the used material. Figure 3.1 depicts an example of such a network. This network is composed of one substation, one cabinet, and an arbitrary number of smart meters. Every cable has two fuses (to connect or disconnect the cable), one at each endpoint. By opening or closing fuses, one can influence the electricity flow through the network. Figure 3.1a and 3.1b show two possible electricity flows over the same physical network. We depict closed fuses in black and the direction of the electricity flow using black arrows.

Smart meters continuously measure electricity consumption and periodically report it to a central data centre. Based on this information, together with the grid topology, the electric load of cables can be computed. If the reader wants to go more into the details of this computation, we invite him or her to read [HMF⁺16]. The flow has a big impact on the load cables. In Figure 3.1, we depict two different flows for the same grid topology. In both cases, the measured electric consumption are equal. However, in Figure 3.1a the load on the left cable (thick line) equals $\frac{14+25+10}{2} = 24.5$ whereas it equals $14 + 25 + 10 = 49$ in Figure 3.1b.

The central system monitors electric loads to avoid any overload in the network, implementing the adaptive system vision. In the remaining part of this document, we refer to this goal as the "*no overload*" policy. They have two action points: either on the production side or the consumption side. They can reduce or increase the production by (dis)connecting production unit or the consumption by modifying the maximum permitted consumption. We called these actions: *reduce production*, *increase production*, *reduce amps limit* and *increase amps limit*. However, as all adaptive systems, smart grids are prone to failures [HFK⁺14a]. In case of a failure, an engineer could diagnose the system, and determine the adaptation process responsible for this failure. For instance, considering some reports about regular power cuts during the last couple of days, in a particular area, a stakeholder may want to interrogate the system and determine what past decision(s) have led to this suboptimal state. More concretely, he will ask: did the system make any decisions that could have impacted the customer consumption? If so, what goal(s) the system was trying to reach and what were the values used at the time the decision(s) was(were) made?

2 3.2 Data uncertainty

3 3.2.1 Impacts of ignoring data uncertainty

4 Although power grids are becoming more and more automated, today human
5 interventions are still the norm. For example, most fuses are manually modified by
6 technicians rather than automatically reconfigured. The states of fuses are thus
7 manually documented by technicians on the field. This of course results in mistakes.
8 One way to address this problem is to take uncertainty into consideration. This
9 means considering fuse states as uncertain.

10 As a consequence, this uncertainty propagates to the load calculation formulas,
11 which depend on the fuse states. If the uncertainty of fuse states is not considered,
12 it exists a non-null probability that the observed phenomenon does not reflect
13 the real situation. For example, as we have seen in the previous subsection, a
14 modification of the electricity flow may impact the load of a cable with a factor of
15 2. Cable load approximations are used to detect cable overloads and to reconfigure
16 the network if necessary. By not considering uncertainty, wrong reconfigurations
17 might be triggered, which could worsen the situation.

18 3.2.2 Managing uncertainty is not effortless

19 In the following, we describe how uncertainty is commonly handled by appli-
20 cation developers using current state-of-the-art approaches. We show, through
21 code samples, the limitations of these approaches and why we think that these
22 limitations can be addressed by integrating uncertainty management directly at
23 the language level. In the code excerpts below, we compute the average cable load
24 over the whole grid based on uncertain cable loads. A complete version can be
25 found on the GitHub repository. All codes contain at least two classes: *SmartGrid*
26 and *Cable*. The former contains two fields: an array of Cables named *cables* and a
27 function to compute the average load of cables, named *compute_avg_load*. The
28 latter contains one field: an uncertain number which represents the uncertain load.

29 **Manual implementation** One approach for managing uncertainty is to manu-
30 ally implement all the required features. Listing 3.1 illustrates this in Python. In
1 addition to the *SmartGrid* and *Cable* class, the excerpt contains an additional one:

2 *UNumber*.

3 As we can see, this approach comes with several drawbacks. First, developers
4 are required to have deep knowledge of probability theory. For example, uncertainty
5 can be represented by a normal distribution identified by its mean and variance
6 as shown in the constructor (`__init__`), line 2. One needs knowledge on how
7 to represent it, how to add two normal distributions, etc. For example, here we
8 overload the sum and the division operators for two normal distributions (`__plus__`
9 and `__div__` methods).

10 Second, manual implementation inevitably increases the size of the code base.
11 This will de facto augment the risk of errors in the code. Plus, the code will be
12 more difficult to maintain afterwards.

13 Third, although some languages offer the possibility to overload operators,
14 some typing errors can only be detected at runtime. For instance, since Python
15 is dynamically typed, performing an addition operation between two types of
16 uncertain numbers (*i.e.*, represented by two classes) fails and raises an exception
17 only at runtime. Whilst, statically typed languages such as C# can detect such
18 typing errors at development time. Nonetheless, the returned exception message
19 would not be particularly meaningful, as can be seen on the example of C#:
20 *Operator '+' cannot be applied to operands of type UNumber1 and UNumber2.*

```
21 class UNumber:
22     def __init__(self, mean=0, variance=0):
23         [...]
24
25     def __add__(self, other):
26         [...] # typing error management + casting other to UNumber if it is a Number
27         return UNumber(self.mean + other.mean, self.variance + other.variance)
28
29     def __div__(self, other):
30         [...] # typing error management + casting other to UNumber if it is a Number
31         value = ((self.mean/other.mean)
32                 + (self.mean*other.variance)) / pow(other.mean, 3)
33         variance = (self.variance / other.mean) +
34                     (pow(self.mean, 2) * other.variance) / pow(other.mean, 4)
35         return UNumber(value, variance)
36
37
38 class SmartGrid:
39     [...]
40     def compute_avg_load(self):
41         sum_load = 0
42         for c in self.cables:
```

2	sum_load += c.load	22
3	return sum_load / len(self.cables)	23
4		24
5	class Cable:	25
6	def __init__(self, id, load=UNumber(0, 0)):	26
7	[...]	27

Listing 3.1: Manual management of uncertainty in Python

9 **Using existing libraries for probability theory** Since uncertainty manage-
10 ment relies on probability theory, another approach is using a library implementing
11 common probability distributions. Staying in Python, there is a widely used library
12 called SciPy³ and in particular the *stats* module, defining different probability
13 distributions. Whereas these libraries are suitable to compute different properties
14 of probability distributions, it remains the responsibility of developers to define
15 how to perform arithmetic operations on distributions, as depicted in Listing 3.2.

16 Using such libraries does not prevent developers to have a deep understanding
17 of the probability theory. They have been designed to help them manipulating
18 probability distributions, but they still required knowledge about them. For
19 example, as shown in Listing 3.2, one may access the mean and the variance
20 of a normal distribution. However, she or he needs to know how two normal
21 distributions can be added. This also increases the code base and thus is prone to
22 errors. Finally, as it is a library, typing errors will still remain either detected at
23 runtime or not helpful for developers.

24		1
25	from scipy.stats import norm	2
26		3
27	def add_gaussian(g1, g2):	4
28	[...]	5
29	return norm(g1.mean() + g2.mean(), g1.var() + g2.var())	6
30		7
31	def div_gaussian(g1, g2):	8
32	[...]	9
33		10
34	class SmartGrid:	11
35	[...]	12
36	def compute_avg_load(self):	13
37	sum_load = 0	14
38	for c in self.cables:	15
1	sum_load = add_gaussian(sum_load, c.load)	

³<https://www.scipy.org/>

```

2         return div_gaussian(sum_load, len(self.cables))
3
4 class Cable:
5     def __init__(self, id, load=norm(0, 0)):
6         [...]

```

Listing 3.2: Limitation using a probability library (Python)

Using existing libraries for uncertainty Another commonly used approach is to use existing libraries for data uncertainty. For example, in Python, using the *uncertainties* [LEB18] library, a developer can manipulate uncertain floats using variables of type *ufloat*. The uncertainty propagation is handled transparently. In Listing 3.3, we depict the code of the load average computation using this library. The only difference from the previous code snippet, is the missing of the *UNumber* class, not required anymore, and the use of the *ufloat* type for the cable load variable.

Using a library can tackle the two first drawbacks existing in the previous approach: required knowledge of probability theory and code complexity. Indeed, the implementation of the probability theory and its complexity are encapsulated inside the library. We can argue that the library will be well tested and documented. However, similarly to the previous example, this approach does not extend the type system.

```

23 from uncertainties import ufloat
24
25 class SmartGrid:
26     def __init__(self):
27         self.cables = []
28
29     def compute_avg_load(self):
30         [...] # same as in Listing 3.1
31
32 class Cable:
33     def __init__(self, id, load=ufloat(0, 0)):
34         [...]

```

Listing 3.3: Managing uncertainty in Python using uncertainties [LEB18]

Using a probability programming framework Developers can also use a probabilistic programming framework like Infer.NET [MWG⁺18] or PyMC3 [SWF16]. These frameworks allow defining complex probabilistic models and applying infer-

2 ence algorithms. Contrary to previously described libraries, these frameworks put
 3 an effort to set a proper API. They can be thought as internal domain-specific
 4 languages (DSL)[Fow10]⁴. In Listing 3.4, we show how the cable load can be
 5 computed using the Infer.NET framework. This approach exposes probability
 6 as a first-class language citizen. Hence, developers are expected to manipulate
 7 probability distributions and not uncertain data types. This framework relies
 8 on an inference engine to transparently combine probability distributions. Using
 9 such an approach requires a decent acquaintance with probability theory. We can
 10 assume that they are heavily tested, which limits the number of errors. Plus, in
 11 this framework they map arithmetic operators to the combination of probability
 12 distributions: developers do not require to create additional work as with the
 13 library approach. But, as it is not implemented at the language level, the typing
 14 system is not extended. Errors will thus either be raised at runtime or not be
 15 helpful for developers.

```

16 public class SmartGit {
17     public List<Cable> cables { get; private set; }
18     private readonly InferenceEngine inference;
19
20
21     public Gaussian computeAvgLoad() {
22         Variable<double> sum = Variable
23             .GaussianFromMeanAndVariance(0,0)
24             .Named("sum");
25
26         int i = 0;
27         foreach(Cable c in cables)
28             sum = (sum + c.load).Named("sum" + i);
29             i++;
30         Variable<double> result = (sum / cables.Capacity).Named("AvgLoad");
31         return (Gaussian) this.inference.Infer(result);
32     }
33 }
34
35 public class Cable {
36     public Variable<double> load { get; set; }
37 }
  
```

Listing 3.4: Limitation using a probability programming framework (C#)

⁴Martin Fowler defines a DSL as “a computer programming language of limited expressiveness focused on a particular domain”[Fow10]. An internal DSL is defined inside a general-purpose language, using only a subset of its concept.

Sum-up Using state-of-art solutions to manage uncertainty, developers have three possibilities: manual implementation, using existing libraries or using a probability programming framework. As detailed in this section, these approaches will cope with at least one of these drawbacks: developers require a deep understanding of the probability theory, code base will increase which augments the risk of errors, and the type systems may detect errors at runtime or do not provide helpful messages.

3.3 Long-term action

3.3.1 Examples

Long-term actions

In this section, we motivate the need to reason over long-term actions. We first give four examples of these actions. Then, we detail why the effects of actions should be considered. Finally, we summarise and motivate the need for incorporating actions and their effects on the knowledge.

Long-term action examples In the previous sections, we have claimed that adaptation processes should handle long-term actions. In order to show their existence, we give four different examples: two based on our use case, one on cloud infrastructure and a last one on smart homes. From our understanding, three phenomena can explain this delay: the time to execute an action(s) (Example 1), the time for the system to handle the new configuration (Example 3) and the inertia of the measured element (Example 2 and 4).

Example 1: Modification of fuse states in smart grids Even if the Luxembourg power grid is moving to an autonomous one, not all the elements can be remotely controlled. One example is the fuses that still need to be opened or closed by a human operator. Open and close actions in the Luxembourg smart grid both imply technicians who are contacted, drive to fuse locations and manually change their states. If several fuses need to be changed due to one decision, only one technician will drive to them, sequentially, and execute the modifications. For example, our industrial partner asks us to consider that each fuse modification takes 15 min whereas any incident should be detected in the minute. Let's imagine that an incident is detected at 4 p.m. and can be solved by modifying three fuses.

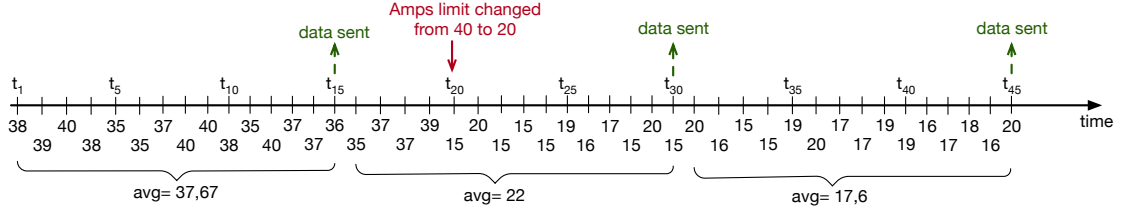


Figure 3.2: Example of consumption measurement before and after a limitation of amps has been executed at t_{20} .

2 The incidents will be seen as resolved by the adaptation process at 4 p.m. + 15
3 min * 3 = 4:45 p.m. In this case, the delay of the action is due to the execution
4 time that is not immediate.

5 **Example 2: Reduction of amps limit in smart grids**⁵ Creos S.A. en-
6 visages controlling remotely amps limits of customers in its smart grid project.
7 Customers will have two limits: a fixed one, set at the beginning, and a flexible one,
8 remotely managed. The action to remotely change amps limits will be performed
9 through specific plugs, such as one for electric vehicles. Even if the action is near
10 instant, due to how power consumption is collected, its impacts would not be
11 visible immediately. Indeed, data received by Creos S.A. corresponds to the total
12 energy consumed since the installation. From this information, only the average of
13 consumed data for the last period can be computed.

14 In Figure 3.2, we depict a scenario that shows the delay between the action is
15 executed and the impacts are measured. Each time point represents one minute,
16 with the consumption at this moment.

17 Let us imagine a customer who has his or her limit set to 40 amps⁶ and consumes
18 near this limit. We consider that data is sent every 15 min. After receiving data
19 sent t_{15} and processing them, the adaptation process detects an overload and
20 decides to reduce the limits to 20 amps for the customer. However, considering
21 the delay for data to be collected and the one to send data⁷, the action is received
1 and executed at t_{20} . At t_{30} , new consumption data is sent, and equals 22 amps.

⁵This example is based on randomly generated data. As this action is not yet available on the Luxembourg smart grid, we miss real data. However, it reflects an hypothesis shared with our partner.

⁶The user cannot consume more than 40 amps at a precise time t_i .

⁷The smart grid is not built upon a fast network such a fiber network.

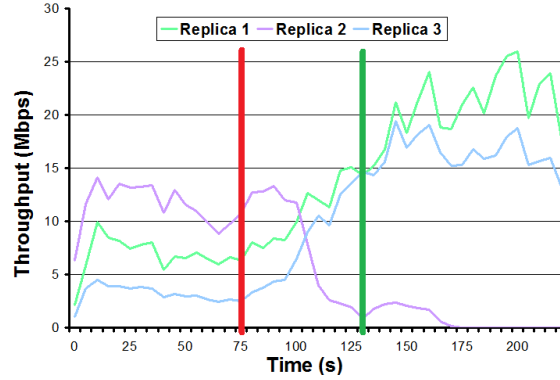


Figure 3.3: Figure extracted from [WBR11]. The red bar depicted the moment when Replica 2 stop receiving new connections. The green one represents the moment where all the rules in the load balancer stop considering R2. Despite these two actions, the throughput of the machine does not drop to 0 due to existing and active connections.

Two situations are now possibles. First, this reduction was enough to fix the overload. Even in this idealistic scenario, the adaptation process must wait in the worst case for 15 min ($t_{30} - t_{15}$) to observe the resolution (without considering the communication time). Second, this reduction was not enough - as the adaptation process considered that the consumption data will be at worst 20 amps and here it is 22. Before observing the incident as solved and knowing that the decision fixed the incident, the adaptation process needs to wait for new data, sent at t_{45} , *i.e.*, around 30 min ($t_{45} - t_{15}$) after the detection. In this case, the delay of this action can be explained by the inertia in the average of the consumption.

Example 3: Switching off a machine from a load balancer An example based on cloud infrastructure of long-term actions is to remove a machine from a load balancer, for example during a scale down operation. Scale down operations allows cloud managers to reduce allocated resources for a specific task. It is used either to reduce the cost of the infrastructure or to reallocate them to other tasks. In [WBR11], Wang *et al.*, present a load-balancing algorithm. In their evaluation, they present the figure depicted in Figure 3.3 that shows the evolution of the throughput after the server Replica 2 (R2) is removing from the load balancer. The red bar shows the moment where R2 stop receiving new connection and the green the moment where it is removed from the load balancer algorithm. However,

2 despite these actions have been taken, R2 should finish the ongoing tasks that it is
3 executing. This explains why the throughout is progressively decreasing to 0 and
4 there is a delay of around 100s between the red bars and the moment where R2
5 stop being active.

6 This example shows a long-term action due to the time required by the system
7 to handle the new configuration.

8 **Example 4: Modifying home temperature through a smart home**
9 **system** Smart home systems have been implemented in order to manage remotely
10 a house or to perform automatically routines. For example, it allows users to close
11 or open blinds from their smartphones. Based on instruction temperatures, smart
12 home systems manage the heating or cooling system to reach them at the desired
13 time. However, heating or cooling a house is not immediate, it can take several
14 hours before the targeted temperature is reached. Plus, if the temperature sensor
15 and the heating or cooling system are not placed nearby, the new temperature can
16 take time before being measured. This can be explained due to the temperature
17 inertia plus the delay for the temperature to be propagated.

18 Through these four examples, we show that long-term actions can be found in
19 different kinds of systems, from CPS to cloud infrastructure. However, not only
20 knowing that an action is running is important but also knowing its expected effect.
21 We detail this point in the following section.

22 **The need to consider effects** One may argue that action statuses are already
23 integrated into the knowledge. For example, the OpenStack Watcher framework
24 stores them in a database⁸, accessible through an API. However, for the best of
25 our knowledge, Watcher does not store the expecting effects of each action. While
26 the adaptation process knows what action is running, it does not know what to
27 expect from them.

28 Considering our example based on the modification of fuses, if the system
29 knows that the technician is modifying fuse states, it does not know what would
30 be the effects. In this case, when the adaptation process analyses the system
1 context it may wonder: what will be the next grid configuration? How will the

⁸<https://docs.openstack.org/watcher/latest/glossary.html#watcher-database-definition>

load be balanced? Will the future configuration fix all the current incidents? If the effects are not considered by the adaptation process, then it may take suboptimal decisions.

Let's exemplify this claim through a scenario based on the fuse example (*cf.* Example 1). As explained before, the overload detected at 4 p.m. takes around 45 min to be fixed. The system marks this incident as "being resolved". In addition to this information, the knowledge contains another one saying that it is being solved by modifying three fuses. However, during the resolution stage, a cable is also being overloaded. The adaptation process has two solutions. It can either wait for the end of the resolution of the first incident to see if both overloaded elements will be fixed or it takes other actions without considering the ongoing actions and their impacts. Applying the first strategy may make the resolution of the second incident late, whereas the second one may generate a suboptimal sequence of actions. For example, the second modifications may undo what has been done before or both actions may be conflicting.

Sum-up Actions, like fuse modification in a smart grid or removing a server from a load balancer, generated during by adaptation processes could take time upon completion. Moreover, the expected effects resulting from such action is reflected in the context representation only after a certain delay. One used workaround is the selection, often empirically, of an optimistic time interval between two iterations of the MAPE-K loop such that this interval is bigger than the longest action execution time. However, the time to execute an action is highly influenced by system overload or failures, making such empirical tuning barely reliable. We argue that by enriching context representation with support for past and future planned actions and their expected effects over time, we can highly enhance reasoning processes and avoid empirical tuning.

Fined and rich context information directly influences the accuracy of the actions taken. Various techniques to represent context information have been proposed; among which we find the models@run.time [MBJ⁺09; BBF09]. The models@run.time paradigm inherits model-driven engineering concepts to extend the use of models not only at design time but also at runtime. This model-based representation has

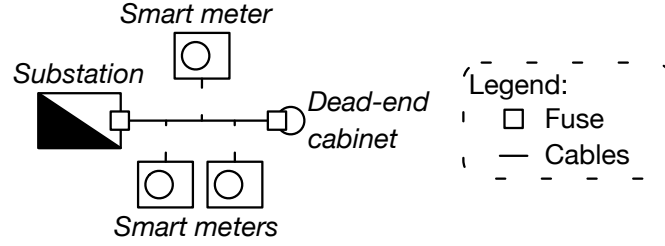


Figure 3.4: Simplified version of a smart grid

2 proven its ability to structure complex systems and synthesise its internal state as
 3 well as its surrounding environment.

4 3.3.2 Use case scenario

5 **Excerpt of a smart grid** Figure 3.4 shows a simplified version of a smart grid
 6 with one substation, one cable, three smart meters and one dead-end cabinet. Both
 7 the substation and the cabinet have one fuse each. The meters regularly send
 8 consumption data at the same timestamp. For this example, we consider one
 9 requirement: minimizing the number of overloads. To achieve this, two actions are
 10 taken into account in this example: decreasing or increasing the amps limits of
 11 smart meters.

12 **Adaptation scenario** The system starts at t_0 with the actions, the requirements
 13 and all element of the context that remain fixed: the grid installation. Meters
 14 send their values at t_1 , t_2 and t_3 . Based on these data, the load on cables and
 15 substation is computed. On t_1 , an overload is detected on the cable, which breaks
 16 the requirement. At the same time point, the system decides to reduce the load of
 17 all smart meters. The impact of these actions will be measured at t_2 and t_3 , *i.e.*,
 18 the consumption will slowly reduce until the cable is no longer overloaded from t_3 .

19 **Diagnosis scenario** As all adaptive systems, smart grids are prone to
 20 failures [HFK⁺14a]. Using our approach, an engineer could diagnose the sys-
 21 tem, and determine the adaptation process responsible for this failure. For instance,
 22 considering some reports about regular power cuts during the last couple of days, in
 23 a particular area, a stakeholder may want to interrogate the system and determine
 1 what past decision(s) have led to this suboptimal state. More concretely, he will

2 ask: did the system make any decisions that could have impacted the customer
3 consumption? If so, what goal(s) the system was trying to reach and what were
1 the values used at the time the decision(s) was(were) made?

3 State of the art

4 Contents

5	4.1	Review methodology	52
6	4.2	Results RQ1: long-term actions	54
7	4.3	Results RQ2: data uncertainty	67
8	4.4	Threat to validity	75
9	4.5	Conclusion	76

10
11
12
13

14 *This chapter reviews works related to the one presented in this dissertation.*
 15 *Two research questions drove this review. The first one aims at investigating state-*
 16 *of-the-art solutions that model adaptive systems to see those that consider long-term*
 17 *action. With the second research question, we search current approaches that handle*
 18 *data uncertainty. Our review shows that none of the studies performed until now*
 19 *take into account long-term action. Additionally, it indicates that different solutions*
 20 *exist to model data uncertainty, but some efforts need to be performed to provide*
 1 *solutions at a higher level.*

2 4.1 Review methodology

3 This review aims at answering two global research questions. To help us answer
4 them, we split them into three sub-research questions.

5 **Research questions** The first research question has been set to study the
6 presence of long-term actions in the modelling layer: (**RQ1**) do state-of-the-art
7 solutions that model adaptive systems allow representing and reasoning over long-
8 term actions (design time and runtime)? We split it into the following sub-questions:

- 9 • **RQ1.1:** How current approaches model the evolution of the context or the
10 evolution of systems (structure or behaviour) over time?
- 11 • **RQ1.2:** What are the solutions that model actions, their circumstances
12 and their effects over time at design time and runtime? Can this model be
13 processed or navigated automatically?
- 14 • **RQ1.3:** What are the solutions that enable the reasoning over the evolving
15 context, structure, or behaviour of systems? Do they also enable reasoning
16 over running actions and their effects?

17 With RQ1.1, we investigated the approaches that model the context, structure,
18 or behaviour of adaptive systems. The second one filters those that also consider
19 long-term actions. Finally, we use RQ1.3 to list solutions that provide a technique,
20 such as an algorithm, to reason over evolving context, structure, or behaviour.

21 With the second research question, we seek for modelling solutions that consider
22 uncertain data and its propagation: (**RQ2**) do state-of-the-art solutions allow
23 modelling uncertainty of data and its manipulation (propagation, reasoning over)?
24 The three sub-questions are:

- 25 • **RQ2.1:** What are the categories of uncertainties that have been addressed
26 by the literature?
- 27 • **RQ2.2:** How the uncertainty of data is modelled?
- 28 • **RQ2.3:** What are the solutions that enable an imperceptible propagation
29 and reasoning over uncertainty?

30 We set RQ2.1 to identify the different kinds of uncertainties that bring challenges in
31 software engineering. Then, we search for the technique to model uncertainty, with
32 a particular interest in data uncertainty with RQ2.2. Lastly, we review approaches
1 that allow developers to propagate uncertainty without writing specific code for

that and to reason over uncertainty using RQ2.3.

Methodology To review the literature, we applied a technique inspired by the snowballing approach [Woh14]. But, due to limited resources, we did not fully implement it. The methodology advocates the use of bibliography (backward navigation) and papers that cite (forward navigation) the selected ones to navigate in the literature. Each article should be evaluated according to a set of inclusion and exclusion criteria. And a starting set should be defined.

In our case, we use the bibliography of the papers that ground in this thesis as the starting set (*cf.* Section 1.4). Then, we apply the backward navigation for a subset of them. Then, we select the paper to add in this review according to a set of inclusion and exclusion criteria. To be picked, a paper should satisfy all inclusion criteria and should not fulfil any of the exclusion criteria. These criteria are the following:

- Inclusion criteria (IC):
 - **IC1**: The paper has been published before August 9 2019.
 - **IC2**: The paper is available online and written in English.
 - **IC3**: The paper describes a modelling approach that abstract the context, the structure, or the behaviour of a system, an approach that enables to reason or navigate through a temporal model, an approach that describes a solution used to engineer a adaptive system, an approach that handles uncertainty, or an approach that helps to manipulate probability distributions.
- Exclusion criteria (EC):
 - **EC1**: The paper has at most four pages (short paper).
 - **EC2**: The paper presents a work in progress (workshop papers), a poster, a vision, a position, an exemplar, a data set, a tutorial, a project, or a Bachelor, Master or PhD dissertation.
 - **EC3**: The paper describes a secondary study (*e.g.*, literature reviews, lessons learned).
 - **EC4**: The document has not been published in a venue with a peer-review process. For example, technical and research report or white papers.

- 2 – **EC5**: The document is an introduction to the proceedings of a venue
3 or a special issue, or it is a guest paper.

4 The first two inclusion criteria are accessibility criteria: they guarantee that
5 the paper is accessible for any reader of this document. With the third one, IC3,
6 we can include all papers that can be used to answer our research questions. We
7 define the exclusion criteria to keep only papers that have been published in a
8 peer-reviewed venue, and that present an approach.

9 In this review, 412 papers have been processed, and 84 kept for the review. We
10 report our selection results in an Excel file publicly available on GitHub ¹. Results
11 have also been exported to Comma-separated values (CSV) files for those who
12 cannot open Excel files.

13 **4.2 Results RQ1: long-term actions**

14 In this section, we detail our findings regarding the first research questions.
15 First, we detail all approaches that propose a solution to model the evolution of a
16 system’s context, structure, and behaviour. Then, we list approaches that model
17 actions. Before summarising and answering the research question, we list solutions
18 that model and reason over evolving context, behaviour, and behaviour, *i.e.*, that
19 implement an adaptation process.

20 **4.2.1 Modelling the evolution of system’s context, struc-** 21 **ture, or behaviour**

22 Different categories of approaches exist to represent the context, structure, or
23 the behaviour of a system. In this section, we detail our findings with an overview
24 given in Table 4.1.

25 **Models@run.time-based approach** In the MDE community, researchers de-
26 fined the models@run.time paradigm to implement an adaptation process [BBF09;
27 MBJ⁺09]. This approach is based on a runtime model that reflects the current state
28 of the system. It can contain information about either the context of the system, its
29 behaviour, or its structure. Moreover, there is a causal link between the model and
1 the system: modifications of the model, made by a stakeholder or a process, trigger

¹<https://github.com/lmouline/thesis/tree/master/sota/src>

Approach	Reference	History?
Models@run.- time-based	[BBF09; MBJ ⁺ 09; HFN ⁺ 14b; HFN ⁺ 14a]	[HFN ⁺ 14b; HFN ⁺ 14a]: add a time dimension to the models- @run.time paradigm
Formal model	[WMA12; WHH10; BK11]	X
Object-based model	[HIR02; HFK ⁺ 14a; TOH17]	[HIR02; HFK ⁺ 14a]: use a model with <i>temporal relations</i>
Goal model	[CvL17; IW14; MAR14; CPY ⁺ 14; BPS10]	X
State machine	[HFK ⁺ 14a; IW14; ARS15; AGR11; BdMM ⁺ 17; BBG ⁺ 13; MCG ⁺ 15; FGL ⁺ 11; GS10; DMS18; DBZ14; ZGC09; GPS ⁺ 13; TGE ⁺ 10]	X
Sequential di- agram	[TOH17]	X
Component model	[DL06; DBZ14; GCH ⁺ 04; FMF ⁺ 12]	X
Trace model	[Mao09]	X
Graph model	[KM90; GvdHT09; MS17]	[MS17]: defines a temporal graph, with two functions to retrieve temporal information. [GvdHT09]: historical graph of configurations.

Table 4.1: Approaches to model systems' context and behaviour (RQ1.1)

modifications in the system. For example, changing the status of a fuse in a model that reflects a smart grid triggers the action to open or close it. Hartmann *et al.*, extended this paradigm to introduce a temporal dimension [HFN⁺14b; HFN⁺14a]. It allows designers to store not only the current state of the system but also its past (previous states) and future (predicted). In this thesis, we will use this extension of the paradigm to build our knowledge model, and more precisely to represent long-term actions.

Formal model In [WMA12], Weyns *et al.*, defined a formal model for adaptive systems, called FORMS. Their goal was to establish a reference model, which can be used for discussion or implementation. For self-organisation systems, the literature provides another formal model: MACODO [WHH10]. It uses the Z language [dL04] to formalise the context of the system following the set theory and the first order predicate calculus. The behaviour is formalised with what they call *laws*. A law is a function from one set to another. The third formal model found in our review was specified by Bartels and Kleine [BK11]. This one uses Communicating Sequential Process principles [Hoa78], a formalism designed for reactive and concurrent systems. However, none of these models includes a time dimension, relevant to abstract long-term action.

Object-based model One category of approach found in the literature is object-based models. These models follow object-oriented principles. First, Henricksen *et al.*, defined a model for pervasive computing systems². In their model, some entities are linked to their attributes through uni-directional association. These associations can be dynamic (can evolve) or static (do not change over time). A dynamic association can also be temporal. In this case, the entity can have several attributes with a timestamp attached. Second, Hartmann *et al.*, use a temporal model to store the context and its history of a smart grid system [HFK⁺14a]. This model is based on their extension of the models@run.time described above [HFN⁺14b; HFN⁺14a]. Third, Tahara *et al.*, [TOH17] use the Maude language [CDE⁺02], to represent the context. Among these three solutions, only the last one does not include a temporal dimension.

²A pervasive system is composed of cheap and interconnected devices that are ubiquitous and can support users' tasks.[HIR02]

Goal model Goal modelling is a technique used by several contributions in our findings [CvL17; IW14; MAR14; CPY⁺14; BPS10]. This technique, mainly used in requirement engineering, represents the different goals of an application. By modelling the requirements of a system, we can argue that they *de facto* represent the context by the goals that are achieved or not. Moreover, in [CvL17], the authors add a satisfactory rate on each goal. However, these methods do not include a time dimension.

State machine State machines have the capacity to abstract, in the same model, the context and the behaviour of a system. The different states represent the context while the transitions between the states abstract the behaviour. [HFK⁺14a; IW14; ARS15; AGR11; GPS⁺13] use the Final State Machine (FSM) formalism. For example, Hartmann *et al.*, abstracts the behaviour of a smart meter in [HFK⁺14a]. In [GPS⁺13], the authors represent the functionalities of the system and their impact with states. And, transitions abstract the different execution flow between the different functionalities. In [BdMM⁺17; BBG⁺13], authors apply the labelled transition system [Kel76]. When authors want to consider the stochastic behaviour of the system, then they use probabilistic state machines. In [BdMM⁺17], the authors extended the model with probabilities, which represent the probability for a transition to be executed. Another strategy is to use Markov Chain [MCG⁺15; FGL⁺11; GS10; DMS18]. A Markov chain can be thought as a FSM with probabilities attached to the transition. In [MCG⁺15], authors also add information regarding current actions being executed with their progress status. But, no history is kept, the information is lost when actions finish. Other approaches use state machine without specifying the formalism used [DBZ14; ZGC09; TGE⁺10]. Tajalli *et al.*, use two state machines: one to represent the system context and behaviour, and another one to represent the adaptation mechanism.

Sequential diagram Through our review, we find one approach that uses a sequential diagram to represent the behaviour of the system [TOH17]. However, nothing is mentioned regarding the context of the system and its history.

Component model In order to represent the context of a system, one can use a component model. This model is at the architecture level and described the different entities (component) that compose a system with their interactions.

2 Four contributions apply this technique in our review [DL06; DBZ14; GCH⁺04;
3 FMF⁺12]. To also represent the behaviour, some have extended this model with a
4 state machine description in [DBZ14] or with annotation in [GCH⁺04]. However,
5 no time dimension is considered in these approaches.

6 **Trace model** Context and behaviour of a system can be inferred by analysing
7 its logs. In [Mao09], researchers defined an approach to create a model that reflects
8 the runtime state of the system. However, this approach does not keep the history
9 of the system.

10 **Graph model** Finally, the last technique used to represent the context of a
11 system is to use a graph model [KM90; GvdHT09; MS17]. In the former, nodes
12 represent process units, and edges the communication between them. In the
13 second one, nodes represent the possible configurations of the system, and the
14 edges represent the actions to reach a configuration. The graph used in [GvdHT09]
15 represents the different configuration over time of the system. Plus, some meta-data
16 about previous adaptations (*e.g.*, average time in this configuration) are added.
17 The latter defined a temporal graph [MS17]. Their temporal graph is a graph
18 that is augmented with two functions. One function serves to indicate if a graph
19 element, a node or an edge, exists for a given period. The other can retrieve the
20 value of a graph element for a given period. Using this temporal graph, one can
21 define a model that abstract long-term actions. However, in our work, we use
22 another temporal graph definition provided by Hartmann *et al.*, and implemented
23 in our research group [HFM⁺19].

24 **Sum up** Different approaches are used in the literature to represent the context,
25 the behaviour, or the structure of systems (*cf.* Table 4.1). However, only a few
26 can be used to keep the history of this information [HFN⁺14b; HFN⁺14a; MS17;
27 TOH17; HIR02; HFK⁺14a]. This feature remains a crucial concern to represent
28 long-term actions as information about delayed effects and previous circumstances
29 (next and previous context of an action). In the next section, we detail approaches
1 that model actions.

Approach	Reference	Circumstance	Effects?	Run-time?	Time?
Rule-based	[TOH17; ARS15; BCG ⁺ 12; GHP ⁺ 08; PFT03; GCH ⁺ 04]	Condition	Executable code	✗	✗
Sate machine	[ARS15; IW14; HFK ⁺ 14a; MCG ⁺ 15; FGL ⁺ 11; DBZ14; ZGC09; GPS ⁺ 13; TGE ⁺ 10]	Guard and initial state of a transition	End sate of a transition	✓	✗
Publish / Subscribe	[BdMM ⁺ 17]	Condition	Executable code	✗	✗
Goal-modelling	[MAR14; BWS ⁺ 12; BPS10]	Unsatisfied goal	Fulfilment of a goal	✗	✗
Programming language	[CG12]	Condition	Language operator	✗	✓
Event-Condition Action	[DL06; CDM09; PBC ⁺ 11]	Condition	Action code	✗	✗
Model transformation	[CPY ⁺ 14; KM90]	Query	Transformation rule	✗	✗
Formal model	[WHH10; BK11; CMR15]	Condition	Code of the action	✗	✗
Dynamic Software Product-Lines	[GS10; CCH ⁺ 13]	Condition	Selected configuration	✗	✗
Graph model	[GvdHT09]	Node	Node	✗	✗

Table 4.2: Approaches to model actions, their circumstances, and their effects (RQ1.2)

2 4.2.2 Modelling actions, their circumstances, and their ef- 3 fects

4 In Table 4.2, we regroup the different approaches of our review that model
5 actions. In this section, we describe the different categories that we identified.

6 **Approaches based on rules** One solution to model actions is to use a rule en-
7 gine [TOH17; ARS15; BCG⁺12; GHP⁺08; PFT03; GCH⁺04]. A condition and an
8 executable code characterise rules. The executable code is executed if the current
9 state of the system meets the condition. Conditions can thus serve to abstract
10 the circumstances of actions and the executable code as its effect. However, this
11 information is available at design time and lost during the execution. Moreover,
12 these approaches do not allow the representation of the side effects of an action. For
13 example, changing the fuse state has a direct effect on the fuse state. But it also
14 impacts the power grid load. We can notice two exceptions in our review: [TOH17]
15 and [ARS15]. In both cases, rules are used to trigger a state modification in a
16 state machine. We explain the advantages and disadvantages of the state machine
17 approach in the next paragraph.

18 **Approaches based on a state machine** Several approaches use a state machine
19 to represent the adaptation mechanism [ARS15; IW14; HFK⁺14a; MCG⁺15;
20 FGL⁺11; DBZ14; ZGC09; GPS⁺13; TGE⁺10]. States represent the state of the
21 system, and the transition abstract the execution of actions. One advantage of this
22 approach is that they represent both the circumstances and the effects of actions.
23 Additionally, it can be used to represent actions at design time and runtime. But
24 this link remains at a high-level. An entire state is considered as the circumstance
25 or the effect of an action while, in most cases, it just a subset of the elements of
26 the state that triggers the action or is affected by it.

27 **Approaches based on a Publish/Subscribe mechanism** In our review, we
28 found one approach that uses the publish/subscribe mechanism to trigger
29 actions [BdMM⁺17]. The circumstances are thus modelled with the condition
30 of the consumer. In this case, the action is a script. The effects are thus spread,
31 and cannot be navigated. Moreover, this solution only represents the actions at
1 design time and not their executions.

Approaches based on a goal-model In addition to the goals of the system, goal models offer the capacity to represent the actions that can achieve them. Three approaches have used this ability to represent action in their model [MAR14; BWS⁺12; BPS10]. Baresi *et al.*, extended goal model with *adaptive goals*. These goals contain a condition, objectives (to weaken or enforce some goals), and *actions* (here, an action modify goals or operation-executable code- in the goal model). Here, effects can be seen as the fulfilment of a goal. And a condition is when a goal is not satisfied anymore. However, this information is not kept over time. Furthermore, no information about the runtime execution of the actions is kept.

Approaches based on a programming language Among our findings, one approach defined a language to define actions: [CG12]. This language allows developers modelling their actions, with their conditions, and their effects. However, the language does not include a temporal dimension. Plus, the language is suitable to describe actions at design time but provide no mechanism to track them during their execution.

Approaches based on the Event-Condition Action mechanism To trigger adaptation, one can use the Event-Condition Action approach: the action is triggered if an event respects the condition. In our review, all the works use this methodology in order to weave or remove aspects of the program, following Aspect-Oriented Programming³ [CDM09; PBC⁺11; DL06]. However, there is no temporal dimension. Plus, this solution is suitable to describe actions at design time but does not allow navigation through runtime information.

Approaches based on model transformations Following the models@run-time, one way to adapt a system is to modify the model to trigger the actions. One way to do this, is to use the Query/View/Transformation [Gro16b] approach as Chen *et al.*, did in [CPY⁺14]. Here, the circumstance and the effects are represented at the model level in the query and the transformation part. When the context is represented as a graph, actions will thus be modelled as graph modifications. In [KM90], authors represent an action by adding or removing graph elements (node and edge). However, these solutions do not take into account any time

³Aspect-Oriented Programming is a programming paradigm that prones the separation of concern. For that, a program is seen as a set of aspects, each aspect implementing one concern.

2 dimension, side effects or runtime information.

3 **Approaches based on a formal model** In our review, we found three formal
4 models [WHH10; BK11; CMR15]. First, the MACODO formalisation [WHH10]
5 that defines actions as functions from one set to another. These functions represent
6 actions to reorganise a system: adding, removing, or merging group of components.
7 Second, in [BK11], the authors use the process calculus Communicating Sequential
8 Process, which allows engineers formalising reactive and concurrent systems where
9 atomic *events* are handled by *processes* (an infinite transition system). Actions here
10 correspond to a modification of a component configuration in response to an event.
11 Last, Cimatti *et al.*, defined a planning algorithm which includes time [CMR15]. In
12 their algorithm, they formalise actions as elements with a field to store the execution
13 time, possibly uncertain. None of these models adds a temporal dimension. Plus
14 these solutions do not model the execution of actions.

15 **Approaches based on dynamic software product-lines** One approach to
16 represent the possible adaptation of a system is to use a Software Product-Lines
17 model. Engineers model all the possible variations of a system. Then, at runtime,
18 the system selects the one that can achieve the requirements in the current context.
19 This approach is referred to as Dynamic Software Product-Lines and used by two
20 approaches in our findings [GS10; CCH⁺13]. Here actions can be executed to
21 achieve the selection of the different variation point. Effects are thus represented
22 as the new configuration selected. However, these solutions do not include a time
23 dimension or cannot represent runtime information.

24 **Approaches based on a graph model** Finally, the last approach found in our
25 review is to use a graph model. In [GvdHT09], researchers use a graph where
26 nodes represent possible configurations and edges represent the modification of the
27 configuration, using our wording an action. However, there is no time dimension,
28 and only design time information is modelled.

29 **Sum up** As shown in Table 4.2, the literature provides different solutions to
30 model actions. However, none of them includes a temporal dimension. Thus, even
31 if they represent effects, they cannot describe them over time. For example, they
32 will be able to model that a new server will be added, but not when. Moreover,
1 these solutions mainly remain at design time, except for those that use a state

machine. No information concerning the runtime, like the status of the execution of an action, the runtime values or the effects, are not represented. One limitation is that they do not represent side-effects over time. For example, they will represent the addition of a server but not the effects on the bandwidth or the workload. Additionally, as they do not represent runtime information, no autonomous solution can be employed to detect any unknown effects of an action to the system. The last limitation is the representation of circumstances. In the approaches of the review, they are mainly represented as the condition that triggers the action. Even if a human can guess it after, there is no direct link between the action execution and the circumstances. No model can allow automatic navigation over this time-related information.

4.2.3 Reasoning over evolving context or behaviour

In this section, we review the solutions that enable reasoning over evolving context or behaviour. That is solutions that can be used to implement adaptive systems. Table 4.3 gives an overview of our findings, detailed in the rest of this section.

Model-based approach Following the MDE methodology, one approach to reason over an evolving context or behaviour is the models@run.time paradigm [BBF09; MBJ⁺09]. Hartmann *et al.*, extended it to include a temporal dimension [HFN⁺14b; HFN⁺14a] for reasoning over the history of a system. Approaches detailed in [BdMM⁺17] and in [CPY⁺14] follow this paradigm. However, in their process, they do not consider long-term action, especially those that are under execution of with effects in the near future.

Rule-based adaptation In our review, some approaches employ a rule-based mechanism to define the adaptation mechanism [ARS15; TOH17; GHP⁺08]. However, they only reason over the context or the behaviour, not on the running actions or their future effects.

Architecture-based adaptation Architecture-based adaptations adjust the architecture of a system to achieve requirements. In our review, different approaches use this mechanism [CG12; GCH⁺04; GvdHT09; FMF⁺12]. For example, Cheng *et al.*, define a language to design actions for architecture-based adaptation. Also,

Approach	Reference	Reasoning over long-term action
Model-based	[BBF09; MBJ ⁺ 09; HFN ⁺ 14b; HFN ⁺ 14a; BdMM ⁺ 17; CPY ⁺ 14]	None
Rule-based	[ARS15; TOH17; GHP ⁺ 08]	None
Architecture-based	[CG12; GCH ⁺ 04; GvdHT09; FMF ⁺ 12]	[CG12] represents action effects at design time, used as conditions for the execution of the next action.
Simulation-based	[HFK ⁺ 14a]	Consider effects of actions to select those to execute. Do not consider running actions
Formal model	[WMA12; IW14; WHH10; BK11]	None
Complex event processing	[AFR ⁺ 10]	None
Graph model	[MS17; KM90]	None
Aspect Oriented Programming	[GB; DL06; CDM09; PBC ⁺ 11; FA04; PFT03; MBN ⁺ 09]	None
Component-based	[DL06]	None
State machine	[MCG ⁺ 15; FGL ⁺ 11; DBZ14; ZGC09; GPS ⁺ 13; TGE ⁺ 10]	Effects modelled but do not contain any runtime information about actions execution
Dynamic software product-lines	[GS10; CCH ⁺ 13]	None
Requirement-driven	[BPS10]	None
Extension of MAPE-k	[MBE ⁺ 11]	None

Table 4.3: Approaches to reason over evolving context or behaviour (RQ1.3)

the adaptive mechanism can compute a reconfiguration script by comparing the current component model with the expected one [FMF⁺12]. The only solution to consider running actions and their future effects is [CG12]. In this solution, designers can add effects to actions. The execution of the next action can be triggered by the measurement of these effects. But, a reasoning process cannot automatically navigate the action model to reason over effects before they occur.

Simulation-based adaptation In our review, one approach applies a simulation-based approach [HFK⁺14a]. In their work, the authors simulated different sequences of actions, evaluate and select the optimal one regarding the requirements. To perform this approach, they have to know the look ahead and consider the impact of each action on the system. However, they do not reason over actions being executed or their future effects.

Formal model In our review, four formal models describe adaptive systems with their adaptation mechanism [WMA12; IW14; WHH10; BK11]. For example, Iftikhar and Weyns use a state machine formalism to specify SASs [IW14]. But, none of the formal models includes a time dimension, which would allow stakeholders to consider running long-term actions.

Approach based on complex event processing Among our findings, one is using an approach based on a complex event processing engine [AFR⁺10]. Here, actions are triggered in response to an event, as much complex as necessary. Using this approach, one cannot reason over running actions and their future effects.

Approach based on a graph model Two approaches in our findings employ a graph model [KM90; MS17]. These approaches reason over a graph to trigger an adaptation process. In [MS17], authors use a temporal graph algebra, which can store the history of the context. But, none of them includes running actions in their model.

Approach based on aspect-oriented programming From what we see in our findings, aspect-oriented programming is an approach heavily used by researchers [GB; DL06; CDM09; PBC⁺11; FA04; PFT03]. More specially, they use dynamic aspect-oriented programming: aspects are automatically weaved and removed aspects in a software. Besides, Morin *et al.*, uses aspect-oriented mod-

elling [MBN⁺09]. Part of the MDE methodology, this approach models the different aspects with their pointcuts⁴. However, none of them allows reasoning over actions being executed or their future effects.

Component-based approach One approach uses a component model as a basis to reason for the adaptation mechanism [DL06]. Here, actions modify the configuration of a component to adjust the behaviour or the context of the system. But, in this approach, running actions are not considered by the adaptation process.

Approach based on a state machine Another approach widely adopted in the findings is the use of a state machine [MCG⁺15; FGL⁺11; DBZ14; ZGC09; GPS⁺13; TGE⁺10]. Transitions represent actions. Effects of action are also known and modelled. However, no information about running actions is represented and thus considered.

Approach based on dynamic software product-lines Dynamic software product-lines approaches see the adaptation mechanism as a runtime selection between different software product-lines options. However, the two solutions that use this approach in our finding [GS10; CCH⁺13] do not use information about running actions and their future effects to take decisions.

Requirement-driven adaptation In our review, Baresi *et al.*, defines a reasoning approach over a goal model [BPS10]. However, running actions are excluded from the adaptation process.

Extension of MAPE-k loop Maurer *et al.*, [MBE⁺11] extended the traditional MAPE-k loop. Authors added a step before the monitoring one called *adaptation*. In the context of cloud infrastructure, this step allows preparing the entity before deployment, such as contract establishment for service level agreement. But, the MAPE-k loop is not modified to consider running action and their effects.

Sum up In the literature, we can find different approaches to reason over an evolving context or behaviour, as depicted in Table 4.3. However, none of them provides a solution to reason over running execution of their future effects. To make decisions, current solutions only consider current, past, or future context or behaviour.

⁴A pointcut is part of the software where an aspect can be weaved or removed.

4.2.4 Modelling and reasoning over long-term actions

In the previous sections, we detailed our review concerning modelling techniques for adaptive systems. First, we saw that just a couple of approaches that model the context, the structure, or the behaviour include a time dimension. Then, despite this element, current approaches do not include actions with their circumstances and their effects over time. Finally, we saw that none of the solutions proposes an approach to reason over running actions. To answer RQ1, our review shows that no solution in the state-of-the-art represents and reason over long-term actions and their executions.

4.3 Results RQ2: data uncertainty

In this section, we detail our findings regarding the modelling of uncertainty. First, we categorise the different kinds of uncertainty addressed in the literature. Second, we explain how state-of-the-art solutions model data uncertainty. Third, we show current approaches propagate uncertainty. Finally, we conclude by answering the second research questions of our review.

4.3.1 Categories of data uncertainty

The literature provides different approaches to tackle challenges that come with different aspects of uncertainty. In Table 4.4, we give an overview of the different categories of these approaches found in our review.

Data uncertainty The main category of uncertainty addressed by the different research community is the uncertainty of data [BBM⁺18; BDI⁺17; BGG⁺13; BMB⁺18; BMM14; CGH⁺17; MWV16; SWF16; VMO16; ZAY⁺19; Hal06; JWB⁺15; ZSA⁺16; BGP92; BSH⁺06; BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; SMH11; Thr00; LTB⁺00; Plu⁺03]. In these studies, they tackle challenges due to the uncertainty that can be attached to a value, in our word a data.

Design uncertainty The modelling community studied the uncertainty in the design of a model-based solution [FSC12; FC19; EPR15; EPR14; SCH⁺13; ZSA⁺16]. An example used by Famelis *et al.*, [FSC12], is the uncertainty in modelling a state machine. One may not know with the most thorough confidence the transitions to add.

Category	Reference
Data uncertainty	[BBM ⁺ 18; BDI ⁺ 17; BGG ⁺ 13; BMB ⁺ 18; BMM14; CGH ⁺ 17; MWV16; SWF16; VMO16; ZAY ⁺ 19; Hal06; JWB ⁺ 15; ZSA ⁺ 16; BGP92; BSH ⁺ 06; BAV ⁺ 12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; SMH11; Thr00; LTB ⁺ 00; Plu ⁺ 03]
Design uncertainty	[FSC12; FC19; EPR15; EPR14; SCH ⁺ 13; ZSA ⁺ 16]
Requirement uncertainty	[WSB ⁺ 10; WSB ⁺ 09; SCH ⁺ 13]
Uncertainty in model transformations	[BBM ⁺ 18; EPR15; EPR14]
Time uncertainty	[Gar08]
Uncertainty in business process	[JWB ⁺ 15]
Environment uncertainty	[EM10; ZSA ⁺ 16]
Behaviour uncertainty	[ZAY ⁺ 19]
Hardware uncertainty	[CMR13]

Table 4.4: Categories of uncertainty addressed by the literature (RQ2.1)

Requirement uncertainty Related to design uncertainty, there is the requirement uncertainty. This uncertainty is due to the lack of confidence when stakeholders model the requirements of a system. In our finding, three studies have addressed challenges related to this kind of uncertainty [WSB⁺10; WSB⁺09; SCH⁺13].

Model transformation Due to design uncertainty, researchers advocate the use of partial models [FSC12]. As explained in Section 2.2, model transformation is a cornerstone feature in the MDE community. But, this process also contains uncertainty, tackled by three studies from our review [BBM⁺18; EPR15; EPR14].

Time uncertainty When occurring with time-related phenomena, uncertainty about when they are expected to occur exists. Garousi *et al.*, studied the time uncertainty of events in a distributed system.

Uncertain process Staying in the modelling approaches, we found one study that deals with the uncertainty in business processes [JWB⁺15]. In this paper, JimÁlnez-RamÁlrez *et al.*, studied the uncertainty in the properties of business processes.

Uncertainty in environment Systems tend to be more and more complex and evolve in an uncertain environment. To face challenges due to this uncertainty, researchers defined a new category of systems refers to as adaptive system. Here, we can add all studies found that answer RQ1 (*cf.* Section 4.2). For the sake of conciseness, here we will just add two studies [EM10; ZSA⁺16]

Uncertainty in behaviour The uncertainty in the environment can cause uncertainty in the behaviour of a system. It thus may complexify the testing phase of the system. In [ZAY⁺19], the authors tackle a challenge for the testing phase of a system with uncertain behaviour.

Uncertain hardware Software relies on faulty hardware, which can create errors in the computation and damage them. In our review, we found one study that faces this uncertainty in hardware [CMR13].

Sum up During our review, we found nine different categories of uncertainty studied in the literature shown in Table 4.4. These categories cover different phase of a software lifecycle, from requirement specification to the execution environment. In this thesis, we focus on data uncertainty: the lack of confidence in the data manipulated by a software.

4.3.2 Modelling data uncertainty

In this section, we detail the different techniques present in the literature that an engineer can use to model data uncertainty. We give an overview of the different approaches in Table 4.5.

Data type with a field for uncertainty In [BBM⁺18; BMB⁺18; MWV16; VMO16], authors use a complex type, named *UReal*, that contain two fields: one to represent the value and another one to represent the *standard uncertainty*. For example, when manipulating a dimension value, one may say that the value is $19.1\text{cm} \pm 0.1$. With the data type, an instance will have 19.1 as value and 0.1 as *standard uncertainty*. Then, based on these values, they can define a normal distribution where the mean equals the value (here 19.1) and the variance the *standard uncertainty* (here 0.1). BarbarÃa *et al.*, [BGP92] used a similar approach in their database model. In their model, a probability value (a value between 0 and 1) is attached to a database value. Finally, Schwarz *et al.*, [SMH11] attached

Approach	Reference	Used for data uncertainty
Data type with a field for uncertainty	[BBM ⁺ 18; BMB ⁺ 18; MWV16; VMO16; BGP92; SMH11]	✓
Probabilistic programming	[BDI ⁺ 17; BMM14; BGG ⁺ 13; CGH ⁺ 17; SWF16; BAV ⁺ 12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB ⁺ 00; Plu ⁺ 03]	✓
Multiple possibilities	[FSC12; FC19; EPR15; EPR14; SCH ⁺ 13; BSH ⁺ 06]	(✓)
Randomness	[Gar08]	✗
Domain specific language	[WSB ⁺ 10; WSB ⁺ 09; JWB ⁺ 15; CMR13]	✓
Model-level uncertainty	[ZAY ⁺ 19]	✓
Formal model	[Hal06; ZSA ⁺ 16]	✓

Table 4.5: Approaches to model data uncertainty (RQ2.2)

a confidence value to variables in a state machine. However, these solutions limit the representation of uncertainty to one distribution, *e.g.*, a Gaussian distribution. Although all the complexity of probability theory is hidden from the developer, it hinders its ability to choose another probability distribution that could better fit. Depending on the domain, the optimal probability distribution to represent the uncertainty of data varies. For example, the Gaussian distribution suits for metrology data [Met08] whereas Rayleigh distribution fits with GPS location data [Bor13].

Probabilistic programming The research community has investigated how to introduce probability distributions in a programming language. The different approaches are regrouped under the term *probabilistic programming* [GHN⁺14]. This strategy remains the main approach used in our review [BDI⁺17; BMM14; BGG⁺13; CGH⁺17; SWF16; BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB⁺00; Plu⁺03]. Using this approach, probability distributions can be manipulated as a variable in a programming language.

One example is the *Uncertain<T>* language developed by Bornholt *et al.*,

[BMM14]. To manipulate uncertain data, they defined an interface, *Uncertain*<*T*>, which can be specialised by any probability distribution. For example, an uncertain double is defined as followed: *Uncertain*<*double*> *un* = *new Gaussian*(4,1). Therefore, the language hides the distribution for developers. However, as in ours, they can still define and use different distributions. We strongly think that using dynamic typing and always hiding the real type, developers may not know or understand what they manipulate. It may end with runtime errors that do not provide any clear explanation. By forcing static types, we can help developers to manipulate uncertain data, but we lose in terms of flexibility [MD04].

As stated by Gordon *et al.*, [GHN⁺14], “the purpose of a probabilistic program is to implicitly specify a probability distribution”. Knowledge about the probability distribution is still required to understand and manipulate a code done by one of these languages. We think that work can be done to abstract the concepts at a higher level. Doing so, engineers can write reasoning algorithms over uncertain data, without knowledge about probability distributions. Moreover, there is a shift in how to apprehend the problem of uncertain data. Using a probabilistic program, engineers will try to see the probability of an event *E* to be in a situation *S* whereas in this work they are interested in knowing if the current instance of *E* is in *S*. For example, using a probabilistic program, engineers will see the overall probability that the temperature is greater than 20°C. In our problem, they are want to see what is the confidence, *i.e.*, the probability, that the current measurement is greater than 20°C.

Multiple possibilities In order to face design uncertainty, Famelis *et al.*, defined the concept of *partial models* [FSC12; FC19]. A partial model is a graph where elements can be annotated with *TRUE*, *FALSE*, or *MAYBE*. *TRUE* and *FALSE* respectively indicate that the graph element should be present or not. When the presence of the element is uncertain, a designer can annotate it with *MAYBE*. Eramo *et al.*, apply this approach to handle uncertainty in model transformation [EPR15; EPR14]: the process generate partial models. Partial models can also be used to reflect requirement uncertainty [SCH⁺13]. Finally, in the database community, Benjelloun *et al.*, [BSH⁺06] defined an approach where data can have different possible values. This approach is only suitable for data when different possibilities

2 can be listed.

3 **Randomness** As seen in the previous section, the time uncertainty may affect
4 some events, which can complexify the test phase of such systems. To address this
5 challenge, Garousi *et al.*, [Gar08] defines an approach where the time of occurrence
6 of events happen with a random parameter. However, this approach can only be
7 used in the testing phase to represent the lack of confidence in a value. Indeed,
8 when one has to reason over received data, such as measurement data, she cannot
9 randomly select the value.

10 **Domain-specific language** In our review, we found four studies that define a
11 specific language to handle uncertainty in their domain. First, for uncertainty in
12 requirements, Whittle *et al.*, [WSB⁺10; WSB⁺09] designed the RELAX language.
13 The language introduces fuzzy words to reflect uncertainty. For example, a re-
14 quirement could be: “The workload *SHALL NOT* be greater than *THRESHOLD*”.
15 In [JWB⁺15], the authors present a declarative language for business processes
16 that allow stakeholders adding probability information to properties. For example,
17 to handle uncertain hardware, authors of [CMR13] have implemented a language
18 that can specify reliability constraints on the execution of a function. If a language
19 engineer considers that uncertainty is a first-class citizen concern, as done by these
20 approaches, then she should integrate techniques to handle it in the language.

21 **Model-level uncertainty** Zhang *et al.*, [ZAY⁺19] uses a UML profile to define
22 different kind of uncertainties, called U-Model [ZSA⁺16]. Following this approach,
23 a designer can define a model to test CPS by generating test cases. One may use
24 this technique to model data uncertainty.

25 **Formal model** In our review, we found two formal models for uncertainty [Hal06;
26 ZSA⁺16]. First, Hall *et al.*, defined a model that implements the recommendation
27 of the GUM [Hal06]. Second, Zhang *et al.*, presented a conceptual model of
28 uncertainty which regroups different kinds of uncertainty in a CPS [ZSA⁺16]. An
29 engineer can implement one of these formal models to represent data uncertainty.

30 **Sum up** As depicted in Table 4.5, in our review, we find seven categories of
31 approach that model uncertainty, with the most widely present: probabilistic
1 programming. As shown in the table, most of these approaches can be used to

Approach	Reference	Imperceptible propagation?
Attached to language operators	[BBM ⁺ 18; BDI ⁺ 17; BGG ⁺ 13; BMB ⁺ 18; CGH ⁺ 17; MWV16; SWF16; VMO16; BAV ⁺ 12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB ⁺ 00; Plu ⁺ 03]	✓
Through state machine	[SMH11]	✓
Manual	[BBM ⁺ 18]	✗

Table 4.6: Approaches to propagate data uncertainty (RQ2.3)

Approach	Reference
Access to the confidence parameter	[BBM ⁺ 18; BMB ⁺ 18; MWV16; VMO16; BGP92; SMH11]
Access to probability features	[BDI ⁺ 17; BMM14; BGG ⁺ 13; CGH ⁺ 17; SWF16; BAV ⁺ 12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB ⁺ 00; Plu ⁺ 03]

Table 4.7: Approaches to reason over the uncertainty of data (RQ2.3)

implement data uncertainty. In this thesis, we mainly focus on using a similar approach as probabilistic programming or defining new data types that contain a field for uncertainty to help developers modelling uncertain data.

4.3.3 Propagation and reasoning over uncertainty

In this section, we study state-of-the-art approaches to propagate uncertainty when uncertain data are manipulated. Plus, we study the different approaches to reason over uncertainty. Table 4.6 and Table 4.7 present a summary of our findings.

[Propagation] Attached to language operators According to our finding, the most common approach to propagate uncertainty is to attach the propagation mechanism to the language operator [BBM⁺18; BDI⁺17; BGG⁺13; BMB⁺18; CGH⁺17; MWV16; SWF16; VMO16; BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB⁺00; Plu⁺03]. From a programming language point of view, (uncertain) data are mainly manipulated through language operators, such as arithmetic operators. In these works, researchers define strategies to map the

2 semantics of a language operator to a process that propagates the uncertainty.
3 For example, mapping the addition operator to the addition of two probability
4 distributions.

5 **[Propagation] Through state machine** One study propagate the uncertainty
6 using a state machine [SMH11]. In this work, uncertain events are handled by
7 *interactors*. Due to the uncertainty, multiple interactors can match the event. They
8 are called, and their results are weighted according to the uncertainty of the event.
9 The state machine can thus be in different states, with different confidence. The
10 most probable one is selected.

11 **[Propagation] Manual propagation** Finally, one solution requires manual
12 propagation [BBM⁺18]. This work is used in model transformations. When a
13 designer implements a transformation rule, she also has to implement the code
14 manually to propagate uncertainty.

15 **[Reasoning] Access to the confidence parameter** Six approaches in our
16 findings allow accessing to the confidence parameter. For example, using [VMO16]
17 developers can access to the *standard uncertainty*.

18 **[Reasoning] Access to properties of probability distribution** Probabilis-
19 tic programming allow the manipulation of probability distributions as programming
20 language variables. Using such approaches, developers can access different proper-
21 ties of a probability distribution [BDI⁺17; BMM14; BGG⁺13; CGH⁺17; SWF16;
22 BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB⁺00; Plu⁺03].
23 For example, they can access to the mean or the variance of a Gaussian distribution.
24 Of, they can compute a sample from the distribution.

25 **Sum up** The widely used approach to imperceptibly propagate uncertainty is
26 to attach the propagation uncertainty to the language operators as we can see
27 in Table 4.6. The main advantage of this approach is to keep a language with
28 a syntax as close as possible to what developers are used to. However, current
29 solutions provide, what we call, *low-level* techniques to reason over uncertainty (*cf.*
30 Table 4.7). Indeed, state-of-the-art solutions allow developers to access either the
1 values stored or the properties of a probability distribution.

4.3.4 Modelling of data uncertainty and its manipulation

In this review, we have seen that different kinds of uncertainty have been addressed by the literature (RQ2.1). Among them, in this thesis, we focus on data uncertainty. Different strategies can be used to model the uncertainty, with the most used one, which consists of using a probabilistic program (RQ2.2). This approach offers the propagation of uncertainty by mapping this process to language operators (RQ2.3). Plus, the properties of probability distributions can also be accessed using such techniques (RQ2.3). However, we think that research efforts now can be done to provide a language with a higher level of abstraction, as initiated by Vallecillo *et al.*, [VMO16]. One goal is to help developers implementing reasoning algorithms over uncertain data with a high-level understanding of the probability theory. Moreover, we think that specific operators should be specified to reason over this uncertainty. Another limitation of these works is that they studied uncertainties on numerical values. There are still open research questions to employ these techniques in an object-oriented language, which also contains references, nested objects or hierarchical relations.

4.4 Threat to validity

In this section, we discussed several threats to validity that may damage the results of our review.

To perform this review, we use a strategy inspired by the snowballing methodology [Woh14]. However, due to a lack of resources, we did not apply the backward and forward propagation on all the selected papers. Additionally, as mentioned by the authors, the quality of the results of a review that follow this methodology highly depends on the starting set, which can suffer from bias.

Moreover, the results of a review are impacted by the accuracy of the data extraction step and the research questions formulated. In this case, no discussion has been done around the formulation of the research questions. And the data extraction has been performed by a unique person, which will increase the inaccuracy.

2 4.5 Conclusion

3 In this chapter, we review the state-of-the-art approaches to answer two research
4 questions. First, we look for studies that model adaptive systems, their contexts or
5 behaviours to see if they also consider long-term action (RQ1).

6 In Section 4.2, we answer the first research question. We first show that the
7 literature provides a few solutions to keep the history of the structure, context, or
8 behaviour of systems [HFN⁺14b; HFN⁺14a; MS17; TOH17; HIR02; HFK⁺14a].
9 The other solutions, depicted in Table 4.1, such as goal models [CvL17; IW14;
10 MAR14; CPY⁺14; BPS10] or object-based models [HIR02; HFK⁺14a; TOH17] do
11 not support the time dimension natively. However, this feature remains important
12 to enable the modelling of long-term actions. Then, we show that none of the
13 state-of-the-art approaches allows stakeholders to model long-term actions. Indeed,
14 they do not incorporate a time dimension in their models. We give an overview
15 of the different solutions in Table 4.2. Finally, we show that, using one of the
16 current solutions, one cannot reason over running actions with their expected effects.
17 Some solution exists to represent action effects at design time. First, those that
18 employ a state machine [MCG⁺15; FGL⁺11; DBZ14; ZGC09; GPS⁺13; TGE⁺10]
19 represent actions as state transitions. Effects are thus modelled as the target state.
20 In [HFK⁺14a], the authors model and use the effects of actions to simulate different
21 sequences of actions. Lastly, researchers defined the Stitch language [CG12] that
22 can abstract action effects. They are used as conditions for the execution of the next
23 action. However, none of them models runtime information, useful, for instance,
24 during a diagnosis task.

25 Then, we search for solutions that model data uncertainty and present our
26 findings in Section 4.3. We first explain that the literature addressed nine kinds
27 of uncertainties, depicted in Table 4.4. Among them, the most studied one is
28 the uncertainty relative to data [BBM⁺18; BDI⁺17; BGG⁺13; BMB⁺18; BMM14;
29 CGH⁺17; MWV16; SWF16; VMO16; ZAY⁺19; Hal06; JWB⁺15; ZSA⁺16; BGP92;
30 BSH⁺06; BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; SMH11; Thr00;
31 LTB⁺00; Plu⁺03]. In this thesis, we also concentrated on this kind of uncer-
32 tainty. Then, we detail how researchers model data uncertainty (*cf.* Table 4.5).
1 In our findings, the most used one is the probabilistic programming [BDI⁺17;

BMM14; BGG⁺13; CGH⁺17; SWF16; BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB⁺00; Plu⁺03]. Using this programming paradigms, developers can manipulate probability distribution as variables. They can define complex probability distributions that result from a sequence of operations. Another approach is to consider a variable as a pair of a standard deviation and a value [BBM⁺18; BMB⁺18; MWV16; VMO16; BGP92; SMH11]. Finally, we look for studies that propose a solution to propagate or reason over uncertainty (*cf.* Table 4.6). The principal approach found to propagate the uncertainty is to map this operation to language operators [BBM⁺18; BDI⁺17; BGG⁺13; BMB⁺18; CGH⁺17; MWV16; SWF16; VMO16; BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB⁺00; Plu⁺03]. Concerning reasoning approaches, we find only two ways, as shown in Table 4.7. The first one consists in giving access to the confidence parameter [BBM⁺18; BMB⁺18; MWV16; VMO16; BGP92; SMH11]. The second one is to allow developers to read and manipulate features of the probability distribution [BDI⁺17; BMM14; BGG⁺13; CGH⁺17; SWF16; BAV⁺12; CNR13; JK12; PPT08; Pfe01; RP02; SCG13; Thr00; LTB⁺00; Plu⁺03].

To summarise, our review shows that none of the current approaches model or enable reasoning over actions with delayed effects. Thus, some research efforts are still required to specify solutions that allow designers to add long-term action in their model and to implement techniques to reason over them. In this thesis, we start studying this problem, and we present a model-based solution, detailed in Chapter 7. Moreover, it different solutions have been found in our review to manage data uncertainty, the main one being referred to as probabilistic programming. This solution allows developers to manipulate probability distributions as common variables of a programming language. However, as seen in our review, not all kinds of uncertainty can be represented by this approach. For example, some researchers represent the uncertainty of a value with a set of different possibilities. Therefore, open challenges still need research efforts to handle data uncertainty. Towards solving these challenges, we start by defining a language that integrates data uncertainty as a first-class citizen (*cf.* Chapter 6).

Part II

Towards a modelling frameworks for adaptive systems

2

3 Uncertainty and time in modeling frameworks

4 Contents

5 **5.1 Summary of previous chapter 82**

6 **5.2 Vision 82**

7
8
10

11 *This chapter describes the vision defended in this document. We first summarise*
 12 *the challenges in modelling adaptive systems. Then, we explain our vision and*
 1 *introduce our two contributions that support it.*

2 In this chapter, we present the vision that we defend in this document. We first
3 summarise the challenges addressed and our review of the state-of-the-art. Then,
4 we describe our vision.

5 5.1 Summary of previous chapter

6 Adaptive systems engineer can use the MDE methodology to design the adap-
7 tation process *cf.* Chapter 1). Among the open challenges we identified for this
8 method, this thesis tackles the followings:

- 9 • How to ease the manipulation of data uncertainty for software engineers?
- 10 • How to enable reasoning over unfinished actions and their expected effects?
11 (long-term actions)
- 12 • How to model the decisions of an adaptation process to diagnose it?

13 We show in Chapter 4 that it exists no solution to model long-term action or
14 to reason over them. Besides, we establish that research efforts are still required
15 to handle data uncertainty at the language level. To come to this conclusion, we
16 perform a review driven by two research questions (*cf.* Section 4.1):

- 17 • *[RQ1]* Do state-of-the-art solutions that model adaptive systems allow repre-
18 senting and reasoning over long-term actions?
- 19 • *[RQ2]* Do state-of-the-art solutions allow modelling uncertainty of data and
20 its manipulation (propagation, reasoning over)?

21 5.2 Vision

22 We think that both challenges can be addressed by the definition of a modelling
23 framework that includes, besides all traditional elements, temporal and uncertainty
24 as first-class concepts. We depict this vision in Figure 5.1. Using this framework,
25 designers can define the structure of the data model, in the figure shown as
26 the metamodel, and the adaptation process. The metamodel contains high-level
27 information about the time and uncertainty. Based on this information, the
28 uncertainty quantification process and a high-level API will be generated, both
29 depicted in blue in the figure. The process will read data of the model and add
30 uncertainty information. If we take a smart grid example, when the system receives
31 a consumption value, the process adds a probability distribution that represents the
1 uncertainty of the value. The uncertainty quantification process, as the adaptation

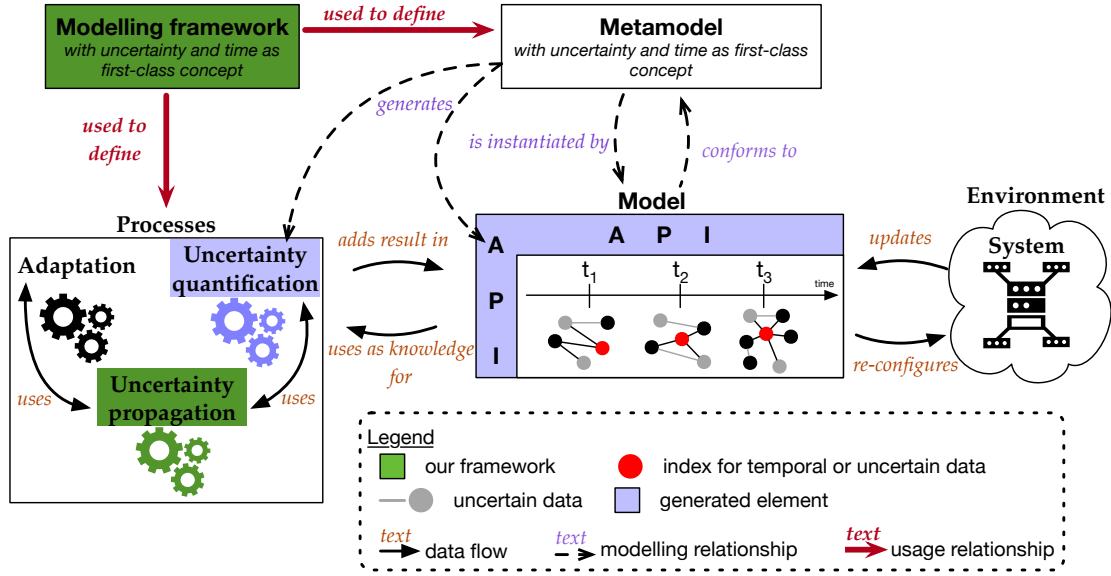


Figure 5.1: Overview of our vision

process, uses a language with uncertainty and time as first-class citizens. Therefore, they both use a process that automatically propagates uncertainty, which is part of our modelling framework. The API generated from the metamodel will help stakeholders to manipulate the model, conforms to the metamodel. This API will use specific indexes for temporal and uncertain data. These indexes will help the manipulation of large models. In the figure, we don't depict the storage mechanism nor the mechanism that load data.

Hartmann *et al.*, partially implemented this vision [HFM⁺19], which led to the GCM. Using this framework, a modeller will not specify any time-related information in the models. They will be automatically generated by the framework, with dedicated data structures for temporal data to enable efficient storage and query.

We present, in this thesis, two contributions towards this vision. First, we detail our language named Ain'tea, which permits designers managing uncertainty at the language level, in Chapter 6. We can assume that the language is part of the modelling framework, used to define reasoning processes such as the adaptation process. This solution addressed the challenge of the manipulation of uncertain data (cf. Sub-Challenge #1). Second, we describe a temporal knowledge model

2 in Chapter 7 to structure and store the state and behaviour of a running adaptive
3 system, with running long-term actions. This model addresses the challenge of
4 reasoning over unfinished actions, and understanding of adaptive system behaviour
5 (cf. Sub-Challenge #2 and #3). This metamodel can be understood as an example
1 of a result of our modelling framework.

Ain'Źtea: managing data uncertainty at the language level

Contents

6.1	Uncertainty as a first-class language citizen	87
6.2	Evaluation	105
6.3	Conclusion	115

After identifying and discussing the key concepts associated with data uncertainty, this chapter presents Ain'tea, a language that integrates them directly into the grammar, type system and semantics part. To validate and exemplify our approach, we apply it to a smart-grid scenario and compare it to framework-based approaches. We show that developers benefit from the language semantics and type system which guide them to manipulate uncertain data without deep probability theory knowledge.

In this chapter, we present the contribution that tackles Sub-challenge #1 described in Section 1.3, concerning data uncertainty. Our contribution is split in three parts.

- Description and definition of the main concepts and operators to introduce in a language with uncertainty as a first-class citizen. Doing so, developers will be able to manipulate uncertain and certain data in a similar way, hiding complex concepts behind ones they are used to.
- An uncertainty-aware static type checker enabling the design-time detection of programming errors. This type checker can be used as *implicit documentation*, which helps developers in implementing uncertainty-aware algorithms.
- We implement these two contributions in the Ain'tea language, publicly available¹. An overview of this language is depicted in Figure 1.4. Then, we use a real-world case study, built with our partner Creos S.A., to answer the two following research questions:
 - *RQ1*: Does the uncertainty management have an impact on the conciseness of a language?
 - *RQ2*: Can the type system detect errors related to uncertainty management?

Our use-case implementation shows first that our approach does not impact the conciseness of the language (RQ1). Second, it highlights the feasibility and the advantages (RQ2) of an uncertainty-aware type checking system on the language level.

The remainder of this chapter is as follows. In Section 6.1, we present and discuss the main concepts and operators for uncertainty management as a first-class citizen of a language. Section 6.2 presents our Ain'tea implementation and validates the design time error detection and its impact on language conciseness. Section 6.3 concludes and presents some open research perspectives.

¹<https://github.com/lmouline/aintea/>

6.1 Uncertainty as a first-class language citizen

6.1.1 Language overview

Data are uncertain when some data points are not precisely known. This lack of confidence is generally represented by a probability distribution, like the Gaussian distribution [Met08]. Let D be a set of data points and P_D a set of probability distributions. **An uncertain data point $u_d \in U_D$ is therefore defined by a pair (d, p_d) , where $d \in D$ and $p_d \in P_D$. This representation permits to store a value, which has been observed, computed, given, or measured with a probability distribution that represents the certainty.**

Thus, we associate one data point to the value of a variable in a programming language. Adding uncertainty as a first-class language citizen thus implies to enable the creation and the manipulation of such pairs. We propose to add new data types to represent such pairs and define operators on top of these data types to manipulate them. Uncertainty propagation is done through these operators. This impacts the different elements of our language [HR04; Deg16]: the syntax (abstract and concrete) and the semantics (static and dynamic). This is summarised in Figure 6.1.

In the abstract syntax, we define the probability distributions and uncertain data types as primitive types. Plus, it contains a definition of the operators to manipulate them. In this article, we do not propose a concrete syntax as part of the contribution, we only give an example through our implementation, Ain'tea. In the static semantics, the type system implements typing rules that follow the probability theory. Moreover, meaningful error messages for end users are provided. Finally, dynamic semantics define how the different operators are executed and thus how uncertainty is propagated. We define the dynamic semantics following the operational approach [Mos01]. It also defines the execution of the creation or deletion of uncertain data in the program.

To facilitate reading, we start by explaining the principles behind the manipulation of uncertain data based on the boolean case and deal with the numeric case (discrete and continuous) afterwards. We define the operators that can be applied on such data as well as their formal semantics.

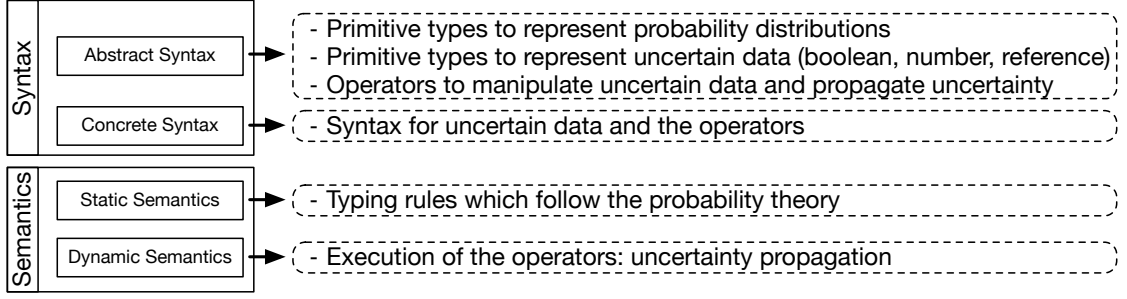


Figure 6.1: Impact of having uncertainty as a first-class language citizen on a language

6.1.2 Uncertain boolean

What an uncertain boolean is?

Boolean variables can take only two possible values, *TRUE* and *FALSE*. For a given data point d , the levels of confidence that d actually takes each of these values are thus proportionally tied to each other. The higher the confidence level of one value is, the lower the confidence level of the other. More precisely, if d takes the value *TRUE* with probability c then it takes the value *FALSE* with a probability $(1 - c)$. That is, $P(b = \text{TRUE}) = 1 - P(b = \text{FALSE})$, with $P(X)$ a function that computes the probability of X .

Bernoulli distribution for uncertain boolean values

We use the Bernoulli probability distribution [Wal96] to represent the confidence level of uncertain boolean values. $\text{Bernoulli}(p)$ denotes a random variable that equals 1 with a probability of p and 0 with a probability of $(1 - p)$. Without loss of expressiveness, we arbitrarily decide that p represents the probability that the data point takes the *TRUE* value. In other words, a *TRUE* value with a confidence level c_1 is associated with a $\text{Bernoulli}(p = c_1)$, while a *FALSE* value with a confidence level (c_2) is associated with a $\text{Bernoulli}(p = 1 - c_2)$. Thus, the uncertain booleans $(\text{TRUE}, 0.76)$ and $(\text{FALSE}, 0.24)$ differ in their observed value but not in the probability distribution.

More generally, in our proposed language, an uncertain boolean is thus represented as a pair (d, p_d) , with d is a boolean value and p_d is a Bernoulli distribution set with the probability of the *TRUE* value. **The abstract syntax of our lan-**

guage contains thus the Bernoulli distribution and a definition of the pair (boolean value, Bernoulli distribution). Existing approaches [BMB⁺18] store only the Bernoulli distribution and use the aforementioned equivalence to convert a *FALSE* value to a *TRUE* one. We find these approaches to be counter-intuitive as developers would only manipulate *TRUE* values, regardless of the value actually observed. As we want to keep the data manipulation as close as possible to common programming languages, we decide to represent a boolean as the composition of both the value and the Bernoulli distribution. Moreover, our approach keeps the initial value, which has been observed or measured.

The abstract syntax also contains the following operators. First, the definition of boolean (and, or, not) and equality ($=$, \neq) operators are extended to uncertain booleans. In addition, we define the cast operator between these two data types. Finally, one may reason over the confidence level. We thus define four novel operators: existence, confidence, dot, and uncertain equality. We detail them in the next section.

Operational semantics

We denote by $(b, B(p))$ an uncertain boolean with a boolean value b and a Bernoulli distribution $B(p)$ with probability p . In order to distinguish between the equality operator and the mathematical equality symbol, in the rest of this article, we represent the first by $=$ and the second by $:=$. We define the following uncertain booleans to exemplify the different operators: $u_{b1} := (TRUE, B(0.3))$, $u_{b2} := (TRUE, B(0.65))$, and $u_{b3} := (FALSE, B(0.45))$.

Operator 1 The *existence operator* returns true if an uncertain data point $u_d \in U_D$ has a value with at least a given probability $t \in C$, where $C := \{c \in \mathbb{R} \mid 0 \leq c \leq 1\}$, and false otherwise: $U_D \times C \xrightarrow{\text{exists}} B$.

Using this operator, developers know if it exists at least one value for which its confidence level is greater than a given threshold. For example, using it, one may know if a fuse is open ($isOpen := TRUE$) or closed ($isOpen := FALSE$) with a confidence of at least 80%. The operational semantics of this operator is specified by the following function:

$$\text{exists}((v, B(p)), t) := (p \geq t) \parallel (1 - p \geq t)$$

2 Applying the operator on u_{b1} , we thus get:

- 3 • $exists(u_{b1}, 0) := (0.3 \geq 0) \parallel (0.7 \geq 0) := TRUE$,
- 4 • $exists(u_{b1}, 0.5) := (0.3 \geq 0.5) \parallel (0.7 \geq 0.5) := TRUE$,
- 5 • $exists(u_{b1}, 0.9) := (0.3 \geq 0.9) \parallel (0.7 \geq 0.9) := FALSE$.

6 In plain English, these examples mean: considering u_{b1} , there exists a value with a
 7 confidence level of at least 0 and 0.5 but there is no value with a confidence level
 8 of at least 0.9.

9 **Operator 2** The **confidence operator** computes the most confident value D
 10 with a minimal confidence level $t \in C$ of a given uncertain data point $u \in U_D$:
 11 $U_D \times C \xrightarrow{confidence} D$.

12 Taking an example from the smart grid domain, one may use this operator to
 13 get the most probable state of the fuse (open or close), if its confidence is superior
 14 to 80%.

15 This operator raises an error in two cases: when (i) no value exists with at least
 16 the given confidence level or (ii) the confidence level of both values, true or false,
 17 are equal. Its semantics is thus:

$$confidence((v, B(p)), t) := \begin{cases} true, & \text{when } exists((v, B(p)), t) \wedge p > 1 - p \\ false, & \text{when } exists((v, B(p)), t) \wedge p < 1 - p \\ \perp, & \text{when } p = 0.5 \end{cases}$$

18 If we apply this operator on u_{b1} with the same base as the previous example,
 19 then it returns:

- 20 • $confidence(u_{b1}, 0) := FALSE$,
- 21 • $confidence(u_{b1}, 0.5) := FALSE$,
- 22 • $confidence(u_{b1}, 0.9) := ERROR$.

23 With a confidence level of at least 0 or 0.5, u_{b1} is more likely to be equal to *FALSE*.
 24 Otherwise, we cannot know its value with a minimal confidence level of 0.9.

25 **Operator 3** The **cast operator** casts an uncertain data point to a certain one
 26 and vice-versa. The cast operation from uncertain to certain is defined as follow:
 1 $U_D \xrightarrow{cast} D$. Formally, the opposite operation is described as follows: $D \xrightarrow{cast} U_D$.

2 The cast operation from an uncertain to a certain one can be used to get the
 3 most confident value, without any constraint on the confidence (which is done by
 4 the confidence operator). This helps developers to reason or to make decisions
 5 upon the most probable situation. It can be performed by using the confidence
 6 operator with 0 as given confidence level:

$$cast(u_b) := confidence(u_b, 0)$$

7 The opposite operation, from certain to uncertain, is mainly a syntactic ma-
 8 nipulation. Indeed, certain data is always considered as uncertain data with the
 9 maximum confidence level. In the case of uncertain boolean, it's 1:

$$cast(b) := \begin{cases} (b, B(1)), & \text{when } b = TRUE \\ (b, B(0)), & \text{when } b = FALSE \end{cases}$$

10 Performing this operator on our example results in:

- 11 • $cast(u_{b1}) := confidence(u_{b1}, 0) := FALSE$,
- 12 • $cast(u_{b2}) := confidence(u_{b2}, 0) := TRUE$,
- 13 • $cast(TRUE) := (TRUE, B(1))$,
- 14 • $cast(FALSE) := (FALSE, B(0))$,

15 **Operator 4** The ***dot operator*** allows accessing both elements of uncertain data:
 16 the value $d \in D$ or the probability distribution $p_d \in P_D$: $U_D \xrightarrow{dot} D \cup P_D$.

17 This operator allows accessing the elements that compose uncertain data. For
 18 the value, it permits to resolve what is the value measured, observed, computed or
 19 given, without uncertainty consideration using the *value* keyword:

$$(v, B(p)).value := v$$

20 For the confidence, it gives access to the confidence in order to reason over the
 1 probability distribution using the *confidence* keyword:

$$(v, B(p)).confidence := B(p)$$

2 Used on u_{b1} and u_{b3} , this operator will return:

- 3 • $u_{b1}.value := TRUE$,
- 4 • $u_{b1}.confidence := B(0.3)$,
- 5 • $u_{b3}.value := FALSE$,
- 6 • $u_{b3}.confidence := B(0.45)$,

7 **Operator 5** When applied between two uncertain data, **uncertain equality op-**
 8 **erators** ($=, \neq$) compute the probability that both uncertain values are equal or not:
 9 $U_D^2 \xrightarrow{u-equality} U_B$, where U_B is the set of uncertain booleans. When used between
 10 an uncertain and a certain value, they return the probability that the uncertain
 11 variable is equal to the certain value: $U_D \times D \xrightarrow{u-equality} U_B$.

12 In both cases, the resulting value of the uncertain boolean is computed by apply-
 13 ing the operator to the boolean values. When used between two uncertain booleans,
 14 the calculated probability corresponds to the probability that both values are equal
 15 to $TRUE$ or both equal to $FALSE$. To do so, we compute the union of the inter-
 16 section of the different probabilities. It results therefore in the following semantics:

$$\begin{aligned}
 (v, B(p)) = b &:= \begin{cases} u_b[v = b, B(p)], & \text{when } b = TRUE \\ u_b[v = b, B(1 - p)], & \text{when } b = FALSE \end{cases} \\
 (v_1, B(p_1)) \neq b &:= \begin{cases} u_b[v \neq b, B(1 - p)], & \text{when } b = TRUE \\ u_b[v \neq b, B(p)], & \text{when } b = FALSE \end{cases} \\
 (v_1, B(p_1)) = (v_2, B(p_2)) &:= (v_1 = v_2, [B(p_1) \cap B(p_2)] \cup [B(1 - p_1) \cap B(1 - p_2)]) \\
 (v_1, B(p_1)) \neq (v_2, B(p_2)) &:= (v_1 \neq v_2, [B(p_1) \cap B(1 - p_2)] \cup [B(1 - p_1) \cap B(p_2)])
 \end{aligned}$$

17 To compute the intersection of the union of Bernoulli distributions, first, we
 18 should evaluate whether they are disjoint and dependent. We call two variables
 19 dependent if they are, directly or indirectly, defined based on at least one common
 20 variable (uncertain or not). For example, let $temp$ be a temperature, the boolean
 21 $b_1 = t \leq 0$ and $b_2 = t > 18$ are dependent because they share the same variable
 22 t . As both values are directly defined using t , they are dependent. We call two
 1 variables disjoint when they do not share the same a common set of possible values.

2 In our example, b_1 and b_2 are disjoint because they don't share any possible values,
 3 *i.e.*, $] - \infty, 0] \cap]18, +\infty[= \emptyset$.

4 Below we illustrate the different formulas to compute the union or intersection
 5 of Bernoulli distributions. Overall, there are three cases: (i) disjoint variables,
 6 regardless if they are independent or not, (ii) independent and non-disjoint variables,
 7 and (iii) dependent and non-disjoint variables. To the best of our knowledge, there
 8 is no formula in the latter case. An exception is raised in such cases.

$$\begin{aligned}
 B(p_1) \cap B(p_2) &= 0 && \text{(Disjoint var.)} \\
 B(p_1) \cup B(p_2) &= B(p_1 + p_2) \\
 B(p_1) \cap B(p_2) &= B(p_1 * p_2) && \text{(Indep. and non-disjoint var.)} \\
 B(p_1) \cup B(p_2) &= B((p_1 + p_2) - (p_1 * p_2)) \\
 B(p_1) \cap B(p_2) &= \perp && \text{(Dep. and non-disjoint var.)} \\
 B(p_1) \cup B(p_2) &= \perp
 \end{aligned}$$

9 Let us consider u_{b1} and u_{b3} two independent and non-disjoint variables. Applying
 10 the equality operators on them will result in:

$$\begin{aligned}
 11 \quad &\bullet u_{b1} = u_{b3} := (TRUE = FALSE, B(0.3 * 0.45) \cup B(0.7 * 0.55)) \\
 12 \quad &\quad := (FALSE, B(0.135 + 0.385)) := (FALSE, B(0.52)) \\
 13 \quad &\bullet u_{b1} \neq u_{b3} := (TRUE \neq FALSE, B(0.3 * 0.55) \cup B(0.7 * 0.45)) \\
 14 \quad &\quad := (TRUE, B(0.165 + 0.315)) := (FALSE, B(0.48))
 \end{aligned}$$

15 In plain English, u_{b1} and u_{b3} have a probability of 52% of having the same value,
 16 *i.e.*, 48% of having different values.

17 **Operator 6 Identity operators** ($==, \neq$) return true if two uncertain data are
 1 identical: same value and same probability distribution. Formally, $U_D^2 \xrightarrow{\text{identity}} B$

This operator is an extension of the equal operator available in programming languages or the equals method in Java. It compares the probability parameter of the Bernoulli distribution and the values of the uncertain booleans:

$$\begin{aligned}
 (v_1, B(p_1)) == (v_2, B(p_2)) &:= (p_1 = p_2) \wedge (v_1 = v_2) \\
 (v_1, B(p_1)) \neq (v_2, B(p_2)) &:= p_1 \neq p_2 \vee (v_1 \neq v_2)
 \end{aligned}$$

2 For example, the comparison of u_{b1} and u_{b2} returns:

- 3 • $u_{b1} == u_{b2} := (0.65=0.3) \wedge (TRUE=TRUE) := FALSE,$
- 4 • $u_{b1} \neq u_{b2} := (0.65 \neq 0.3) \vee (TRUE \neq TRUE) := TRUE.$

5 **Operator 7 Uncertain boolean operators** (\wedge, \vee, \neg) compute the result of the
6 boolean operation and its confidence: $U_B^2 \xrightarrow{u\text{-booleans}} U_B$

The value of the resulting uncertain boolean is computed by applying the boolean algebra on the values of both input uncertain booleans. The second part of the computation combines (union and intersection) their probability distributions:

$$\begin{aligned} (v_1, B(p_1)) \wedge (v_2, B(p_2)) &:= (v_1 \wedge v_2, B(p_1) \cap B(p_2)) \\ (v_1, B(p_1)) \vee (v_2, B(p_2)) &:= (v_1 \vee v_2, B(p_1) \cup B(p_2)) \\ \neg(v_1, B(p_1)) &:= (\neg v_1, B(1 - p_1)) \end{aligned}$$

7 If we assume u_{b1} and u_{b3} independent and non-disjoint variables, applying these
8 operators return:

- 9 • $u_{b1} \&\& u_{b3} := (TRUE \&\& FALSE, B(0.65*0.45)) := (FALSE, B(0.2925)),$
- 10 • $u_{b1} \parallel u_{b3} := (TRUE \parallel TRUE, B(0.65 + 0.45 - 0.65*0.45)) := (TRUE,$
11 $B(0.8075)),$
- 12 • $\neg u_{b1} := \neg(\neg TRUE, B(1 - 0.65)) := (FALSE, B(0.35)).$

13 6.1.3 Uncertain number

14 What is an uncertain number?

15 Numerical variables can theoretically take an infinite number of values. They
16 can be either on a continuous domain, simulated by floating-point values, or on
17 a discrete one, simulated by integer values. The confidence level of such data is
18 either **precisely linked to the value** (observed, measured, ...) or **is distributed**
19 **over the range of possible values**. For example, values get from sensors are
20 always given with a certain accuracy, *e.g.*, $18.4^\circ\text{C} \pm 0.1^\circ\text{C}$. The confidence level
21 of the measured values is thus distributed over a range. Values set by humans
22 are also uncertain due to possible human errors. However, there might be no
23 information concerning the distribution of this uncertainty. The confidence level is
1 thus attached to the precise value.

Type	Data type	Gaussian	Rayleigh	binomial	Dirac
Continuous	float, double	✓	✓		✓
Discret	byte, short, integer, long			✓	✓

Table 6.1: Which distribution can be used to represent the uncertainty of which data type

Representation of uncertain numbers

Like uncertain booleans, uncertain numbers are composed of two elements: a numerical value and a probability distribution that represents its uncertainty. According to the nature of the numerical value, developers have to decide which distribution fits the uncertainty of their variable. In probability theory, one distinguishes between continuous distributions and discrete distributions. The former defines the distribution of the probability density over a continuous domain. The latter instead apply to a discrete domain. In our proposed language, we support the following distributions: two continuous probability distributions, Gaussian [Wal96] and Rayleigh [Wal96], and one discrete distribution, the binomial distribution [Wal96]. In addition, we support the Dirac delta function [GS64], which can be used on a continuous or a discrete domain. We refer to Chapter 2 for a detailed description of these distributions.

In our language, we add new data types that represent each distribution. In addition, we add a new type for all possible combinations, described in Table 6.1. Existence, confidence, dot, cast and uncertain equality operators are also defined for these data types. We add operators specific to numerical values: arithmetic and comparison operators. The semantics of these operators are formalised in the next section.

Operational semantics

We denote by (v, p_d) an uncertain number with a numerical value v and a probability distribution $p_d \in P_D$. The following uncertain numbers are defined to be used as examples for the description of the operators:

- $u_{n1} := (7, \text{binomial}(20, 0.37))$,
- $u_{n2} := (10, \text{binomial}(30, 0.33))$,
- $u_{n3} := (12, \text{Gaussian}(12, 9))$,

- $u_{n4} := (23, \text{Gaussian}(23, 5))$.

Operator 1 - Existence operator.

In probability distributions, the value with the highest probability is called the mode [MGB63]. We denote it by m . For discrete uncertain numbers, the operator, defined in Operator 1, returns *TRUE* if the probability of the mode value is greater or equal to a given base:

$$\text{exists}((v, p), t) := P(x = m) \geq t, \text{ where } P(x = m) \text{ is the probability of } m$$

Based on u_{n1} , we thus get:

- $\text{exists}(u_{n1}, 0) := P(x=7) \geq 0 := 0.1542985 > 0 := \text{TRUE}$,
- $\text{exists}(u_{n1}, 0.5) := P(x=7) \geq 0.5 := 0.1542985 > 0.5 := \text{FALSE}$,
- $\text{exists}(u_{n1}, 0.9) := P(x=7) \geq 0.9 := 0.1542985 > 0.9 := \text{FALSE}$.

This operator allows verifying that there is no value with a confidence level of at least 50 or 90% for u_{n1} , but there is at least one value with a non-null confidence level.

For uncertain continuous numbers, no semantic exists for a continuous domain since the probability of the mode cannot be computed for continuous probability distributions, $\int_m^m P(u_n) dx = 0$:

$$\text{exists}((v, p), t) := \perp$$

Operator 2 - Confidence operator.

The semantics of the confidence operator relies on the existence operator. For uncertain discrete numbers, it returns the mode if the existence operator returns true:

$$\text{confidence}((v, p), t) := \begin{cases} m, & \text{if } \text{exists}((v, p), t) \\ \perp, & \text{otherwise} \end{cases}$$

Applying this operator on the u_{n1} using the same base as in the previous example, the operator returns the mode value, 7, only for the first case:

- $\text{confidence}(u_{n1}, 0) := 7$,
- $\text{confidence}(u_{n1}, 0.5) := \perp$,

- $confidence(u_{n1}, 0.9) := \perp$.

This means that the most probable value which has at least zero as confidence level is 7 for u_{n1} .

For continuous numbers, as this operator is based on the existence operator, which is not defined, it is also not defined:

$$confidence((v, p), t) := \perp$$

Operator 3 - Cast operator. In addition to the cast operator between uncertain and certain data, this operator is extended to support casting operations between two uncertain numbers, when an approximation function between two distributions exists. In Table 6.2, we summarize the permitted casts in our language.

When casting an uncertain number (discrete or continuous) to a certain one, we return the value with the highest confidence, *i.e.*, the mode m .

$$cast((v, p) := m$$

For example, casting u_{n1} and u_{n3} to certain numbers will result in:

- $cast(u_{n1}) := 7$,
- $cast(u_{n3}) := 12$.

When casting a certain to an uncertain number, we should be able to specify a distribution that represents a confidence level of 100% on a precise value. This is not always possible with all distributions. From the distributions implemented in our language, it is only possible for the binomial distribution and the Dirac delta function. For the other distributions, it is either impossible, like for the Poisson distribution (not implemented here) or it requires domain-specific heuristics. For example, the Gaussian distribution can be initialized with a variance value that is as close as possible to zero. But the definition of “as close as possible” differs according to the domain. For instance, when handling temperature values, 0.01 could be considered sufficiently close to 0. Nonetheless, this accuracy is not sufficient for other fields such as meteorology. As no global value could be selected, we decided to forbid such casting operations when it is not possible without heuristics. We, therefore, define the following semantics:

$$cast(v) := \begin{cases} (v, Dirac(coeff = 1, shift = v)), & \text{when the expected distribution} \\ & \text{is a Dirac delta function} \\ (v, binomial(n = v, p = 1)), & \text{when the expected distribution} \\ & \text{is a binomial distribution delta function and } v \in \mathbb{N} \\ \perp, & \text{otherwise} \end{cases}$$

2 For example, casting 30 and 6.7 would result in:

- 3 • $cast(30) := (v, Dirac(1, 30))$
- 4 • $cast(30) := (v, binomial(30, 1))$
- 5 • $cast(6.7) := (v, Dirac(1, 6.7))$

6 Finally, the operator can be applied between two uncertain numbers if and
7 only if a mapping or an approximation between the source and target probability
8 distributions exists in probability theory:

$$cast(v, p_1) := \begin{cases} (v, approximation(p_1)), & \text{if the approximation exists} \\ \perp, & \text{otherwise} \end{cases}$$

9 Following the casting rules described in Table 6.2, u_{n1} can be cast into an
10 uncertain number with a Gaussian distribution and u_{n3} can be cast into an uncertain
11 number with a binomial distribution:

- 12 • $cast(u_{n1}) = (7, Gaussian(7.4, 4.662))$,
- 13 • $cast(u_{n3}) = (12, binomial(48, 0.25))$.

14 **Operator 4 - Dot operator.** For uncertain numbers, this operator has the same
15 semantics as described in Operator 4:

$$(v, p_d).value := v$$

16

$$(v, p_d).confidence := p_d$$

1 When used on u_{n1} and u_{n3} , this operator returns:

From \ To	Gaussian	Rayleigh	binomial	Dirac	certain number
Gaussian			✓		
Rayleigh					
binomial	✓			✓	✓
Dirac.					✓
certain nb.	✓	✓	✓	✓	

From: source type of uncertain number; To: targetted type. For readability purpose, we put the distribution names to represent the different types of uncertain numbers

Table 6.2: Cast operations allowed in our language

- $u_{n1}.value := 7$,
- $u_{n1}.confidence := binomial(20, 0.37)$,
- $u_{n3}.value := 12$,
- $u_{n3}.confidence := Gaussian(12, 9)$.

Operator 8 Uncertain arithmetic operators $(+, -, *, /)$ compute the arithmetic operation for uncertain numbers U_N : $U_N^2 \xrightarrow{u\text{-arithmetic}} U_N$

When performing arithmetic operations on uncertain variables, both values and probability distributions are considered. Two strategies to perform arithmetic operations on uncertain numbers are identified: numerical and analytical. While the second strategy is used in simple expressions, the first strategy is used when the expression is complex by returning an approximation of the arithmetic expression. Our language implements only the analytical method. Therefore, any semantics that requires the execution of a numerical method is considered undefined. If the second one is required, we consider the operator undefined. In Section 6.1.5, we detail the allowed operations in our language.

As with boolean expressions, the independence and joint of the two combined variables impact on the formula used to effectuate an arithmetic operation. Calculations are more complex when they are dependent. For example, although the addition of two independent Gaussian is done by adding the mean and the variance of the two distributions, the covariance matrix must be calculated when they are dependent. In such cases, the numerical approach is used.

Arithmetic operations are applied on both elements that define an uncertain number, the value and the probability distribution:

$$\begin{aligned}(v_1, p_1) + (v_2, p_2) &= (v_1 + v_2, p_1 + p_2) \\ (v_1, p_1) - (v_2, p_2) &= (v_1 - v_2, p_1 - p_2) \\ (v_1, p_1) * (v_2, p_2) &= (v_1 * v_2, p_1 * p_2) \\ (v_1, p_1) / (v_2, p_2) &= (v_1 / v_2, p_1 / p_2)\end{aligned}$$

For example, adding u_{n3} and u_{n4} will result in: $u_{n3} + u_{n4} := (12 + 23, \text{Gaussian}(12 + 23, 5 + 9)) := (35, \text{Gaussian}(35, 14))$.

Operator 9 Inequality operators ($<$, \leq , $>$, \geq) computes the confidence that the left side value is (strictly) less or greater than the right one : $(U_N \times N)^2 \xrightarrow{\text{inequality}} U_B$

When the inequality operator is used between an uncertain and a certain number, we compute the confidence that the left-hand side is greater than or equal to the right-hand side:

$$(v, p_d) \prec a := (c > 0, B(c)); c := P(u_n \prec a); \prec \in \{<, >, \leq, \geq\}$$

a , a numeric number in the same set as v

For example, comparing u_{n1} and u_{n3} will return:

- $u_{n1} > 10 := (0.17 > 0, B(0.17)) := (TRUE, B(0.17))$,
- $u_{n3} \leq 10 := (0.25 > 0, B(0.25)) := (TRUE, B(0.25))$.

When the inequality operator is used between two uncertain numbers, the probability variable is substituted by $u_{n1} \prec u_{n2} \Leftrightarrow u_{n1} - u_{n2}$. From there, we can apply the operator, \prec , between the result and 0. An operation can be executed, if and only if the subtraction is defined between the two operands.

$$(v_1, p_1) \prec (v_2, p_2) := [(v_1, p_1) - (v_2, p_2)] \prec 0; \prec \in \{<, >, \leq, \geq\}$$

For example, comparing u_{n3} and u_{n4} will result in: $u_{n3} \geq u_{n4} := u_{n3} - u_{n4} \geq 0 := (-11, \text{Gaussian}(-11, 4)) \geq 0 := (FALSE, B(0.00))$.

Operator 5 - Uncertain equality operator.

When the equality operator is applied to a discrete uncertain and a certain number, it returns the probability that the uncertain number equals (or not) the certain one:

$$(v_1, p_1) \prec a := (c > 0, B(c)); c := P(u_n \prec a); \prec \in \{=, \neq\}$$

For example, applying these operators on u_{n1} or u_{n2} will result in:

- $u_{n1} = 2 := (0.00 > 0, B(0.00)) := (FALSE, B(0.00))$,
- $u_{n2} \neq 13 := (0.89 > 0, B(0.89)) := (TRUE, B(0.89))$.

As it is impossible to compute the confidence of a precise number, this operation cannot be performed for continuous uncertain numbers:

$$(v_1, p_1) \prec a := \perp$$

To compare two discrete uncertain numbers, we use the same strategy as for the inequality operator: we subtract both and compare it with zero.

$$(v_1, p_1) \prec (v_2, p_2) := (v_1, p_1) - (v_2, p_2) \prec 0; \prec \in \{=, \neq\}$$

As this comparison is impossible with continuous uncertain numbers, this operator is undefined for such:

$$(v_1, p_1) \prec (v_2, p_2) := \perp$$

Operator 6 - Identity operator. Similarly to uncertain booleans, the identity operator ($==$) applied to uncertain numbers returns true only if both values and both probability distributions are equal:

$$(v_1, p_1) = (v_2, p_2) := v_1 = v_2 \& \& p_1 = p_2$$

The unequal operator (\neq) returns true if both values or distributions are unequal:

$$(v_1, p_1) \neq (v_2, p_2) := v_1 \neq v_2 \vee p_1 \neq p_2$$

2 6.1.4 Uncertain references

3 What is an uncertain reference?

4 References, or pointers, allow storing a directed association from one element
5 (*e.g.*, a Java object) to another one. **A reference is defined as uncertain when**
6 **the existence of this relation is not known with the most thorough**
7 **confidence.** For example, to represent the topology of the smart grid we can
8 imagine substituting cables by simple references from one entity to another. These
9 references are not known with absolute confidence due to the uncertainty of fuses
10 states.

11 Mapping to uncertain booleans

12 Like for uncertain booleans, uncertain references have two states: either the
13 reference exists or not. The confidence level on the existence can thus be represented
14 by an uncertain boolean. We map the *TRUE* value to the existence of the reference
15 and the *FALSE* one to its non-existence. Internally, uncertain references are
16 represented by two components: a reference value and an uncertain boolean.

$$(Reference, c) \mapsto (Reference, (v, B(p = c))), v \in \{TRUE, FALSE\}$$

17 The abstract syntax of our language is thus extended to add a new data type:
18 uncertain reference. The semantics of the existence, confidence, cast, and dot
19 operator are extended to consider this new type.

20 Operational semantics

21 We denote by (r, u_b) an uncertain reference with a value r and an uncertain
22 boolean u_b , which represents its uncertainty. The following two uncertain references
23 exemplify this:

- 24 • $u_{r1} := (r_1, u_{b1}), u_{b1} := (TRUE, B(0.88)),$
- 25 • $u_{r2} := (r_2, u_{b2}), u_{b2} := (TRUE, B(0.34)).$

26 **Operator 1 - Existence operator.** The semantics of this operator is defined
1 upon the existence operator of the uncertain boolean that represents the uncertainty:

$$exists((r, u_b), t) := exists(u_b, t)$$

When applied on the two examples from above, this operator would return:

- $exists(u_{r1}, 0.8) := exists(u_{b1}, 0.8) := TRUE$,
- $exists(u_{r2}, 0.8) := exists(u_{b2}, 0.8) := FALSE$.

Operator 2 - Confidence operator. This operator relies on the confidence operator of the uncertain boolean. If this operation tells that the most confident value given a base is *TRUE*, then the existence operator returns the reference when applied on an uncertain reference:

$$confidence((r, u_b), t) := \begin{cases} r, & \text{when } confidence((u_b), t) = TRUE, \\ \perp, & \text{otherwise} \end{cases}$$

If applied to the two examples, it will return:

- $confidence(u_{r1}, 0.8) := r_1$,
- $confidence(u_{r2}, 0.8) := \perp$.

Operator 3 - Cast operator.

When casting an uncertain reference to a certain one, it will return the reference value only if its confidence level is strictly superior to zero:

$$cast(r, u_b) := confidence((r, u_b), 0)$$

Casting both of our results into certain references will thus return in:

- $cast(u_{r1}) := confidence(u_{r1}, 0) := r_1$,
- $cast(u_{r2}) := confidence(u_{r2}, 0) := r_2$.

When casting a reference to an uncertain one, the confidence level of its existence is set at the maximum, 1:

$$cast(r) := (r, (TRUE, 1))$$

For example, casting a reference r_3 to an uncertain one returns: $cast(r_3) := (r_3, (TRUE, 1))$.

Operator 4 - Dot operator.

	Gaussian	Rayleigh	binomial	Dirac	certain nb.
Gaussian	Gaussian	X	Gaussian	Gaussian	Gaussian
Rayleigh	X	Rayleigh	X	Rayleigh	Rayleigh
binomial	Gaussian	X	binomial	binomial	binomial
Dirac	Gaussian	Rayleigh	binomial	Dirac	Dirac
certain nb.	Gaussian	Rayleigh	binomial	Dirac	certain nb.

Table 6.3: Typing rules for arithmetic operations

The dot operator allows accessing the different elements of the uncertain reference:

$$(r, u_b).value := r$$

$$(r, u_b).confidence := u_b$$

2 Applying this operator on u_{r1} thus returns:

- 3 • $u_{r1}.value(u_{r1}) := r_1$,
- 4 • $u_{r1}.confidence(u_{r1}) := u_{b1}$.

5 6.1.5 Static semantic: typing rules

6 This section introduces the particularity of the type-system of our language
7 compared to those without support for uncertainty. In particular, we stress two
8 points. First, the typing of implicit casts between uncertain and certain data.
9 Second, the typing and interactions between different data types when applying
10 arithmetic operations.

11 **Implicit casts** In our language, we enable implicit casts. In addition to the
12 casting operations shown in Table 6.2, our proposed type system enables casts
13 between uncertain and certain booleans.

14 **Typing rules for arithmetic operations** Probability theory defines how two
15 probability distributions can be combined and what the result should be, when
16 possible. We follow these rules to define the result of the arithmetic operations
17 between uncertain numbers. Below, Table 6.3 depicts the results of each operation
18 between two uncertain numbers.

19 When an arithmetic operation is applied to uncertain values with two different
1 probability distributions, the resulting distribution may follow a well-defined distri-

bution or not. In case the resulting distribution is different from the distributions supported by our language, we consider it as undefined. For example, performing an arithmetic operation between a Gaussian and a Rayleigh distribution is not permitted since it does not result in a well-defined distribution and should be calculated by applying the convolution rule.

For the implemented distributions, all operations between two numbers represented by the same distribution, result in an uncertain number with the same distribution. For example, an addition between two uncertain numbers with a Gaussian distribution returns another one with a Gaussian distribution. One exception: combining two binomial distributions result in another one only if the parameter p is identical in both cases. If p is different, the resulting distribution is a Poisson distribution, not implemented in our language. Hence, we consider it as undefined.

An operation between a probability distribution and a scalar results in this probability distribution, shifted (for addition or subtraction) or with a coefficient applied (for multiplication or division). We perform these operations on the probability distribution when an uncertain number is combined with a certain one.

A Gaussian distribution can be approximated by a binomial one and vice-versa. When these two are being combined, the type of the final result can be one of these two. There are thus two possible choices: either we cast both into uncertain numbers with a Gaussian distribution or both into uncertain numbers with a binomial distribution. By default, we decide to opt for the first option. Therefore, the arithmetic operator returns an uncertain number with a Gaussian distribution.

6.2 Evaluation

We validated our approach by implementing a language named Ain'tea, which features an uncertainty-aware static type system and provides the concepts and operators presented in Section 6.1. We evaluate the following two research questions based on our implementation of Ain'tea:

- **RQ1:** Does native uncertainty management on the language level have an impact on the conciseness of a language, *i.e.*, its ability to express concisely statements?
- **RQ2:** Can the type system detect errors related to uncertainty management?

2 To address them, we compared our implementation to two state-of-the-art frame-
3 works: Infer.NET [MWG⁺18] and OpenTURNS [BDI⁺17]. The code used for the
4 experiments of the validation is available on the repository of our implementation.

5 In this section, we first detail the implementation of the Ain'tea language. Then,
6 we show that our solution can detect type errors earlier than the two solutions we
7 compare our work to. Before discussing the results, we compare the number of lines
8 of codes required for the different solutions to perform uncertainty propagation.

9 6.2.1 Ain'tea: our implementation

10 We use the Xtext language workbench² to define the concrete syntax and the
11 type checker of our language (static semantics). Based on the concrete syntax,
12 Xtext generates the abstract one. We implement the dynamic semantics with K3³.

13 In addition, we implemented different code samples which shows the different
14 features of our language. We also developed a simplified version of our use case
15 example, introduced in Chapter 3. We presented it as a tutorial of our language. All
16 code can be executed using a runner, available on the project repository.

17 Overview of the language

18 Ain'tea supports classes, with fields and functions. It has an expression language
19 with arithmetic and boolean expressions, *IF*-conditions and affectation statements.

20 Ain'tea is composed of four elements: an abstract syntax, a concrete syntax,
21 static and dynamic semantics. As depicted in Figure 6.2, the concrete syntax
22 has been implemented using Xtext. As we used a grammar-first approach, the
23 abstract syntax has been automatically generated from the concrete one. In
24 addition, stub-classes are generated for the semantics (static and dynamic). Using
25 Xtend, we implement the static semantics. The dynamic semantics is defined as an
26 operational semantics and implemented using K3 [JCB⁺15]. The implementation
27 of our uncertainty-aware type checker and semantic uses two strategies. For
28 simple operations (*e.g.* operator semantics for uncertain booleans), we implement
29 them using K3. For more complex operations, like the computation of integral or
1 combining different distributions, we use a third party library: Apache Commons

²<http://www.eclipse.org/Xtext/>

³<http://diverse-project.github.io/k3/>

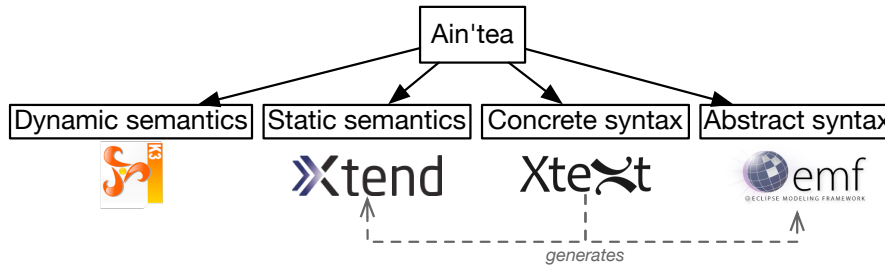


Figure 6.2: Global architecture of Ain'tea

2 Math Library⁴.

3 Finally, by using the Xtext language workbench, other important features
4 have been generated for our language. Among them, there is the error maker,
5 auto-completion and an Eclipse plugin.

6 Syntax

7 The syntax of the language has been inspired by Java: keywords to define
8 classes, fields and functions are similar. Regarding the uncertain operators defined
9 in Section 6.1, we use the same syntax as for certain operators where ever possible.
10 We give an overview of their corresponding syntax in the following list:

- 11 • confidence: []
- 12 • existence: **exist**
- 13 • cast: **as**
- 14 • dot: .
- 15 • inequality: >, >=, <, <=
- 16 • (uncertain) equality: ==, !=
- 17 • identity: ===, !==
- 18 • uncertain arithmetic: +, -, *, /
- 19 • uncertain boolean: &&, ||, !

20 Inequality, arithmetic and boolean operators have identical syntax for uncertain
21 and certain data. By making this choice, combining uncertain or certain data
22 is done with the same syntax. For example, adding two uncertain numbers is
23 performed by the following syntax: $u_{N1} + u_{N2}$.

1 When the equality operator (==, !=) is used with at least one uncertain data

⁴<http://commons.apache.org/proper/commons-math/>

2 type, we apply the uncertain equality operator. For example, the following code
3 $u_{N1} == u_{N2}$ returns the confidence that both values are equal. To use the identity
4 operator, we use the following syntax: $===$ and $!==$. For example, this code
5 $u_{N1} === u_{N2}$ checks if both uncertain numbers are identical (cf. Operator 6).

6 Following his name, the syntax of the dot operator is a dot ($.$). The value
7 can be accessed using the *value* keyword. For example, to get the value of an
8 uncertain boolean, one will write $u_b.value$. To get the probability distribution, the
9 *confidence* keyword has been introduced. For example, the Bernoulli distribution
10 of an uncertain boolean is resolved using the following syntax: $u_b.confidence$.

11 For the cast operator, we add the *as* keyword. For instance, to cast an uncertain
12 number to a certain one, one will do $u_n \text{ as } Number$. Here, *Number* represents any
13 number type of our language: short, integer, long, float and double.

14 An internal method is used to call the existence operator: *exist*. For example, in
15 order to check if a value of an uncertain boolean exists with at least 0.8 confidence,
16 one needs to add the following line in its code: $exist(u_B, 0.8)$.

17 Finally, the confidence operator is called using square brackets: $[]$. For example,
18 to get the most confidence value of an uncertain number with at least 0.7 as
19 confidence, one will write: $u_N[0.7]$

20 The type of uncertain data can be thought of as a pair of two types: one for
21 the raw data and one for the uncertainty representation. We decide to use a syntax
22 similar to the one used in Java to declare generic types: $Type1<Type2>$. We
23 arbitrarily choose to put the probability distribution for the first type and the type
24 of the raw data point for the second. For example, the declaration of an uncertain
25 boolean is done using the following syntax: $Bernoulli<bool>$.

26 **Dependency and joint detection**

27 As we explained in Section 6.1, dependency and joint between two variables
28 impact the semantics of the different operators. In our implementation, we first
29 consider that any variable initialized by a constant is independent. Then, we
30 compute a dependency graph between the different variables. However, we use an
31 algorithm which covers only the simple cases.

32 Concerning the joint, we also apply a simple and naive algorithm to compute
1 the domain covered by boolean variables. But, this is only performed on boolean

resulting from comparison operators with one static constant. We let for future work the implementation of an algorithm that applies the same rules at runtime. Below, an example of what our implementation can do.

Let us imagine three uncertain booleans b_1 , b_2 , and b_3 defined as follow, with u_n an uncertain number:

- $b_1 := u_n \geq 10$
- $b_2 := u_n < 15$
- $b_3 := u_n > 20$

From this definition, b_1 is defined on the range $[10, +\infty)$, b_2 on $(-\infty, 15)$, and b_3 on $(20, +\infty)$. b_1 and b_2 are thus non-disjoint, like b_1 and b_3 . Indeed, b_1 and b_2 are both defined on the range $[10, 15)$, and b_1 and b_3 on $(20, +\infty)$. However, b_2 and b_3 are disjoint as it does not exist any common range between their definition.

6.2.2 Conciseness

Regarding the syntax, we distinguish three kinds of operations which manipulate data: writing⁵, reading, and combination. For these three kinds of operations, we compare the number of lines of codes required by our language and the two frameworks. To perform this, we implemented a simplified version of our case study using these three solutions. Excerpts of the source code are shown in Listing 6.1-6.3.

Writing and reading operations remain similar to what exists in many other programming languages. There is no difference between the three solutions to this aspect. In the listings, we can notice that all lines of code relative to it have the same size: line 6, 11, and 20 in Listing 6.1; line 7, 11-12, and 21 in Listing 6.2; and line 9, 16, and 27 in Listing 6.3.

Both languages used for developing the frameworks (C#⁶ and Python⁷) allow overloading operators. As highlighted in the listings, the three solutions have the same syntax to combine (through arithmetic or boolean operators) uncertain and certain data: lines 9 and 23 in Listing 6.1; lines 10 and 23 in Listing 6.2; and lines 13 and 34 in Listing 6.3.

⁵This operation also includes the creation and the deletion. The first one corresponds to the first write operation and the last one is done by setting the value to *NULL*.

⁶<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/overloadable-operators>

⁷<https://docs.python.org/3/reference/datamodel.html#special-method-names>

2 In contrary to OpenTurns and our solution, Infer.NET requires an explicit
3 call to the engine which computes and propagates the probabilities. We can see
4 these calls in line 10, 16 and 22 in Listing 6.1. This framework is mainly done for
5 probabilistic programming, where developers implement a model, then executes
6 it. From our understanding of this tool, it was not designed to allow an iterative
7 propagation of uncertainty and a control flow that depends on this propagation.
8 Thus, we need to split the propagation in different models and explicitly call their
9 inference engine to enable such a process.

10 Finally, all three solutions allow reasoning on uncertain data. In Listing 6.1 at
11 line 17, Listing 6.2 at line 18, and Listing 6.3 at line 24, we highlight an IF-condition
12 based on the confidence of an uncertain boolean⁸. Infer.NET provides a method to
13 help manipulate uncertain booleans, by providing methods to access the probability
14 of the *TRUE* and *FALSE* value. In this case, our confidence operator can be
15 thought of as a sugar syntax of these methods.

```

16 public void ComputeLoadNoCable(Substation substation) {
17     Variable<bool> noCableConn = Variable.Bernoulli(1);
18     if(substation.Fuses.Count > 0)
19         noCableConn = substation.Fuses[0].IsClosed;
20     else
21         substation.Load = GaussianFromMeanAndVariance(0, 0.001);
22     return;
23     for(int i=1; i<substation.Fuses.Count; i++)
24         noCableConn = noCableConn & substation.Fuses[i].IsClosed;
25     Bernoulli bernoulli = (Bernoulli) InferenceEngine.Infer(noCableConn);
26     substation.Load = GaussianFromMeanAndVariance(0, bernoulli.GetVariance());
27 }
28
29
30 public void ComputeLoad(Substation substation) {
31     Variable<bool> isDisco = substation.IsDisconnected();
32     Bernoulli bern = (Bernoulli) InferenceEngine.Infer(isDisco);
33     if(bern.GetProbTrue() >= 0.95)
34         ComputeLoadNoCable(substation);
35     return;
36     Variable<double> load = GaussianFromMeanAndVariance(0, 0.001);
37     foreach(Fuse fuse in substation.Fuses)
38         Bernoulli isClosedBern = (Bernoulli) InferenceEngine.Infer(fuse.IsClosed);
39         load = load + (fuse.Cable.Load * isClosedBern.GetProbTrue());
40     substation.Load = load;

```

⁸As OpenTurns does not include an uncertain boolean implementation, we implemented one similar to the one we provide in Ain'tea, using their Bernoulli implementation. The implementation is accessible in our GitHub repository

Listing 6.1: Excerpt of the Infer.NET implementation (C#)

```

4  def compute_load_no_cable(substation):
5      if not isinstance(substation, Substation):
6          raise TypeError('Wrong type')
7      if len(substation.fuses) > 0:
8          no_cable_conn = substation.fuses[0].isClosed
9      else:
10         substitution.load = ot.Normal(0, 0.001)
11         return
12     for fuse in substation.fuses[1:]:
13         no_cable_conn = no_cable_conn & fuse.isClosed
14     substitution.load = ot.Normal(0,
15         no_cable_conn.getStandardDeviation()*no_cable_conn.getStandardDeviation())
16
17 def compute_load(substation):
18     if not isinstance(substation, Substation):
19         raise TypeError('Wrong type')
20     is_disco = substation.is_disconnected()
21     if is_disco.exist(0.95) and is_disco.value_with_confidence(0.95):
22         compute_load_no_cable(substation)
23         return
24     load = ot.Normal(0, 0.001)
25     for fuse in substation.fuses:
26         load = load + (fuse.cable.load*fuse.isClosed.confidence.getP())
27     substitution.load = load
28
29

```

Listing 6.2: Excerpt of the OpenTurns implementation (Python)

```

30 void ComputeLoadNoCable(Substation substation) {
31     Bernoulli<bool> noCableConn = new Bernoulli<bool>(true, 1);
32     Fuse[] fuses = substation.fuses;
33     Fuse f;
34     if(fuses.length > 0)
35         f = fuses[0];
36         noCableConn = f.isClosed;
37     else
38         substitution.load = new Gaussian<double>(0, 0.001);
39         return;
40     for(int i=1; i<fuses.length; i=i+1)
41         f = fuses[i];
42         noCableConn = noCableConn && f.isClosed;
43     Bernoulli bern = noCableConn.confidence;
44     if(bern.probability >= 0.97) {
45         substitution.load = new Gaussian<double>(0, 0.0289);
46     } else {
47         [...]
48

```

2	}	19
3	}	20
4		21
5	void ComputeLoad(Substation substation) {	22
6	Bernoulli<bool> isDisco = substation.isDisconnected();	23
7	if(exist(isDisco, 0.95) && isDisco[0.95])	24
8	ComputeLoadNoCable(substation);	25
9	return;	26
10	Gaussian<double> load = new Gaussian<double>(0, 0.001);	27
11	Fuse[] fuses = substation.fuses;	28
12	for(int i=0; i<fuses.length; i=i+1)	29
13	Fuse f = fuses[i];	30
14	Cable c = f.cable;	31
15	Bernoulli<bool> isClosed = f.isClosed;	32
16	Bernoulli bern = isClosed.confidence;	33
17	load = load + c.load * bern.probability;	34
18	substation.load = load;	35
19	}	36

Listing 6.3: Excerpt of the Ain'tea implementation

RQ1 aims at evaluating the impact of native uncertainty management on the conciseness of a language. As shown by the above evaluation, adding uncertainty as a first-class citizen can be done without damaging the conciseness. As we have seen through this chapter, managing uncertainty impacts the semantics of the language, by complexifying the traditional operators (arithmetic, boolean, comparison). A few operators can also be added in order to enable reasoning over the uncertainty. The impact on the concrete syntax is thus limited.

However, developers will use the same syntax to manipulate more complex concepts. Adding two uncertain numbers and two numbers is semantically different. One threat to validity is the lack of impact assessment on developers, who may have difficulty manipulating these concepts. However, as we hide this complexity behind traditional operators, the risk is rather low.

6.2.3 Error handling at development time

Among the different root causes of typing errors, one is code refactoring. In order to validate our approach, we thus define a scenario based on it.

Let's assume a developer implements the load approximation for a smart grid. She/he must consider different scenarios that can exist from a simple cable to a more complex situation which looks like a small graph. As the code is complex, in

```

class Substation {
    Fuse[] fuses;
    Gaussian<double> load;
    Cable[] getCables() {}
}
void computeLoadNoCable(Substation s) {
    Bernoulli<bool> noCableConn;
    Fuse[] fuses = s.fuses;
    if(fuses.length > 0) {
        Fuse f = fuses[0];
        noCableConn = f.isClosed;
    } else {
        s.load= new DiracDeltaFct<double>(0, 1);
    }
}
for (int i=1; i<fuses.length; i=i+1) {

```

DiracDeltaFunctionDouble is not compatible with GaussianDouble. Compatible types are: [ainte.lang.BinomialType, ainte.lang.GaussianType]

Figure 6.3: Detection of a type error

one place (*e.g.*, a file) she/he defines the smart grid classes and she/he implements this computation in another one. At first, the developer decides to use the Dirac delta function to represent the uncertainty of the substation load. During the development process, she/he notices that the Gaussian one suit more his/her problem. However, all the operations are not compatible with this new type, for example when there is no cable connected.

As we depict in Figure 6.3, our approach can detect this type of error statically. Plus, we can also notice that we can help developers fix this kind of issue by listing the compatible types with the new substation type (*Gaussian<double>*).

Among the two selected framework, only Infer.NET is implemented in a language which supports static type checking: C#⁹. We also implement a simplified example that raises a typing error. To do so, we perform an addition between two incompatible probability distributions: a Gaussian and a Gamma¹⁰. As we show in Figure 6.4, the error is only detected at runtime.

In addition, we see in Figure 6.3 that by specifying the type system, the language is able to provide high-level explanations of the errors. Here, it explains that the two distributions are not compatible plus gives the list of compatible distributions with the field type. In contrary, in Figure 6.4, the error is detected later than its root cause and informs that the operation is not supported. Indeed, as depicted in Figure 6.4a, the error of line 19 is detected at line 21 (highlighted in green) without any reference to this operation. Plus, the error message detailed all type

⁹<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/index>

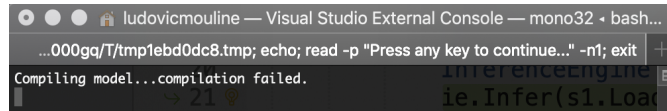
¹⁰The Rayleigh distribution is not present in the framework. We thus choose another one.

```

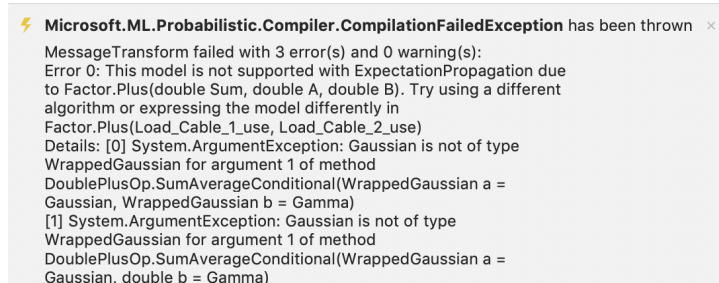
10 Substation s1 = new Substation();
11 Cable c1 = new Cable
12 {
13     Load = Variable.GaussianFromMeanAndVariance(10, 0.5).Named("Load_Cable_1")
14 };
15 Cable c2 = new Cable
16 {
17     Load = Variable.GammaFromMeanAndVariance(10, 0.5).Named("Load_Cable_2")
18 };
19 s1.Load = (c1.Load + c2.Load).Named("Load_Substation_1");
20 InferenceEngine ie = new InferenceEngine();
21 ie.Infer(s1.Load);

```

(a) Code with a type error, detected at the green highlighted line



(b) Console output



(c) Excerpt of the error message

Figure 6.4: Infer.NET detect the error at runtime.

2 mismatches between the declaration of the method and the given types.

3 This evaluation validates the ability of a type system to detect errors related
4 to uncertainty management (RQ2). When a type system is implemented, error
5 messages are defined to help developers. And so, support to develop uncertainty-
6 aware software can be provided.

7 6.2.4 Discussion

8 By mapping arithmetic and boolean operators to the propagation of uncertainty,
9 we hide the complexity of the combination of probability distributions. It helps
10 developers to stay in the paradigm that they use every day, introducing as few
11 concepts as possible. This being said, as we intend to have a language that
12 manipulates several kinds of uncertainties, they still need a high-level knowledge
1 of probability distributions. For example, they should understand the difference

between a Gaussian and a Rayleigh. But they do not need to know if or how they can be combined. Indeed, thanks to our type checker, the language will provide information or raise errors when it is not possible.

However, if the uncertainty of a domain can be represented using only one probability distribution, thus this distribution could also be hidden for the developer. Therefore, they will only manipulate uncertain and certain data. For example, using the approach introduced in [MWV16], a developer will only manipulate *UReal* and *Real* for uncertain and certain numbers.

By hiding the complexity of the combination of probability distributions, we also hide the computation overhead introduced. This work has not been done with performance as a goal. We do not guarantee any performance of our language. The evaluation lacks a quantification of this overhead.

6.3 Conclusion

Data are inherently uncertain. This uncertainty can modify the understanding of the elements represented by these data. Furthermore, it may damage the decision made upon this knowledge. Managing this uncertainty calls for a strong understanding of probability theory. However, it can be far from developers expertise. In order to help them manipulating this uncertainty, we define Ain'tea, a language which integrates concepts related to data uncertainty. Mainly, we define uncertain data types and we map uncertainty propagation to arithmetic, boolean, and comparison operators. In addition, we define operators specific to the manipulation of the uncertainty representation. In our validation, we show that our language is as concise as state-of-the-art solutions. Contrary to these solutions, we also show that our solution can detect errors earlier. Thanks to the semantics, which supports uncertainty, errors message help developers in their development of algorithms that use uncertain data.

A temporal knowledge metamodel of adaptive systems

Contents

6	7.1 Knowledge formalization	118
7		
8	7.2 Modelling the knowledge	128
9	7.3 Validation	133
10	7.4 Conclusion	144
11		
12		
13		

In this chapter, we first propose a knowledge formalism to define the concept of a decision. Second, we describe a novel temporal knowledge model to represent, store and query decisions as well as their relationship with the knowledge (context, requirements, and actions). We validate our approach through a use case based on the smart grid at Luxembourg. We also demonstrate its scalability both in terms of execution time and consumed memory.

2 To address Sub-challenge #2 and Sub-challenge #3 (*cf.* Section 1.3), we propose
3 a temporal knowledge model which can trace decisions over time, along with their
4 circumstances and effects. By storing the decisions, the adaptation process could
5 consider ongoing actions with their expected effects, also called impacts. Besides,
6 in case of faulty decisions, developers may trace back effects to their circumstances.

7 Our approach allows structuring and storing the state and behaviour of a
8 running adaptive system, together with a high-level API to efficiently perform
9 diagnosis routines. Our framework relies on a temporal model-based solution that
10 efficiently abstracts decisions and their corresponding circumstances. Specifically,
11 based on existing approaches for modelling and monitoring adaptation processes,
12 we identify a set of properties that characterise context, requirements, and actions
13 in self-adaptive systems. Then, we formalise the common core concepts implied in
14 adaptation processes, also referred to as knowledge, by means of temporal graphs
15 and a set of relations that trace decisions' impact to circumstances. Finally, thanks
16 to exposing common interfaces in adaptation processes, existing approaches in
17 requirements and goal modelling engineering can be easily integrated into our
18 framework.

19 The rest of this chapter is structured as follows. In the remaining part of this
20 section, we motivate our approach, we summarise core concepts manipulated in
21 adaptation processes, and we present a use case scenario based on the Luxembourg
22 Smart Grid (*cf.* Chapter 3). Then, we provide a formal definition of these concepts
23 in Section 7.1. After, we describe the proposed data model in Section 7.2. In
24 Section 7.3, we demonstrate the applicability of our approach by applying it to the
25 smart grid example. We conclude this chapter in Section 7.4.

26 **7.1 Knowledge formalization**

27 We consider knowledge to be the association of context information, require-
28 ments, and action information, all in one global and unified model. While context
29 information captures the state of the system environment and its surroundings, the
30 system requirements define the constraints that the system should satisfy along
31 the way. Actions, on the other hand, are meant to reach the goals of the system.

32 In this section, we provide a formalization of the knowledge used by adaptation
1 processes based on a temporal graph. Due to the complexity and interconnectivity

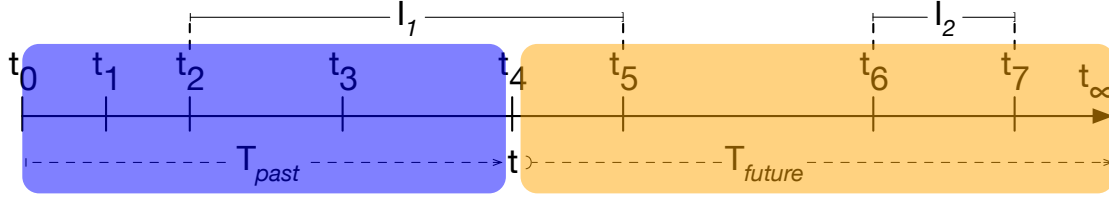


Figure 7.1: Time definition used for the knowledge formalism

of system entities, graph data representation is an appropriate way to represent the knowledge. Augmented with a temporal dimension, temporal graphs are then able to symbolize the evolution of system entities and states over time. We benefit from the well-defined graph manipulation operations, namely temporal graph pattern matching and temporal graph relations to represent the traceability links between the decisions made and their circumstances.

Before describing this formalism, we describe the semantics used for the temporal axis. Then, we exemplify the knowledge formalism using the Luxembourg smart grid use case, detailed in Section 3.3.2.

7.1.1 Formalization of the temporal axis

The formalism described below has been defined with two goals in mind. First, the definition of the time space should allow the distinction between past and future. Making this distinction enables the differentiation between measured data and predicted (or planned data). Second, it should permit the definition of the life cycle of an element of the knowledge, which can be seen as a succession of states with a validity period that should not overlap each other.

Time space T is considered as an ordered discrete set of time points non-uniformly distributed. As depicted in Figure 7.1, this set can be divided into 3 different subsets $T = T_{past} \cup \{t\} \cup T_{future}$, where:

- T_{past} is the subdomain $\{t_0; t_1; \dots; t_{current-1}\}$ representing graph data history starting from t_0 , the oldest point, until the current time, t , excluded.
- $\{t\}$ is a singleton representing the current time point
- T_{future} is subdomain $\{t_{current+1}; \dots; t_\infty\}$ representing future time points

The three domains depend completely on the current time $\{t\}$ as these subsets slide as time passes. At any point in time, these domains never overlap: $T_{past} \cap \{t\} = \emptyset$,

$T_{future} \cap \{t\} = \emptyset$, and $T_{past} \cap T_{future} = \emptyset$. The definition of these three subsets reaches the first goal.

In addition, there is a right-opened time interval $I \in T \times T$ as $[t_s, t_e)$ where $t_e - t_s > 0$. In English words, it means that the interval should represent at least one time point and should follow the time order. For any $i \in I$, $start(i)$ denotes its lower bound and $end(i)$ its upper bound. As detailed in Section 7.1.2, these intervals are used to define the validity period for each node of the graph (our second goal).

Figure 7.1 displays an example of a time space $T_1 = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$. In this case, the current time is $t = t_4$. According to the definition of the past subset (T_{past}) and the future one (T_{future}), there is: $T_{past1} = \{t_0, t_1, t_2, t_3\}$ and $T_{future1} = \{t_5, t_6, t_7\}$. Two intervals have been defined on T_1 , namely I_1 and I_2 . The first one starts at t_2 and ends at t_5 and the last one is defined from t_6 to t_7 . As shown with I_1 , an interval could be defined on different subsets, in this case it is on all of them (T_{past} , t , and T_{future}).

7.1.2 Formalism of the knowledge

Graph definition First, let K be an adaptation process over a system knowledge represented by a graph such as $K = (N, E)$, comprising a set of nodes N and a set of edges E . Nodes represent any element of the knowledge (context, actions, etc.) and edges represent their relationships. Nodes have a set of attribute values: $\forall n \in N, n = (id, P)$, where P is the set of key-value attributes. An attribute value has a type (numerical, boolean, ...). Every relationship $e \in E$ can be considered as a couple of nodes with a label $(n_s, n_t, label) \in N \times N$, where n_s is the source node and n_t is the target node.

Adding the temporal dimension In order to augment the graph with a temporal dimension, the relation V^T is added. So now the knowledge K is defined as a temporal graph such as $K = (N, E, V^T)$.

A node is considered valid either until it is removed or until one of its attributes value changes. In the latter case, a new node with the updated value is created. Whilst, an edge is considered valid until either its source node and target node are valid, or until the edge itself is removed. Otherwise, nodes and edges are considered invalid. The temporal validity relation is defined as $V^T : N \cup E \rightarrow I$. It takes as a

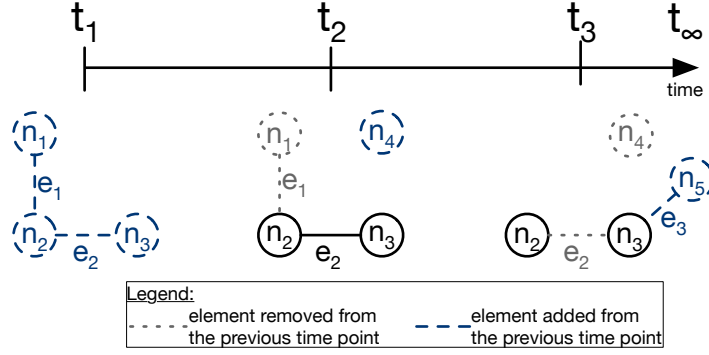


Figure 7.2: Evolution of a temporal graph over time

parameter a node or an edge ($k \in N \cup E$) and returns a time interval ($i \in I$, cf. Section 7.1.1) during which the graph element is valid.

Figure 7.2 shows an example of a temporal graph K_1 with five nodes (n_1, n_2, n_3, n_4 , and n_5) and three edges (e_1, e_2 , and e_3) over a lifecycle from t_1 to t_3 . In this way, K_1 equals $(\{n_1, n_2, n_3, n_4, n_5\}, \{e_1, e_2, e_3\}, V_1^T)$. Let's assume that the graph is created at t_1 . As n_1 is modified at t_2 , its validity period starts at t_1 and ends at t_2 : $V_1^T(n_1) = [t_1, t_2)$. n_2 and n_3 are not modified; their validity period thus starts at t_1 and ends at t_∞ : $V_1^T(n_2) = V_1^T(n_3) = [t_1, t_\infty)$. Regarding the edges, the first one, e_1 , is between n_1 and n_2 and the second one, e_2 from n_2 to n_3 . Both are created at t_1 . As n_1 is being modified at t_2 , its validity period goes from t_1 to t_2 : $V_1^T(e_1) = [t_1, t_2)$. e_2 is deleted at t_3 . Its validity period is thus equal to: $V_1^T(e_2) = [t_1, t_3)$.

Lifecycle of a knowledge element One node represents the state of exactly one knowledge element during a period named the validity period. The lifecycle of a knowledge element is thus modeled by a unique set of nodes. By definition, the validity periods of different nodes cannot overlap. A same time period cannot be represented by two different nodes, which could create inconsistency in the temporal graph.

To keep track of this knowledge element history, the Z^T relation is added to the graph formalism: $K = (N, E, V^T, Z^T)$. It serves to trace the updates of a given knowledge element at any point in time. This relation can also be seen as a temporal identity function which takes as parameters a given node $n \in N$ and a specific time point $t \in T$, and returns the corresponding node at that point.

2 Formally, $Z^T : N \times T \rightarrow N$.

3 In order to consider this new relation in the example presented in Figure 7.2,
4 the definition of K_1 is modified to $K_1 = (\{n_1, n_2, n_3, n_4, n_5\}, \{e_1, e_2, e_3\}, V_1^T, Z_1^T)$. In
5 Figure 7.2, let's imagine that n_1 , n_4 , and n_5 represent the same knowledge element
6 k_e . The lifecycle of k_e is thus:

- 7 • n_1 for period $[t_1, t_2)$,
- 8 • n_4 for period $[t_2, t_3)$,
- 9 • n_5 for period $[t_3, t_\infty)$.

10 Let t'_1 be a timepoint between t_1 and t_2 . When one wants to resolve the node
11 representing the knowledge element at t'_1 , she or he gets n_1 node, no matter of the
12 node input (n_1 , n_4 , or n_5): $Z_1^T(n_4, t_1) = n_1$. On the other hand, applying the same
13 relation with another node (n_2 or n_3) returns another node. For example, if n_2
14 and n_3 do not belong to the same knowledge element, then it will return the node
15 given as input, for example $Z_1^T(n_2, t_1) = n_2$.

16 **Knowledge elements stored in nodes** Nodes are used to store the different
17 knowledge elements: context, requirements and actions. The set of nodes N is thus
18 split in three subsets: $N = C \cup R \cup A$ where C is the set of nodes which store
19 context information, R a set of nodes for requirement information and A a set of
20 nodes for action information.

21 Actions define processes that indirectly impact the context: they will change
22 the behaviour of the system, which will be reflected in the context information.
23 Requirements are also processes that are continuously run over the system in order
24 to check the specifications. Here, the purpose of the A and R subset is not to store
25 these processes but to list them. It can be thought as a catalogue of actions and
26 requirements, with their history.

27 Using a high-level overview, these processes can be depicted as: taking the
28 knowledge as input, perform tasks, and modify this knowledge as output. As
29 detailed in the next two paragraphs, action executions and requirement analysis
30 can be formalized by relations.

31 **Temporal queries for requirements** At the current state, the formalism of
32 the knowledge K does not contain any information regarding the requirement
1 analysis. To overcome this, system requirements analysis R_A are added such as

$K = (N, E, V^T, Z^T, R_A)$. R_A is a set of couples composed of patterns $P_{[t_j, t_k]}(K)$ and requirements R over these patterns: $R_P = P \cup R$.

$P_{[t_j, t_k]}$ denotes a temporal graph pattern, where t_j and t_k are the lower and upper bound of the time interval respectively. $P_{[t_j, t_k]}$ is the result of a function which takes the knowledge and an interval as input: $P_{[t_j, t_k]} : K \times I$. The time interval can be either fixed (absolute), *i.e.*, both bounds are precisely defined, or sliding (relative), *i.e.*, the upper bound is computed from the lower bound. For example, $P_{[t_0, t_4]}$ is considered as fixed and $P_{[t_0, t_0+4]}$ is considered as relative. Each element of the pattern should be valid for at least one timepoint: $\forall p \in P_{[t_j, t_k]}, V^T(e) \cap [t_j, t_k] \neq \emptyset$. Patterns can be seen as temporal subgraphs of K , with a time limiting constraint coming in the form of a time interval.

Temporal relations for actions Like for R_A , the knowledge K needs to be augmented with action executions A_E : $K = (N, E, V^T, Z^T, R_A, A_E)$. Actions executions A_E can be regarded as a couple (A, A_F) , where A is the action that is executed and A_F a set of relations or isomorphisms mapping a source temporal graph pattern $P_{[t_j, t_k]}$ to a target one $P_{[t_l, t_m]}$, $A_F : K \times I \rightarrow K \times I$.

The left-hand side of the A_F relation depicts the temporal graph elements over which an action is applied. Every relation may have a set of application conditions. They describe the circumstances under which an action should take place. These application conditions are either positive, should hold, or negative, should not hold. Application conditions come in the form of temporal graph invariants. The side effects of these actions are represented by the right-hand side.

Finally, we associate to A_E a temporal function E_{A_E} to determine the time interval at which an action has been executed. Formally, $E_{A_E} : A_E \rightarrow I$.

Temporal relations for decisions Finally, the knowledge formalism needs to include the last, but not the least, element: decisions made by the adaptation, $K = (N, E, V^T, Z^T, R_A, A_E, D)$ While the source of relations in D represents the state before the execution of an action, the target shows its impact on the context. Its intent is **to trace back impacts of action executions to the decisions they originated from**.

A decision in D is defined as a set of executed actions, *i.e.*, a subset of A_E , combined with a set of requirement analysis, *i.e.*, a subset of R_A . Formally,

$D = \{ A_D \cup R_D \mid A_D \subseteq A_E, R_A \subseteq R_P \}$. We assume that each action should result from only one decision: $\forall a \in A, \forall d1, d2 \in D \mid a \in d1 \wedge a \in d2 \rightarrow d1 = d2$.

The temporal function E_{A_E} is extended to decisions in order to represent the execution time: $E_{A_E} : (A \cup D) \rightarrow I$. For decision, the lower bound of the interval corresponds to the lowest bound of the action execution intervals. Following the same principle, the upper bound of the interval corresponds to the uppermost bound of the action execution intervals. Formally, $\forall d \in D \rightarrow E_{A_E}(d) = [l, u]$, where $l = \min_{a \in A_d} \{E_{A_E}(a)[start]\}$ and $u = \max_{a \in A_d} \{E_{A_E}(a)[end]\}$.

Sum up Knowledge of an adaptive system can be formalism with a temporal graph such as $K = (N, E, V^T, Z^T, R_A, A_E, D)$, wherein:

- N is a set of nodes to represent the different information (context, actions and requirements)
- E is a set of edges which connects the different nodes,
- V^T is a temporal relation which defines the temporal validity of each element,
- Z^T is a relation to track the history of each knowledge elements,
- R_A is a relation that defines the different requirements processes,
- A_E is a relation that defines the different action processes,
- D is a set of action executions, which result from the same decision, and requirement analysis.

Decisions D can allow adaptation processes to reason over ongoing and future executions of decisions. Moreover, it allows tracing the state of the knowledge before and after the decision has been or is executed, thanks to its A_D component. Plus, it represents which action has been used for this. Thanks to the R_A relation, one can access the requirements at the root of the decision and the state of the knowledge used by this requirement.

In the next section, we exemplify this formalism over our case study.

7.1.3 Application on the use case

In this section we apply the formalism described on the use case presented in Section 3.3.2.

Let K_{SG} be the temporal graph that represents the knowledge of this adaptive system: $K_{SG} = (N_{SG}, E_{SG}, V_{SG}^T, Z_{SG}^T, R_{PSG}, A_{PSG}, D_{SG})$. Figure 7.3 shows the nodes and edges of this knowledge.

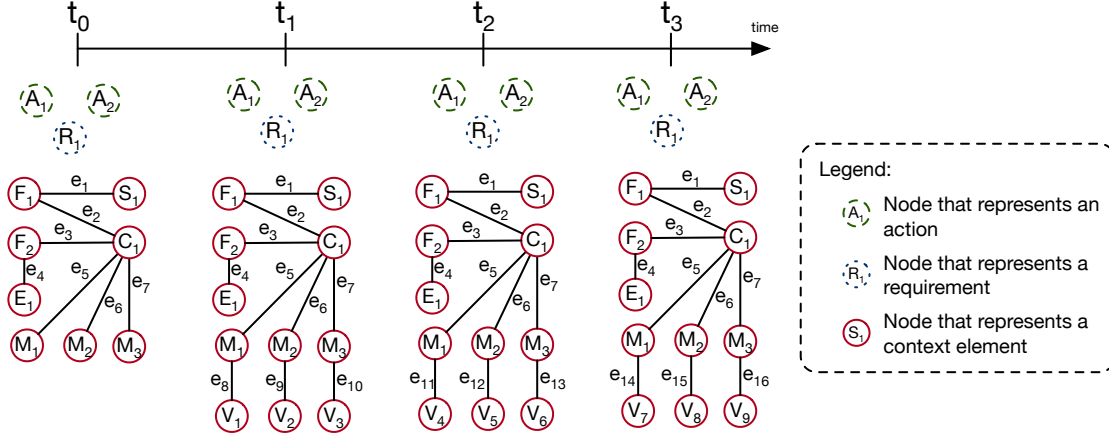


Figure 7.3: Application of the formalism with a temporal graph that represents the knowledge of the smart grid described in Section 3.3.2

Description of N_{SG} N_{SG} is divided into three subsets: C_{SG} , R_{SG} and A_{SG} . R_{SG} contains one node, R_1 in Figure 7.3, which represents the requirement of this example (minimizing the number of overloads): $R_{SG} = \{R_1\}$. Two nodes, A_1 and A_2 , belong to A_{SG} : $A_{SG} = \{A_1, A_2\}$. They represent the two actions of this example, respectively decreasing and increasing amps limits. Regarding the context C_{SG} , there are three nodes to represent the three smart meters (M_1 , M_2 , and M_3), one for the substation (S_1), two for the fuses (F_1 and F_2), one for the dead-end cabinet (E_1), one for the cable (C_1) and one node per consumption value received (V_i): $C_{SG} = \{M_1, M_2, M_3, S_1, F_1, F_2, E_1, C_1\} \cup \{V_i | i \in [1..9]\}$.

According to the scenario, except for nodes to store consumption values, the other nodes are created at t_0 and are never modified. Therefore, their validity period starts at t_0 and never ends: $\forall n \in A_{SG} \cup R_{SG} \cup \{M_1, M_2, M_3, S_1, F_1, F_2, E_1, C_1\}, V_{SG}^T(n) = [t_0, t_\infty)$. Considering the consumption values, all the nodes represent the history of the values for the three smart meters. In other words, there are three knowledge elements: the consumption measured for each meter. Let C_i notes the consumption measured by the smart meter M_i . As shown in Figure 7.3, there is:

- C_1 of M_1 is represented by $\{V_1, V_4, V_7\}$,
- C_2 of M_2 is represented by $\{V_2, V_5, V_8\}$,
- C_3 of M_3 is represented by $\{V_3, V_6, V_9\}$.

Taking C_2 as an example, V_2 is the initial consumption value, replaced by V_5 at t_2 ,

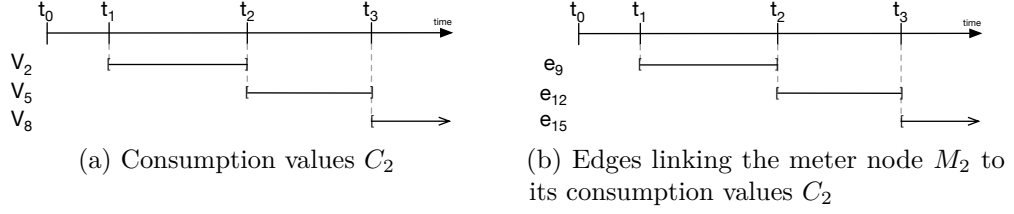


Figure 7.4: Validity periods of consumption values and their edges to the smart meter M_2

itself replaced by V_8 at t_3 . Applying the V_{SG}^T on these different values, results are thus:

- $V_{SG}^T(V_2) = [t_1, t_2)$,
- $V_{SG}^T(V_5) = [t_2, t_3)$,
- $V_{SG}^T(V_8) = [t_3, t_\infty)$.

These validity periods are shown in Figure 7.4a. As meters send the new consumption values at the same time, this example can also be applied to C_1 and C_3 .

From these validity periods, the Z_{SG}^T can be used to navigate to the different values over time. Let's continue with the same example, C_2 . In order to get the evolution of the consumption value C_2 , given the initial one, one will use the Z_{SG}^T relation:

- $Z_{SG}^T(V_2, t_{s1}) = \emptyset$, where $t_0 \leq t_{s1} < t_1$
- $Z_{SG}^T(V_2, t_{s2}) = V_2$, where $t_1 \leq t_{s2} < t_2$
- $Z_{SG}^T(V_2, t_{s3}) = V_5$, where $t_2 \leq t_{s3} < t_\infty$.
- $Z_{SG}^T(V_2, t_{s4}) = V_8$, where $t_2 \leq t_{s4} < t_\infty$.

Description of E_{SG} In this example, edges are used to store the relationships between the different context elements. For example, the edge between the substation S_1 and the fuse F_1 allow representing the fact that the fuse is physically inside the substation. Another example, edges between the cable C_1 and the meters M_1 , M_2 and M_3 represent the fact that these meters are connected to the smart grid through this cable.

One may consider that relations (validity, Z^T , decisions, action executions and requirements analysis) will be stored as edges. But this decision is let to the

implementation part of this formalism.

In our model, only consumption values (V_i nodes) are modified over time. Plus, since the scenario does not imply any edge modifications, only those between meters and values are modified. The edge set contains thus sixteen edges: $E_{SG} = \{e_i \mid i \in [1..16]\}$.

By definition, the unmodified edges have a validity period starting from t_0 and never ends: $\forall i \in [1..7], V_{SG}^T(e_i) = [t_0, t_\infty)$. The history of the three knowledge elements that represent consumption values do not only impact the nodes which represent the values but also the edges between those nodes and the meters ones:

- C_1 impacts edges between M_1 and V_1, V_4 , and V_7 , *i.e.*, $\{e_8, e_{11}, e_{14}\}$,
- C_2 impacts edges between M_2 and V_2, V_5 , and V_8 , *i.e.*, $\{e_9, e_{12}, e_{15}\}$,
- C_3 impacts edges between M_3 and V_3, V_6 , and V_9 , *i.e.*, $\{e_{10}, e_{13}, e_{16}\}$.

Continuing with C_2 as an example, the initial edge value is e_9 from t_1 , which is replaced by e_{12} from t_2 , itself replaced by e_{15} from t_2 . The validity relation, applied to these edges, thus returns:

- $V_{SG}^T(e_9) = [t_1, t_2) = V_{SG}^T(V_2)$,
- $V_{SG}^T(e_{12}) = [t_2, t_3) = V_{SG}^T(V_5)$,
- $V_{SG}^T(e_{15}) = [t_3, t_\infty) = V_{SG}^T(V_8)$,

These validity periods are depicted in Figure 7.4b. As they are driven by those of consumption values (V_2, V_5 , and V_8), they are equal.

As for nodes, the Z_{SG}^T relation can navigate over time through these values. For example, to get the history of the edges between the consumption value C_2 and the meter represented by M_2 , one can apply the Z_{SG}^T relation as follows:

- $Z_{SG}^T(e_9, t_{s1}) = \emptyset$, where $t_0 \leq t_{s1} < t_1$
- $Z_{SG}^T(e_9, t_{s2}) = e_9$, where $t_1 \leq t_{s2} < t_2$,
- $Z_{SG}^T(e_9, t_{s3}) = e_{12}$, where $t_2 \leq t_{s3} < t_3$,
- $Z_{SG}^T(e_9, t_{s4}) = e_{15}$, where $t_3 \leq t_{s4} < t_\infty$.

Description of D_{SG} , A_{ESG} , and R_{ASG} As described in the scenario (cf. Section 3.3.2), the requirement analysis detects that t_1 the requirement is broken. The adaptation process will thus apply the “decreasing amps limits” action on the three meters. Following Example 2 detailed in Section 3.3.1, we consider that the action will impact the consumption values on the next two measurements: t_2 and t_3 .

In the knowledge, we thus have one decision: $D_{SG} = D_1$. This decision has been taken after one requirement analysis, R_{ASG1} , that detects no respect of the requirement R_1 . To determine if there is an overload, this analysis needs to know the topology and the consumption values. The pattern is thus defined by all nodes related to the grid network and consumption values at t_1 : $P_{1[t_1, t_1+1]} = \{S_1, F_1, F_2, C_1, E_1, M_1, M_2, M_3, V_1, V_2, V_3\}$. So we have: $R_{ASG1} = \{R_1, P_{1[t_1, t_1+1]}\}$.

The knowledge also includes the three action executions: A_{ESG1} , A_{ESG2} , and A_{ESG3} . These actions have been executed on, respectively, M_1 , M_2 , and M_3 . Following the definition, they all contain the action A_1 and similar relation which linked the circumstances to the impacts. The circumstances are the state of the knowledge at t_0 , which contain all information of the grid network and the consumption values. We denote them $P_{2[t_1, t_1+1]}$, $P_{3[t_1, t_1+1]}$, and $P_{4[t_1, t_1+1]}$, all equal $P_{1[t_1, t_1+1]}$. The impact contains all consumption values received at t_2 and t_3 . Each action impacts the consumption value of the meter that it modifies. For example, A_{ESG2} only impacts values of meter M_2 . For this action, the output pattern is thus : $P_{5[t_2, t_3]} = \{V_5, V_8\}$. In summary, A_{ESG1} , A_{ESG2} , and A_{ESG3} are defined as follows:

- for the action executed on M_1 : $A_{ESG1} = (A_1, A_{F1})$, with $A_{F1} : P_{2[t_1, t_1+1]} \rightarrow \{V_4, V_7\}$,
- for the action executed on M_2 : $A_{ESG2} = (A_1, A_{F2})$, with $A_{F2} : P_{3[t_1, t_1+1]} \rightarrow \{V_5, V_8\}$,
- for the action executed on M_3 : $A_{ESG3} = (A_1, A_{F3})$, with $A_{F3} : P_{4[t_1, t_1+1]} \rightarrow \{V_6, V_9\}$,

The decision described in the scenario is thus equal to: $D_1 = \{R_{ASG1}, A_{ESG1}, A_{ESG2}, A_{ESG3}\}$. At t_2 , this decision will still be valid. The adaptation process can thus include it in the adaptation process to reason over the ongoing actions. If at t_3 the cable remains overloaded, then one may use this element to check if the system tried to fix it, how and based on which information.

7.2 Modelling the knowledge

In order to simplify the diagnosis of adaptive systems, this thesis proposes a novel metamodel that combines, what we call, design elements and runtime elements. Design elements abstract the different elements involved in knowledge information to assist the specification of the adaptation process. Runtime elements

instead, represent the data collected by the adaptation process during its execution. In order to maintain the consistency between previous design elements and newly created ones, instances of design elements (*e.g.*, actions) can be either added or removed. Modifying these elements would consist in removing existing elements and creating new ones. Combining design elements and runtime elements in the same model helps not only to acquire the evolution of system but also the evolution of its structure and specification (*e.g.* evolution of the requirements of the system). Design time elements are depicted in gray in the Figures 7.5– 7.8. Note that, this thesis does not address how runtime information is collected.

For the sake of modularity, the metamodel has been split into four packages: Knowledge, Context, Requirement and Action. All the classes of these packages have a common parent class that adds the temporality dimension: *TimedElement* class. Before describing the Knowledge (core) package, we detail this element. Then, we introduce in more details the other three packages used by the Knowledge package: Context, Requirement, and Action. In below sections, we use "*Package::Class*" notation to refer to the provenance of a class. If the package is omitted, then the provenance package is this one described by the figure or text.

7.2.1 Parent element: *TimedElement* class

We assume that all the classes in the different packages extend a *TimedElement* class. This class contains three methods: *startTime*, *endTime*, and *modification-time*. The first two methods allow accessing the validity interval bounds defined by the previously discussed V^T relation. The last method resolves all the timestamps at which an element has been modified: its history. This method is the implementation of the relation Z^T described in our formalism (*cf.* Section 7.1.2).

7.2.2 Knowledge metamodel

In order to enable interactive diagnosis of adaptive systems, traceability links between the decisions made and their circumstances should be organized in a well-structured representation. In what follows, we introduce how the knowledge metamodel helps to describe decisions, which are linked to their goals and their context (input and impact). Figure 7.5 depicts this metamodel.

Knowledge package is composed of a *context*, a set of *requirements*, a set of

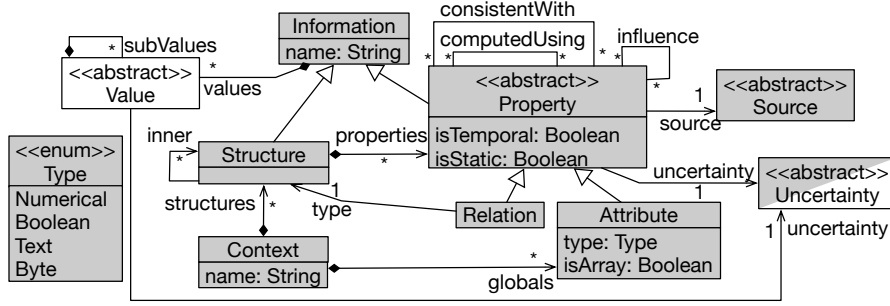


Figure 7.6: Excerpt of the context metamodel

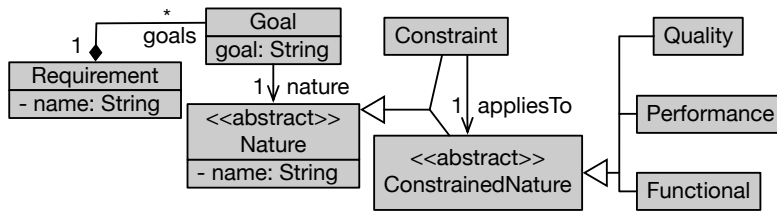


Figure 7.7: Requirement metamodel

2 represent the context as a set of structures (*Context.structures*) and global attributes
 3 (*Context.globals*). A structure can be viewed as a C-structure with a set of properties
 4 (*Property*): attributes (*Attribute*) or relationships (*Relation*). A structure may
 5 contain other nested structures (*Structure.inner*). Structures and properties have
 6 values. They correspond to the nodes described in the formalization section (*cf.*
 7 Section 7.1.2). The connection feature described in Section 2.1.3 is represented
 8 thanks to three recursive relationships on the Property class: *consistentWith*,
 9 *computedUsing* and *influence*. Additionally, each property has a source (*Source*)
 10 and an uncertainty (*Uncertainty*). It is up to the stakeholder to extend data with the
 11 appropriate source: measured, computed, provided by a user, or by another system
 12 (*e.g.*, weather information coming from a public API). Similarly, the uncertainty
 13 class can be extended to represent the different kinds of uncertainties. Finally, a
 14 property can be either historic or static.

7.2.4 Requirement metamodel

16 As different solutions to model system requirements exist (*e.g.*, KAOS [DvLF93],
 1 i* [Yu11] or Tropos [BPG⁺04]), in this metamodel, we abstract their shared concepts.

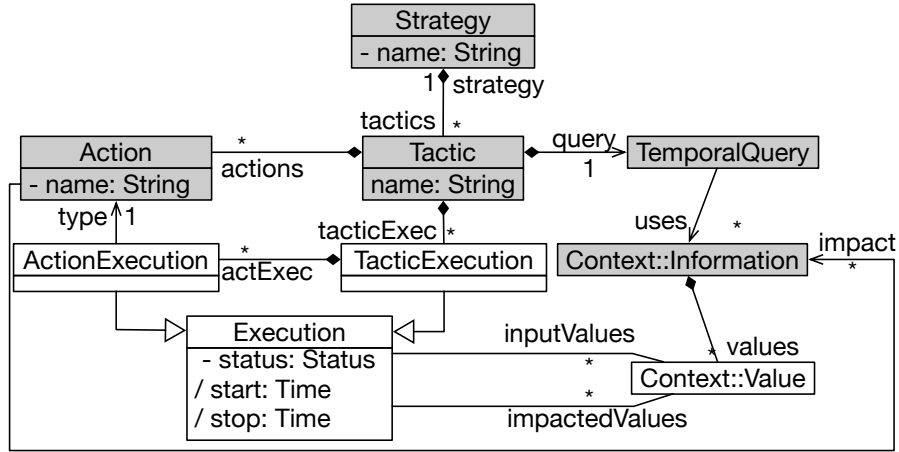


Figure 7.8: Excerpt of the action metamodel

2 The requirement model, depicted in Figure 7.7, represents the *requirement* as a
 3 set of *goals*. Each goal has a *nature* and a textual specification. The nature of the
 4 goals adheres to the four categories of requirements presented in Section 2.1.3. One
 5 may use one of the existing requirements modelling languages (*e.g.*, RELAX) to
 6 define the semantics of the requirements. Since the requirement model is composed
 7 solely of design elements, we may rely on static analysis techniques to infer the
 8 requirement model from existing specifications. The work of Egyed [Egy01] is one
 9 solution among others. This work is out of the scope of the thesis and envisaged
 10 for future work.

11 In the guidance example, the requirement model may contain a **balanced**
 12 **resource distribution** requirement. It can be split into different goals: (i)
 13 *minimizing overloads*, (ii) *minimizing production lack*, (iii) *minimizing production*
 14 *loss*.

15 7.2.5 Action metamodel

16 Similar to the requirements metamodel, the actions metamodel also abstracts
 17 main concepts shared among existing solutions to describe adaptation processes
 18 and how they are linked to the context. Figure 7.8 depicts an excerpt of the
 19 action metamodel. we define a strategy as a set of tactics (*Strategy*). A tactic
 20 contains a set of actions (*Action*). A tactic is executed under a precondition
 1 represented as a temporal query (*TemporalQuery*) and uses different data from

the context as input. In future work, we will investigate the use of preconditions to schedule the executions order of the actions, similarly to existing formalisms such as Stitch [CG12]. The query can be as complex as needed and can navigate through the whole knowledge model. Actions have impacts on certain properties, represented by the *impacted* reference.

The different executions are represented thanks to the *Execution* class. Each execution has a status to track its progress and links to the impacted context values (*Execution.impactedValues*). Similarly, input values are represented thanks to the *Execution.inputValues* relationship. An execution has *start* and *end* time. Not to confuse with the *startTime* and *endTime* of the validity relation V^T . Whilst the former corresponds to the time range in which a value is valid, the *start* and *stop* time in the class execution correspond to the time range in which an action or a tactic was being executed. The start and stop attributes correspond to the relation E_{AE} (see Section 7.1.2). These values can be derived based on the validity relation. They correspond to the time range in which the status of the execution is “*RUNNING*”. Formally, for every execution node e , $E_{AE}(e) = (V(e) \mid e.status = \text{“RUNNING”})$.

Similarly to requirement models, it is possible to automatically infer design elements of action models by statically analyzing actions specification. Since acquiring information about tactics and actions executions happens at runtime, one way to achieve this is by intercepting calls to actions executions and updating the appropriate action model elements accordingly. This is out of the scope of this thesis and planned for future work.

7.3 Validation

To validate and evaluate our approach, we implemented a prototype publicly available online¹. This implementation leverages the GreyCat framework², more precisely the modelling plugin, which allows designing a metamodel using a textual syntax. Based on this specification, GreyCat generates a Java and a JavaScript API to create and manipulate models that conform to the predefined metamodel. The GreyCat framework handles time as a built-in concept. Additionally, it has native support of a lazy loading mechanism and an advanced garbage collection.

¹<https://github.com/lmouline/LDAS>

²<https://github.com/datathings/greycat>

2 This is achieved by dynamically loading and unloading model elements from the
3 main memory when necessary.

4 The validation of our approach has been driven by the two research questions
5 formulated in the introduction section:

- 6 • How to diagnose the self-adaptation process?
- 7 • How to enable reasoning over unfinished actions and their expected effects?

8 To address the first one, we describe how one can use our approach to represent
9 the knowledge of an adaptation process for a smart grid system. Then, we present a
10 code to extract the circumstances and the goals of a decision. For the second one, we
11 present a scenario where a developer can use our approach to reason over unfinished
12 actions and their expected effects. The presented code shows how information can
13 be extracted from our model to enable any reasoning algorithm. Finally, we present
14 a performance evaluation to show the scalability of our approach.

15 **7.3.1 Diagnostic: implementation of the use case**

16 In what follows, we explain how a stakeholder, Morgan, can apply our approach
17 to a smart grid system in order to, first, abstract adaptive system concepts, then,
18 structure runtime data, and finally, query the model for diagnosis purpose. The
19 corresponding object model is depicted in Figure 7.9. Due to space limitation, we
20 only present an excerpt of the knowledge model. An elaborate version is accessible
21 in the tool repository.

22 **Abstracting the adaptive system** At design time (t_d), either manually or
23 using an automatic process, Morgan abstracts the different tactics and actions
24 available in the adaptation process. Among the different tactics that Morgan would
25 like to model is “*reduce amps limit*”. It is composed of three actions: sending a
26 request to the smart meter (*askReduce*), checking if the new limit corresponds to
27 the desired one (*checkNewLimit*), and notifying the user by e-mail (*notifyUser*).
28 Morgan assumes that the *askReduce* action impacts consumption data (*csmpt*).
29 This tactic is triggered upon a query (*tempQ*) that uses meter (*mt*), consumption
30 (*csmpt*) and customer (*cust*) data. The query implements the “*no overload*” goal:
31 the system shall never have a cable overload. Figure 7.9 depicts a flattened version
32 of the temporal model representing these elements. The tag at upper-left corner of
1 every object illustrates the creation timestamp. All the elements created at this

stage are tagged with t_d .

Adding runtime information The adaptation process checks if the current system state fulfills the requirements by analyzing the context. To perform this, it executes the different temporal queries, including *tempQ*. For some reasons, the *tempQ* reveals that the current context does not respect the “no overload” goal. To adapt the smart grid system, the adaptation process decides to start the execution of the previously described tactic (*exec1*) at t_s . As a result, a decision element is added to the model along with a relationship to the unsatisfied goal. In addition, this decision entails the planning of a tactic execution, manifested in the creation of the element *exec1* and its subsequent actions (*notifyU*, *checkLmt*, and *askRed*). At t_s , all the actions execution have an IDLE status and an expected start time. All the elements created at this stage are tagged with the t_s timestamp in Figure 7.9.

At t_{s+1} , the planned tactic starts being executed by running the action *askReduce*. The status of this action turns from IDLE to RUNNING. Later, at t_{s+2} , the execution of *askReduce* finishes with a SUCCEED status and triggers the execution of the actions *notifyUser* and *checkNewLimit* in parallel. The status of *askReduce* changes to SUCCEED while the status of *notifyUser* and *checkNewLimit* turns to RUNNING. The first action successfully ends at t_{s+3} while the second ends at t_{s+4} . As all actions terminates with a SUCCEED status at t_{s+4} , accordingly, the final status of the tactic is set SUCCEED and the *stop* attribute value is set to t_e .

Interactive diagnosis query After receiving incident reports concerning regular power cuts, and based on the aforementioned knowledge model, Morgan would be able to query the system’s states and investigate why such incidents have occurred. As described in Section 3.3.2, she/he will interactively diagnose the system by interrogating the context, the decisions made, and their circumstances.

The first function, depicted in Listing 7.1, allows to navigate from the currently measured values (*vcur1*) to the decision(s) made. The for-loop and the if-condition are responsible for resolving the measured data for the past two days. Past elements are accessed using the *resolve* function that implements the Z^T relation (cf. Section 7.1.2). After extracting the decisions leading to power cuts, Morgan carries on with the diagnosis by accessing the circumstances of this decision. The code to perform this task is depicted in Listing 7.1, the second function

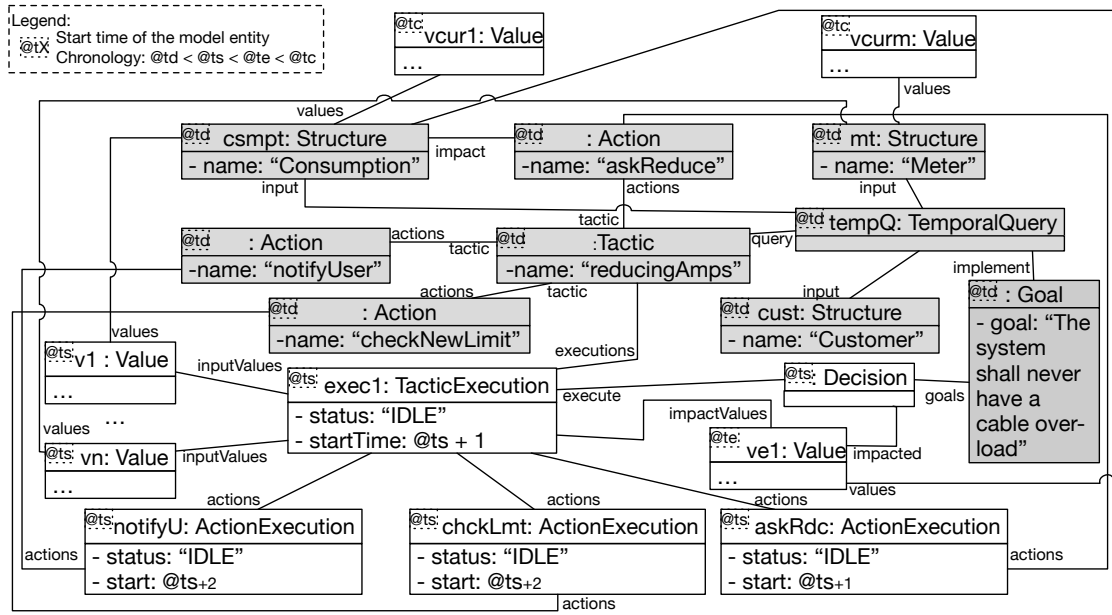


Figure 7.9: Excerpt of the knowledge object model related to our smart grid example

2 (getCircumstances). Note that the relationship *Decision.input* is the aggregation of
 3 *Decision.execute.inputValues*.

```

4
5 // extracting the decisions
6 Decision [] impactedBy(Value v) {
7     Decision [] respD
8     for( Time t: v.modificationTimes() ):
9         if (t >= v.startTime() - 2 day)
10             Value resV = resolve(v,t)
11             respD.addAll(from(resV).navigate(Value.impactd))
12     return respD
13 }
14 // extracting the circumstances of the made decisions
15 Tuple<Value[], Goal[]> getCircumstance(Decision d) {
16     Value [] resValues = from(d).navigate(Decision.input)
17     Goal [] resGoals = from(d).navigate(Decision.goals)
18     return Tuple<>(resValues, resGoals)
19 }

```

Listing 7.1: Get the goals used by the adaptation process from executed actions

7.3.2 Reasoning over unfinished actions and their expected effects

By associating the action model to the knowledge model, we aim at enhancing adaptation processes with new abilities to reason. In this section, we present an example of a reasoning algorithm which considers the impacts of running actions. This example is based on our use case (*cf.* Chapter 3).

Let's imagine that the adaptation process detects overloaded cables in the smart grid. To fix this situation, it takes several countermeasures, among which there are fuse state modifications. As detailed in Section 3.3.1, this action is considered as long-term action. Later, another incident is detected, for example, a substation is being overloaded. Before taking any actions, the adaptation process can, thanks to our solution, verify if the running actions will be sufficient to solve this new incident. If not, it can either take additional actions or replan the running one. The algorithm to reschedule current actions or to compute additional actions is out of the scope of this thesis. Here, we present the code to extract the required information from our model.

Checking if the running actions will be sufficient to solve all current issues can also be thought as: will the issue remain with the new context, *i.e.*, after each action have been executed. In our case, it is like verifying if the second overload will still remain with the new topology, which is coming. The adaptation process, therefore, needs to extract the context in the future. To do so, the adaptation process should know the latest timepoint at which the impact will be measured. Listing 7.2 shows the code to get this timepoint. Running, idle and finished actions are accessed thanks to the first two nested loops with the if-condition. We consider that failed and canceled actions have no effects. As finished actions may still have effects, we also consider them. Then we navigate through all impacted values to get their start time, *i.e.*, the beginning of their validity period (V^T relation, *cf.* Section 7.1.2). Doing so, we are sure to get the latest timepoint at which an impact will be measurable.

```

Time latestImpact(Knowledge k) {
    Time latestTime = CURRENT_TIME
    for (Decision d: from(k).navigate(decisions))

```

```

2      for (TacticExecution te: from(d).navigate(execute))
3          if (te.status == "RUNNING" || te.status == "IDLE" || te.status == "SUCCEED")
4              for (Value v: from(te).navigate(impactedValues))
5                  if (v.startTime() > latestTime)
6                      latestTime = v.startTime()
7
8      return latestTime
9  }
10

```

Listing 7.2: Get latest timepoint at which the impact will be measured

Using this timepoint, then the adaptation process can then compute how the grid should be after the actions have been executed. If the system has no prediction mechanism, then the adaptation process can verify how the power will be balanced over the new topology. Otherwise, it can use this prediction feature to compute the expected loads with the coming topology. Using this information, it can verify if all current incidents will be solved by the ongoing actions or not. If not, it may take additional actions or reschedule them.

Listing 7.3 depicts the code to extract all running actions. The nested loops allow accessing all executions made by decision. Then, we filter only those with the “RUNNING” status. The resulting collection should then be given to the scheduling algorithm, which will decide if rescheduling is possible and how.

```

22
23  TacticExecution [] runningActions (Knowledge k) {
24      TacticExecution [] resA
25      for (Decision d: k.decisions) {
26          for (TacticExecution te: d.execute) {
27              if (te.status == Status.RUNNING) {
28                  resA.add(te)
29              }
30          }
31      }
32      return resA
33  }
34

```

Listing 7.3: Extract ongoing actions and their effects

Using our model, developers have two solutions to model a rescheduling operation. Either they modify the actions, which may delete the history of the previous decision, or they mark all running and idle actions as “CANCELED” and create a new decision, with new actions, which update the circumstances and re-use the same requirements.

7.3.3 Performance evaluation

GreyCat stores temporal graph elements in several key/value maps. Thus, the complexity of accessing a graph element is linear and depends on the size of the graph. Note that in our experimentation we evaluate only the execution performance of diagnosis algorithms. For more information on I/O performance in GreyCat, please refer to the original work by Hartmann *et al.*, [HFJ⁺17; Har16].

```
MATCH (input)-[*4]->(output)
WHERE input.id IN [randomly generated set]
RETURN output
LIMIT O
```

Listing 7.4: Traversal used during the experimentations

We consider a diagnosis algorithm to be a graph navigation from a set of nodes (input) to another set of nodes (output). Unlike typical graph algorithms, diagnosis algorithms are simple graph traversals and do not involve complex computations at the node level. Hence, we believe that three parameters can impact their performance (memory and/or CPU): the global size of the graph, the size of the input, and the number of traversed elements. In our evaluation, we altered these parameters and report on the behaviour of the main memory and the execution time. The code of our evaluation is publicly available online³. All experiments reporting on memory consumption were executed 20 times after one warm-up round. Whilst, execution time experiments were run 100 times after 20 warm-up rounds. The presented results correspond to the mean of all the iterations. We randomly generate graph with sizes (N) ranging from 1 000 to 2 000 000. At every execution iteration, we follow these steps: (1) in a graph with size N , we randomly select a set of I input nodes, (2) then traverse M nodes in the graph, (3) and we collect the first O nodes that are at four hops from the input element. Listing 7.4 describes the behaviour of the traversal using Cypher, a well-known graph traversal language.

We executed our experimentation on a MacBook Pro with an Intel Core i7 processor (2.6 GHz, 4 cores, 16GB main memory (RAM), macOS High Sierra version 10.13.2). We used the Oracle JDK version 1.8.0_65.

³<https://bitbucket.org/ludovicpapers/icac18-eval>

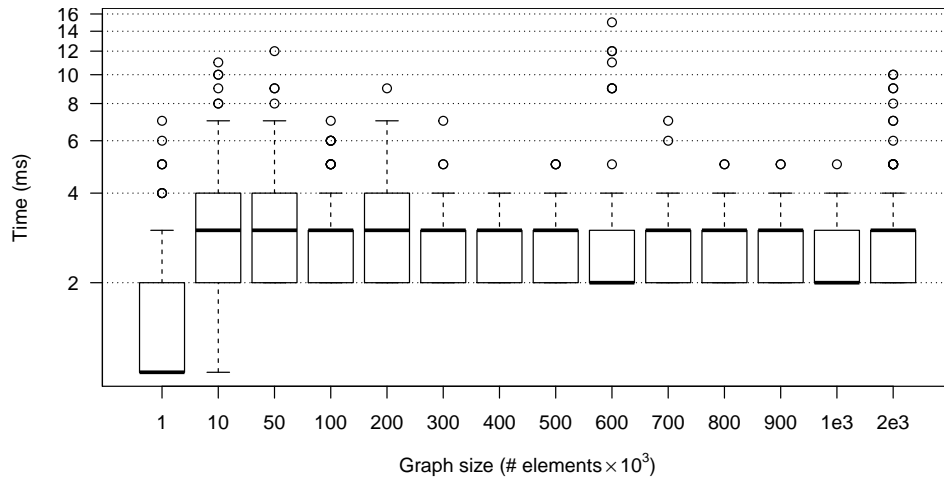
2 **How is the performance influenced by the graph size N ?** This experimen-
3 tation aims at showing the impact of the graph size (N) on memory and execution
4 time while performing common diagnosis routines. We fix the size of I to 10. To
5 assure that the behaviour of our traversals is the same, we use a seed value to select
6 the starting input elements. We stop the algorithm when we reach 10 elements.
7 Results are depicted in Figure 7.10.

8 As we can notice, the graph size does not have a significant impact on the
9 execution time of diagnosis algorithms. For graphs with up to 2,000,000 elements,
10 execution time remains between 2 ms and four 4 ms. We can also notice that the
11 memory consumption insignificantly increases. Thanks to the implementation of
12 a lazy loading and a garbage collection strategy by GreyCat, the graph size does
13 not influence memory or execution time performance. The increase in memory
14 consumption can be due to the internal indexes or stores that grow with the graph
15 size.

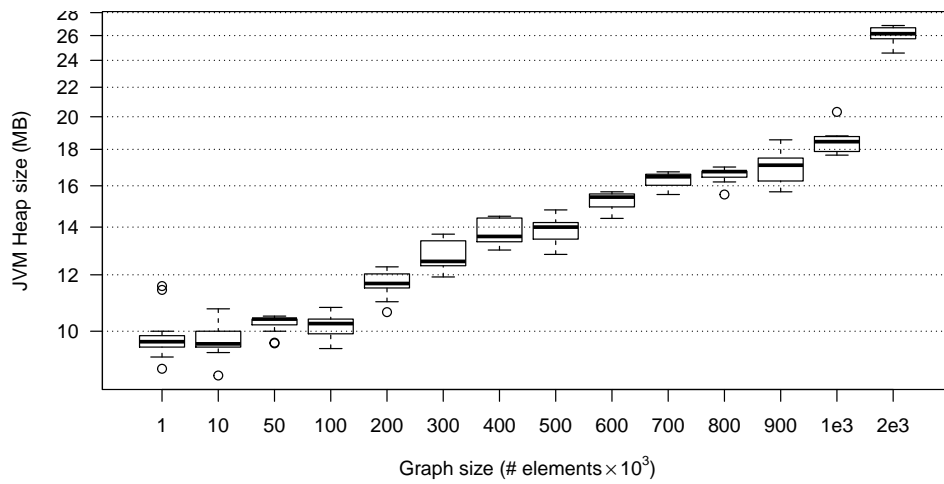
16 **How is the performance influenced by the input size (I)?** The second
17 experiment aims to show the impact of the input size (I) on the execution of
18 diagnosis algorithms. We fix the size of N to 500 000 and we variate I from 1 000
19 nodes to 100 000, *i.e.*, from 0.2% to 20% of the graph size. The results are depicted
20 in Figure 7.11 (straight lines).

21 Unlike to the previous experiment, we notice that the input size (I) impacts
22 the performance, both in terms of memory consumption and execution time. This
23 is because our framework keeps in memory all the traversed elements, namely the
24 input elements. The increase in memory consumption follows a linear trend with
25 regards to N . As it can be noticed, it reaches 2GB for $I=100\,000$. The execution
26 time also shows a similar curve, while the query response time takes around than
27 around 60ms to run for $I=1\,000$, it takes a bit more than 4 seconds to finish
28 for $I=100\,000$. Nonetheless, these results remain very acceptable for diagnosis
29 purposes.

30 **How is the performance influenced by the number of traversed elements**
31 **(M)?** For the last experiment, we aim to highlight the impact of the number of
32 traversed elements (M). For this, we fix I and O to 1, and randomly generate a
1 graph with sizes ranging from 1 000 to 100 000. Our algorithm navigates the whole



(a) Execution time evolution



(b) Memory evolution

Figure 7.10: Experimentation results when the knowledge based size increases

2 model ($M=N$). We depict the results in Figure 7.11 (dashed curve). As we can
3 notice, the memory consumption increases in a quasi-linear way. The memory
4 footprint to traverse $M = 100\,000$ elements is around 0.9GB. The progress of the
5 execution time curve behaves similarly, in a quasi-linear way. Finally, the execution
6 time of a full traversal over the biggest graph takes less than 2.5 seconds.

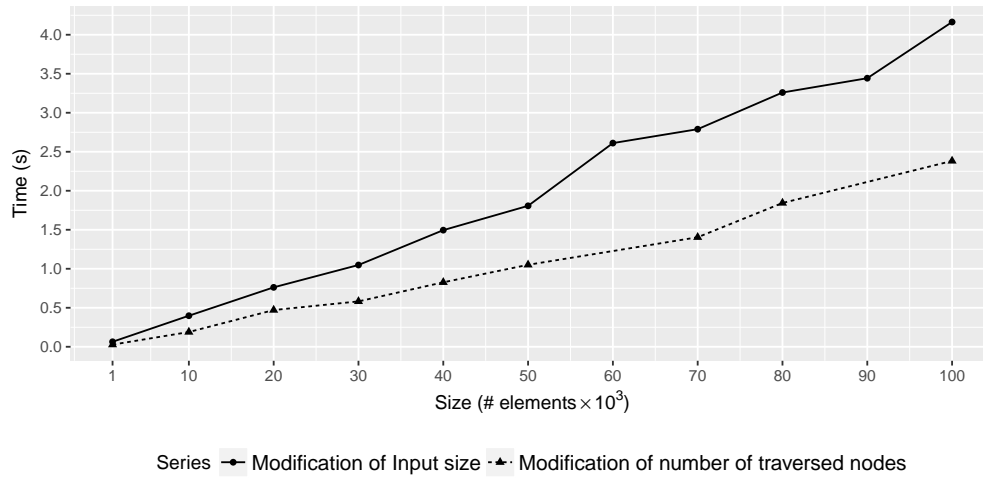
7 7.3.4 Discussion

8 By linking context, actions, and requirements using decisions, data extraction for
9 explanation or fault localization can be achieved by performing common temporal
10 graph traversal operations. In the detailed example, we show how a stakeholder
11 could use our approach to define the different elements required by such systems, to
12 structure runtime data, finally, to diagnose the behaviour of adaptation processes.

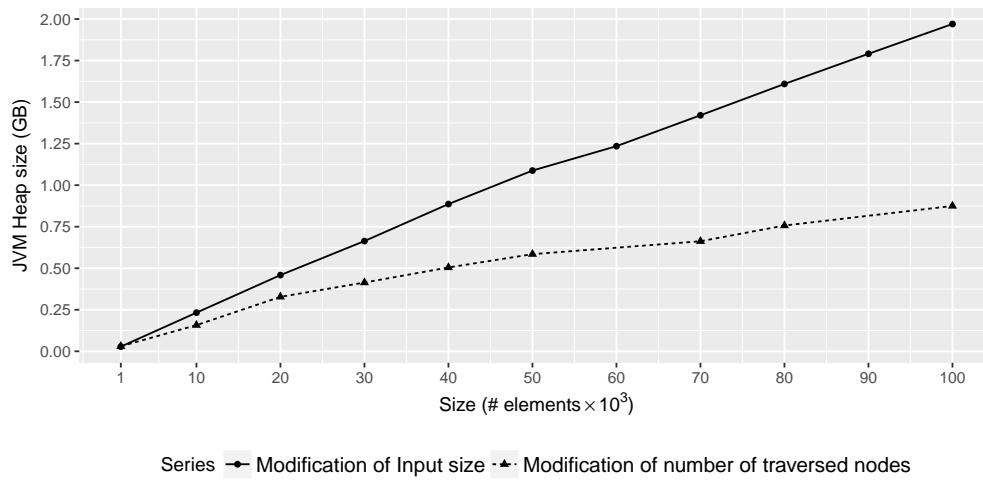
13 Our implementation allows to dynamically load and release nodes during the
14 execution of a graph traversal. Using this feature, only the needed elements are
15 kept in the main memory. Hence, we can perform interactive diagnosis routines on
16 large graphs with an acceptable memory footprint. However, the performance of
17 our solution, in terms of memory and execution time, is restricted by the number
18 of traversed elements and the number of input elements. Indeed, as shown in
19 our experimentation, both the execution time and the memory consumption grow
20 linearly.

21 In the Luxembourg smart grid, a district contains approximatively 3 data
22 concentrators and 227 meters⁴. Counting the global datacenter, the network
23 is thus composed of 231 elements. Each meter sends the consumption value
24 every 15 min, being 908 every hours. Plus, there is from 0 to 273 topology
25 modifications in the network. In total, the system generates from 908 to 1,181
26 new values every hour. If we consider that we have one model element per
27 smart grid entity and one model element per new value, 100,000 model elements
28 correspond thus from $((100,000 - 231) * 1H)/1,181 = 84,5H$ ($\sim 3,5$ days) to
29 $((100,000 - 231) * 1H)/908 = 109,9H$ ($\sim 4,6$ days) of data. In other word, our
1 approach can efficiently interrogate up to ~ 5 days history data in 2.4s of one

⁴Previously, our studies uses the data described in [HFK⁺14b], which corresponded to the all Luxembourg at this date. Since 2014, the smart grid has been more and more deployed. Numbers present in this document now corresponds more to one district.



(a) Evolution of the execution time



(b) Evolution of the memory consumption

Figure 7.11: Results of experiments when the number of traversed or input elements increases

2 district.

3 **7.4 Conclusion**

4 Adaptive systems are prone to faults given their evolving complexity. To enable
5 interactive diagnosis over these systems, we proposed a temporal data model to
6 abstract and store knowledge elements. We also provided a high-level API to
7 specify and perform diagnosis algorithms. Thanks to this structure, a stakeholder
8 can abstract and store decisions made by the adaptation process and link them
9 to their circumstances –targeted requirements and used context– as well as their
10 impacts. In our evaluation, we showed that our solution can efficiently handle up to
11 100 000 elements, in a single machine. This size is comparable to five days history
1 of one district in the Luxembourg smart grid.

Part III

Conclusion and future work

2

3 Conclusion

4 Contents

5 **8.1 Summary 148**

6

7 **8.2 Future work 150**

8

10

11 *In this chapter, we conclude our dissertation by summarising the challenges*
 12 *and the contributions presented in this document. Then, we detail future work. We*
 1 *finish this dissertation by giving an outlook on this thesis.*

2 8.1 Summary

3 Software systems are more and more pervasive and evolve in a more and more
4 complex environment. This complexity comes with a price: the environment is
5 less and less known with high confidence. To face this challenge, systems became
6 adaptive: their structure and their behaviour can be adjusted at runtime in response
7 to changes in the environment, behaviour, or even their specification. One example
8 of such a system is a smart grid: a power grid which includes ICT to optimise
9 energy delivery and service quality. They continuously monitor cable loads to detect
10 any overloads or outages. If one is detected, the system triggers a modification of
11 power flow to heal the system.

12 One way to implement an adaptive system, like a smart grid, is to apply the
13 models@run.time paradigm. This paradigm is part of the MDE methodology,
14 which advocates for the use of models in software engineering. As mentioned in the
15 name, models@run.time defines runtime model with a causal link to the system. A
16 runtime model reflects the state of a system, during its execution. Any modification
17 of the system triggers an update of the model, and vice-versa (causal link).

18 However, we have identified five problems that can reduce the usability of this
19 approach (*cf.* Chapter 1). First, data collected are, for the most part, uncertain,
20 which can mislead the understanding of a system's behaviour, structure, or environ-
21 ment. Second, adaptation actions are never immediate, take time to be executed,
22 and have long-term effects. We refer to these actions as long-term actions. Third,
23 these systems can have emergent behaviour. Four, the different components of a
24 system can evolve at different rates. Last, the evolution of the system is linked
25 with time.

26 Within this dissertation, we argue towards a novel modelling framework that
27 efficiently encapsulates time and uncertainty as first-class concepts. Therefore, in
28 this thesis, we focus on three challenges:

- 29 1. How to ease the manipulation of data uncertainty?
- 30 2. How to enable reasoning over unfinished actions and their expected effects?
- 31 3. How to diagnose the self-adaptation process?

32 In order to solve them, we propose two contributions in this dissertation. First,
1 to answer the first challenge, we defined a language that integrates concepts related

to data uncertainty as a first-class citizen named Ain'tea (*cf.* Chapter 6). In addition to traditional object-oriented language type (boolean, numeric, reference), the language has uncertain boolean, uncertain numeric, and uncertain reference. These types encapsulate two elements: one element to represent the value, and one to represent the uncertainty. Following what has been done in probabilistic programming, the uncertainty is abstracted by a probability distribution. Ain'tea manages five distributions. One distribution is used for uncertain boolean and reference: Bernoulli. The four other distributions are employed for uncertain numbers: Gaussian, Rayleigh, binomial, and Dirac delta function.

As we add new data types, we also modify the language operators and the type system in consequence. Following what is done in the literature, we map the different operators (arithmetic, boolean, comparison) to a process to propagate uncertainty. Therefore, a developer can combine uncertain data as she is used to doing with certain data. Additionally, we define specific operators to reason over uncertainty, such as the *confidence* and *exist* operators. The type system natively considers these new data types. Statically, it checks if the combination of uncertain data follows the constraint of uncertainty propagation, that is the probability theory. If not, error messages are thrown during the development time to guide developers in their choices.

In our validation, we show that our language is as concise as state-of-the-art solutions. Contrary to these solutions, we also show that our solution can detect errors earlier. Thanks to the semantics, which supports uncertainty, error messages help developers in their development of algorithms that use uncertain data.

Our second contribution, which tackles the two other challenges, is a temporal knowledge metamodel (*cf.* Chapter 7). Contrary to state-of-the-art solutions, this metamodel adds the concept of long-term action. Thanks to this structure, a stakeholder can abstract and store decisions made by the adaptation process and link them to their circumstances –targeted requirements and used context– as well as their effects. Moreover, the metamodel is also supported by a graph-based formalism. We formalise the different elements of the knowledge of an adaptive system: requirements, actions (design time and runtime), and context.

When a designer defines a model that conforms to our metamodel, she will benefit from an API. An engineer can use this API to interact with the model at

runtime. In our evaluation, we showed that our solution can efficiently handle up to 100 000 elements in a single machine. This size is comparable to 5 days history of the Luxembourg smart grid.

These contributions do not answer all questions related to the problems identified. Besides, they suffer from limitations that can lead to further research efforts. In the next section, we detail these limitations with the perspectives.

8.2 Future work

Both our contributions suffer from limitations that call for further research. First, our work does not study all the impacts of uncertainty in software languages. Second, our model is just one step towards (self-)explanation of adaptive system¹. In this section, we describe two research directions left open by our work.

8.2.1 Software language and uncertain data

Software relies more and more on data to make decisions or to help decision-making processes. Engineers should, therefore, handle the uncertainty existing in data. In this document, we argue for managing this uncertainty at the language level. Our contribution addresses uncertainty on values and with a limited number of probability distributions. In our language, we focus our studies on primitive data types (numeric and boolean) and references (1:1 relation). Moreover, we map the uncertainty propagation to operators.

However, we do not tackle the global challenge regarding the introduction of data uncertainty management at the language level. We can formulate the challenge as follows: how to handle data uncertainty at the language level? We can split it into different problems that call for research efforts:

- What are the impacts of data uncertainty on the control flow?
- How to extend all data structures with uncertainty?
- What are the mechanisms to evaluate and reduce uncertainty and how to integrate them into the language?
- How to represent data uncertainty?

¹Explanation is a capacity of software that helps engineers to understand the system behaviour.

Data uncertainty and control flow We did not study the impact of control flow statements on the uncertainty propagation. Introducing uncertainty as a first-class citizen will inevitably modify current behaviour of control flow statements such as *IF*-conditions. We strongly think that this new data type in the type system will lead to further research directions. We identified two other situations that should be considered, with their challenges. The first one is the propagation through control flow statements: how the uncertainty is propagated after an *IF*-condition, a *FOR*-loop and a *WHILE*-loop. The second one is the propagation from one kind of uncertainty to another one. For example, how the uncertainty of presence should be propagated to the uncertainty of a sum, average or variance computation?

Conditional expressions, which have a boolean type, modify the control flow by forking it. With certain boolean, the expression is evaluated at runtime, and one branch is selected according to the result. With uncertain ones, the executor cannot decide which branch to execute. We thus identified two possible controls flows: the classical and the uncertain execution.

For the classical one, a cast operation should be performed to get a certain one. It can be done using at least two strategies. First, a random selection from the probability distribution can be made. Second, a cast can be done using the *cast* or *confidence* operator of our language (*cf.* Chapter 6). This operation gives the most confident value. Let us take an example, an uncertain boolean that equals *FALSE* with a confidence of 40%. Following the first strategy, there is a 40% chance of getting *FALSE*, and so 60% for *TRUE*. Using the second one, it always results in a *TRUE* value, as it's the most confident value.

For the uncertain execution, as the executor cannot decide which branch should be executed, it should execute all of them and propagate the uncertainty. For example, let us imagine the following code:

```

uncertain_bool b = (TRUE, 0.4)
uncertain_int n;
if(b)
    n = (5, 0.8)
else
    n = (-5, 0.8)

```

Listing 8.1: Example for uncertain control flow

As *b* is true with a confidence of 40%, *n* should be equal to 5 with a confidence of

2 (40% * 80% = 32%) and to -5 with a confidence of (60% * 80% = 48%) (considering
3 b and n independent).

4 These two execution semantics come with several open questions. One is related
5 to the impact of such techniques to performances (CPU, memory, *etc.*). For the
6 uncertain execution, additional problems persist such as: should the execution be
7 parallel or not? How to ensure that the execution does not have a side effect(s)?
8 And a final one: can other semantics be defined?

9 **Data structures and data uncertainty** In a software language, it exists several
10 data structures, from the simplest ones like arrays to complex ones like graphs
11 or trees. Additionally, the UML specification contains different kinds of relations
12 (1:1, 1:n, n:n, *etc.*). While these data structure and relations are useful to build
13 algorithms to reason over data, they come with new challenges. First, it opens
14 new questions concerning the meaning of an uncertain data structure: what an
15 uncertain array? What an uncertain tree, graph?

16 Let us develop here the challenges brought by an uncertain array. We think
17 that there are three possible definitions of an uncertain array, the last one being
18 the combination of the first two:

- 19 • an uncertain array is a collection when the collection itself is uncertain,
- 20 • an uncertain array is a collection that contains uncertain elements,
- 21 • an uncertain array is a collection that contains uncertain elements and when
22 the collection itself is uncertain.

23 Let us consider the following array of integers: [54, 10, 3, 24]. Following the first
24 definition, the array is uncertain if this collection is not known with the most
25 thorough confidence. That is, the value of the array can be [6, 10, 3, 26] or [6, 10,
26 3, 26, 587]. As a consequence, the size of the array is also uncertain. In the case of
27 an ordered array, the order can also be uncertain. Using the second definition, 54,
28 10, 3, and 24 are all uncertain integers.

29 Additionally, research efforts have to be done to specify the semantics of
30 operators on these uncertain structures. For example, one may focus on how the
31 uncertainty of the array will be impacted by an *add* or *remove* operation. These
32 operations do not affect the uncertainty of uncertain arrays that respect the second
1 definition, if we consider the operation as certain. But it brings open questions for

the two other definitions. As they will modify the size of the collection, the size will be, in the best cases, as uncertain it was or more uncertain.

Lastly, one may investigate the impact of introducing uncertain data structures in common algorithms. For example: how to sort an uncertain collection? How to balance an uncertain tree? How to compute the shortest path? Let us details the sorting of an array. If the full collection is uncertain, a common sorting algorithm can be applied without impacting the uncertainty. In the other cases, algorithms should be redesigned. Indeed, as uncertainty propagation is mapped to operators, and each algorithm will use the comparison operators differently, the result can differ. Sorting the first integer array ([54, 10, 3, 24]), merge sort makes five comparisons while quick sort makes 4. These algorithms should consider uncertainty at a higher level than language operators.

Quantification and reduction of uncertainty In our work on uncertainty, we focus on the definition of uncertain data and the propagation of uncertainty. However, while talking about uncertainty, there are two other key concepts: quantification and reduction. The former is to quantify the uncertainty at runtime of a received data, *e.g.*, to find the appropriated probability distribution with its parameter. For example, when a smart grid system receives a consumption value, the quantification consists in adding the probability distribution that represents the uncertainty. The latter is to increase the confidence of received values. For instance, by comparing the consumption value with past ones, one may increase its confidence in the value. The literature provides techniques to evaluate the uncertainty [WKE08; Met08] and to reduce it [Sha76]. Thus, we think that future work for the modelling community are open. A direction could focus on a model that abstracts these techniques to help engineers to manipulate and integrate them into their models. For example, using a descriptive language that adds meta-information to uncertain values. Below, we show a possible language for that.

```
uatt att1: double {  
  min 0  
  max 100  
  past "GaussianMixtureModel"  
  timeValidity 5seconds  
  consistentWith att2[att1 ~ 2 * att2]  
}
```

Listing 8.2: Example of a descriptive language to define meta-information uncertainty

3 This language can define some information that can be used to quantify or
 4 reduce uncertainty. For example, minimal and maximal values can be used to check
 5 if the received value is in this range or not. If not, the uncertainty will increase.
 6 The *past* attribute can be used to define how it should be represented. The model
 7 will then be used to compare the received values with past ones. *TimeValidity* is
 8 here to help the quantification of the temporal uncertainty. More the value reaches
 9 the given validity, more it is uncertain. Finally, *consistentWith* can be used to
 10 compare the value with another one, here *att2*. Plus, the function given into square
 11 brackets can be used to formalise the relation between the two attributes. In the
 12 listing, *att1* is approximately equal to the double of *att2*. This language results
 13 from a first brainstorming, but research efforts are still required to design and
 14 validate it.

15 **Data uncertainty representation** Researchers defined different strategies, like
 16 keeping multiple possibilities, to represent several kinds of uncertainties. As
 17 examples, we identified the uncertainty of existence and the temporal uncertainty.
 18 The first one corresponds to the confidence that a value exists or not. It can result
 19 from faulty data sources that send wrong data. The second one can be used to
 20 represent the loss of confidence in value over time.

21 First future work would be to investigate how to introduce such techniques
 22 inside a programming language. For example, the language should be able to
 23 store and manipulate different values for a same attribute. Another example, the
 24 language should support a 3 dimensional representation of the value: one for the
 25 possible values, one for the confidence, and one for the time. Then, research efforts
 26 can focus on the definition of a language that uses different strategies to handle
 27 uncertainty. One may study the impacts on the type system, the semantics, and
 28 the syntax of the language.

29 Second, introducing new probability distributions lead to complex combinations
 30 of probability distributions. In the current approach, we use an analytical approach
 1 (we compute the exact solution). However, this cannot be performed between

some probability distributions. In such a situation, a numerical method should be applied. This leads to challenges regarding the threshold between the performance of the language and the accuracy of the method.

8.2.2 (Self-)Explanation of adaptive systems

The complexity of adaptive system behaviour makes it hard to understand system behaviour and adaptation decisions made. If the system enters in a suboptimal state, engineers may have difficulties to figure out the reasons. In this thesis, we define a temporal knowledge model to help them to perform diagnosis routines (*cf.* Chapter 7).

However, this work is just one step towards fully explainable adaptive systems. To achieve this goal, research efforts should be made to answer the following challenge: how to enable autonomous explanation of system behaviour? We identified several sub-challenges that should be addressed first:

- How to accurately link decisions with their impacts?
- How to deal with a large amount of data required for diagnosis purpose?
- How to autonomously extract explanations from the data model?

Link effects to decisions When we defined our temporal knowledge model (*cf.* Chapter 7), we assumed that designers can link actions with their expected effects at design time with the most thorough confidence. However, as systems are more and more complex, we think that they do not know all the impacts in advance. There is an unknown part. As done by Donald Rumsfeld in a famous US Department of Defense press briefing, we can identify two levels of unknowns: unknown knowns, and unknown unknowns. Moreover, these relations are uncertain. All traceability links that we model in this thesis should not be considered as entirely accurate. Lastly, decision effects will be combined with the evolution of the system context and behaviour.

Research efforts are, therefore, needed to define techniques to discover these unknown relations and to separate decisions effects from the evolution of the system. One approach could be to use a learning algorithm to find them. Besides, research efforts are still required to combine our two contributions and achieve our vision of an uncertain and time-aware modelling framework.

2 **Large amount of data** We are conscious that the metamodel defined in Chap-
3 ter 7 remains large. A model that conforms this metamodel can be challenging to
4 be defined and manipulated manually.

5 Research efforts are required to define DSL to facilitate the manipulation of
6 the model. Moreover, autonomous processes can be defined to populate a model
7 conforms to our metamodel automatically. For example, one can specify a method
8 for analysing the code or the model that describes actions to populate the temporal
9 knowledge with long-term actions. Another process can be formalised to add model
10 elements related to context information, the status of action executions, *etc.*.

11 **Automatic extraction of explanation** In the contribution presented in Chap-
12 ter 7, we defined a data model. One can navigate in this structure to identify
13 the causes of a suboptimal decision. However, this task remains manual, and we
14 strongly think that it can be long, redundant, and error-prone. Research efforts
15 should thus be performed to define an algorithm(s) that autonomously extracts
16 explanations.

17 On a request, from a human or another system, the algorithm should collect all
18 the information required to establish an explanation. The algorithm should put
19 this information in a human-readable format. We can distinguish different kinds of
20 information:

- 21 • Information that explains why the system is defined as suboptimal. Here, the
22 algorithm can compare the actual state with the requirements of the systems.
- 23 • Information that determines if the system has ended in this state because
24 of decisions made or because of the evolution of its context and behaviour.
25 To achieve this, the algorithm should be able to separate decisions effects
26 from evolution impacts. The algorithm requires thus accurate links between
27 decisions and their effects. Moreover, it needs to know how the system evolves
28 thanks to, for example, learning techniques or function that encodes it.
- 29 • Information that represents the circumstances of previous decisions. It can
30 be collected by navigating our temporal data model.
- 31 • Information that details past decisions made and their execution status. If
32 some actions have failed, data about the reasons for their failure should also
1 be collected.

2 If one can achieve this research direction, we think that other research directions
3 will be opened. A solution could analyse the explanations to detect any sub-optimal
4 decisions and suggest a set of correctives. This solution would integrate learning
1 mechanisms.

Glossary

adaptive system In this document, we modified the definition of self-adaptive systems provided by Cheng *et al.*, in [CdLG⁺09]. Adaptive systems are able to have their behaviour adjusted in response to the perception of the environment and the system themselves. If a system perform this adjustment on itself, the literature refers to it as self-adaptive system.

action In this document, we use the definition provided by IEEE Standards [III17]: “Process that, given the context and requirements as input, adjusts the system behaviour”.

behaviour We refer to system behaviour.

circumstance State of the knowledge when a decision has been taken.

context In this document, we use the definition provided by Anind K. Dey [Dey01]: “Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and [the system], including the user and [the system] themselves”.

data uncertainty Data are uncertain when some data points are not precisely known..

decision A set of actions taken after comparing the state of the knowledge with the requirement.

DSL In this document, we use the definition provided by Deursen *et al.*, [vDKV00]: “is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.”.

2 **environment** See system environment.

3 **execution** In this document, we use the definition provided in the UML specifi-
 4 cation [OMG17]: “An execution is a performance of a set of actions (potentially
 5 over some period of time) that may generate and respond to occurrences of events,
 6 including accessing and changing the state of [the system]”.

7 **knowledge** The knowledge of an adaptive system gathers information about the
 8 context, actions and requirements.

9 **long-term action** An action that is not immediate, or that takes time to be
 10 executed, or that has long-term effects..

11 **MAPE-k** A theoretical model of the adaptation process proposed by Kephart
 12 and Chess [KC03]. It divides the process in four stages: monitoring, analysing,
 13 planning and executing. These four stages share a knowledge.

14 **metamodel** In this document, we use the definition provided by Douglas C.
 15 Schmidt [Sch06]: “[Metamodels] define the relationships among concepts in a
 16 domain and precisely specify the key semantics and constraints associated with
 17 these domain concepts”.

18 **model** In this document, we use the definition provided by Brambilla *et al.*,
 19 [BCW17]: “[A model is] a simplified or partial representation of reality, defined
 20 in order to accomplish a task or to reach an agreement on a topic.” The model
 21 should conform to a metamodel: each element of the model instantiates one from
 22 the metamodel and satisfies all semantics rules [BJT05].

23 **models@run.time** In this document, we use the definition provided by Blair *et*
 24 *al.*, [BBF09]: “A model@run.time is a causally connected self-representation of the
 25 associated system that emphasises the structure, behaviour, or goals of the system
 26 from a problem space perspective”.

27 **requirement** In this document, we use the definition provided by IEEE Stan-
 28 dards [III17]: “(1) Statement that translates or expresses a need and its associated
 29 constraints and conditions, (2) Condition or capability that must be met or pos-
 30 sessed by a system [...] to satisfy an agreement, standard, specification, or other
 31 formally imposed documents”.

1 **self-adaptive system** See adaptive system.

2 **self-healing** Refers to the capacity of detecting, diagnosing, and repairing any
3 error in the system. See self-healing system.

4 **self-healing system** In this document, we use the definition provided by Kephart
5 and Chess [KC03]: “[A self-healing] system automatically detects, diagnoses, and
6 repairs localised software and hardware problems.”.

7 **SLE** In this document, we use the definition provided by Anneke Kleppe [Kle08]:
8 “the application of systematic, disciplined, and measurable approaches to the
9 development, use, deployment, and maintenance of software languages”.

10 **smart grid** In this document, we use the definition provided by the National
11 Institute of Standards and Technology (NIST) [oSN]: “a modernized grid that
12 enables bidirectional flows of energy and uses two-way communication and control
13 capabilities that will lead to an array of new functionalities and applications.”.

14 **software language** In this document, we use the definition provided by Anneke
15 Kleppe [Kle08]: “any language that is created to describe and create software
16 systems”.

1 **structure** See system structure.

2 Abbreviations

API Application Programming Interface.

CPS Cyber-Physical System.

CPU Central Processing Unit.

CSV Comma-separated values.

DOM Document Object Model.

DSL Domain-Specific Language.

DSML Domain Specific Modelling Language.

E-MOF Essential MOF (EMOF).

EMF Eclipse Modelling Framework.

FSM Final State Machine.

GCM GreyCat Modelling Environment.

GPL General Purpose Language.

GUM Guide to the expression of Uncertainty in Measurement (GUM).

ICT Information and communication technology.

IoT Internet of Things.

KMF Kevoree Modelling Framework.

MAPE-k Monitor, Analyse, Plan, and Execute over knowledge.

MDE Model-Driven Engineering.

MOF Meta Object Facility.

OCL Object Constraint Language.

- ² **OMG** Object Management Group.
- ³ **SAS** Self-Adaptive System.
- ⁴ **SLE** Software Language Engineering.
- ⁵ **UML** Unified Modelling Language.
- ¹ **XMI** XML Metadata Interchange.

List of publications and tools

Papers included in the dissertation

- 2017

- Amine Benelallam, Thomas Hartmann, Ludovic Mouline, François Fouquet, Johann Bourcier, Olivier Barais, and Yves Le Traon. Raising time awareness in model-driven engineering: vision paper. In *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS*, 2017. URL: <https://doi.org/10.1109/MODELS.2017.11>

- 2018

- Ludovic Mouline, Amine Benelallam, François Fouquet, Johann Bourcier, and Olivier Barais. A temporal model for interactive diagnosis of adaptive systems. In *2018 IEEE International Conference on Autonomic Computing, ICAC*, 2018. URL: <https://doi.org/10.1109/ICAC.2018.00029>
- Ludovic Mouline, Amine Benelallam, Thomas Hartmann, François Fouquet, Johann Bourcier, Brice Morin, and Olivier Barais. Enabling temporal-aware contexts for adaptative distributed systems. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC*, 2018. URL: <https://doi.org/10.1145/3167132.3167286>

- in the process of submission

- Ludovic Mouline, Amine Benelallam, Thomas Hartmann, Johann Bourcier, Olivier Barais, and Maxime Cordy. Ain'tea: managing data uncertainty at the language level. *Forthcoming*, forthcoming

Papers not included in the dissertation

- 2 • 2017
- 3 – Ludovic Mouline, Thomas Hartmann, François Fouquet, Yves Le Traon,
- 4 Johann Bourcier, and Olivier Barais. Weaving rules into models@run.time
- 5 for embedded smart systems. In *Companion to the first International*
- 6 *Conference on the Art, Science and Engineering of Programming, Pro-*
- 7 *gramming*, 2017. URL: <https://doi.org/10.1145/3079368.3079394>
- 8 • 2018
- 9 – Alejandro Sánchez Guinea, Andrey Boytsov, Ludovic Mouline, and
- 10 Yves Le Traon. Continuous identification in smart environments using
- 11 wrist-worn inertial sensors. In *Proceedings of the 15th EAI International*
- 12 *Conference on Mobile and Ubiquitous Systems: Computing, Networking*
- 13 *and Services, MobiQuitous*, 2018. URL: [https://doi.org/10.1145/](https://doi.org/10.1145/3286978.3287001)
- 14 3286978.3287001
- 15 **Tools included in the dissertation**
- 16 • Ain'tea: a language which integrated data uncertainty as a first-class citizen
- 17 – <https://github.com/lmouline/aintea>
- 18 • LDAS: a metamodel of knowledge for adaptive systems
- 1 – <https://github.com/lmouline/LDAS>

List of figures

1.1	Overview of the models@run.time and focus of the thesis	5
1.2	Illustration of the problem due to data uncertainty	6
1.3	Illustration of a long-term action	7
1.4	Overview of the language proposed, Ain'tea	12
1.5	Overview of the temporal knowledge model	12
1.6	Structure of the document	14
2.1	Conceptual vision of adaptive system (based on [Wey19])	16
2.2	MAPE-k loop (based on [KC03])	18
2.3	Difference and relation between metamodel and model	25
2.4	Composition of a software language	28
2.5	Probability distributions used in this thesis	32
3.1	Two different electric flows for a same grid topology	36
3.2	Example of consumption measurement before and after a limitation of amps has been executed at t_{20}	44
3.3	Figure extracted from [WBR11]. The red bar depicted the moment when Replica 2 stop receiving new connections. The green one represents the moment where all the rules in the load balancer stop considering R2. Despite these two actions, the throughput of the machine does not drop to 0 due to existing and active connections.	45
3.4	Simplified version of a smart grid	48
5.1	Overview of our vision	83

2	6.1	Impact of having uncertainty as a first-class language citizen on a	
3		language	88
4	6.2	Global architecture of Ain'tea	107
5	6.3	Detection of a type error	113
6	6.4	Infer.NET detect the error at runtime.	114
7	7.1	Time definition used for the knowledge formalism	119
8	7.2	Evolution of a temporal graph over time	121
9	7.3	Application of the formalism with a temporal graph that represents	
10		the knowledge of the smart grid described in Section 3.3.2	125
11	7.4	Validity periods of consumption values and their edges to the smart	
12		meter M_2	126
13	7.5	Excerpt of the knowledge metamodel	130
14	7.6	Excerpt of the context metamodel	131
15	7.7	Requirement metamodel	131
16	7.8	Excerpt of the action metamodel	132
17	7.9	Excerpt of the knowledge object model related to our smart grid	
18		example	136
19	7.10	Experimentation results when the knowledge based size increases . .	141
20	7.11	Results of experiments when the number of traversed or input ele-	
21		ments increases	143
22	8.1	Aperçu du models@run.time et objet de la thèse	xv
23	8.2	Illustration du problème causé par l'incertitude des données . .	xvi
24	8.3	Illustration d'une action à long terme	xviii
25	8.4	Overview of the language proposed, Ain'tea	xxiii
1	8.5	Overview of the temporal knowledge model	xxiv

List of tables

2.1	Characterisation of information of the knowledge	19
4.1	Approaches to model systems' context and behaviour (RQ1.1) . . .	55
4.2	Approaches to model actions, their circumstances, and their effects (RQ1.2)	59
4.3	Approaches to reason over evolving context or behaviour (RQ1.3) .	64
4.4	Categories of uncertainty addressed by the literature (RQ2.1)	68
4.5	Approaches to model data uncertainty (RQ2.2)	70
4.6	Approaches to propagate data uncertainty (RQ2.3)	73
4.7	Approaches to reason over the uncertainty of data (RQ2.3)	73
6.1	Which distribution can be used to represent the uncertainty of which data type	95
6.2	Cast operations allowed in our language	99
6.3	Typing rules for arithmetic operations	104

2 French summary - Résumé en français

3 *La méthodologie d'ingénierie dirigée par les modèles et l'approche des*
4 *systèmes dynamiquement adaptatifs sont combinées pour relever les défis posés*
5 *par les systèmes d'aujourd'hui. Après avoir présenté ces deux techniques de*
6 *génie logiciel, nous décrivons cinq problèmes que nous avons identifiés pour*
7 *ces systèmes : l'incertitude des données, les actions avec effets à long terme, les*
8 *comportements émergents de ces systèmes, les différents rythmes d'évolution*
9 *des sous-parties du système, et la dimension temporelle de leurs structures et*
10 *comportements. Nous présentons les défis qui accompagnent ces problèmes.*
11 *Avant de décrire les deux contributions de cette thèse, nous abordons les sous-*
1 *défis abordés.*

2 8.3 Contexte

3 Les syst mes auto-adaptatifs (SAS) optimisent leurs comportements ou confi-
4 gurations au cours de l'ex cution en r ponse   une modification de leur envi-
5 ronnement ou de leurs comportements [CdLG⁺09]. Kephart et Chess [KC03] ont
6 d fini les bases de cette approche, bas e sur un livre blanc d'IBM [Com⁺06].
7 Depuis, les praticiens l'ont appliqu    diff rents domaines [GVD19] tels que
8 l'infrastructure *cloud* [JG17; Tea15; BKF⁺17] ou les Syst me Cyber-Physique
9 (CPS) [LGC17; FMF⁺12; HFK⁺14a]. Un exemple d'un tel syst me est un r seau
10 intelligent, qui utilise la capacit  d'adaptation pour s'auto-g rir de mani re
11 autonome.

12 Un r seau intelligent est un r seau  lectrique dans lequel les services publics
13 ont introduit les Technologies de l'Information et de la Communication (TIC) pour
14 faire face aux nouveaux d fis de l'approvisionnement en  lectricit  [Far10;
15 IA09; FMX⁺12]. L'une des caract ristiques requises est la capacit  d'auto-
16 gu rison. Un syst me d'auto-g rison peut r parer automatiquement tout
17 incident, logiciel ou mat riel, au moment de l'ex cution [KC03]. Par exemple,
18 un r seau intelligent peut optimiser le flux d' nergie pour faire face aux pannes
19 des transformateurs ² [FMX⁺12].

20 Le processus d'adaptation ne peut  tre r alis  que si le syst me a une
21 connaissance approfondie de la situation et du probl me. Dans ce cas, la situation
22 comprend la structure (les  l ments qui composent le syst me), le comporte-
23 ment (l'ensemble des ex cutions possibles du syst me) et l'environnement (o 
24 il est ex cut ) du syst me. Cette compr hension peut  tre extraite d'une
25 ou d'un ensemble d'abstraction(s) de ces  l ments. Les abstractions fournissent
26 une description des syst mes, de leurs comportements ou de leur environnement.
27 Par exemple, Hartmann Hartmann *et al.*, [HFK⁺14b] fournissent un diagramme de
28 classes qui d crit la topologie du r seau intelligent, lorsqu'il utilise les communi-
29 cations par lignes  lectriques³.

30 Les d fenseurs de l'Ing nierie Dirig e par les Mod les (IDM) plaident
1 pour l'utilisation du m canisme d'abstraction pour faciliter le d veloppement

²Les transformateurs modifient la tension dans les c bles.

³Les donn es sont transmises par des c bles qui distribuent  galement l' lectricit .

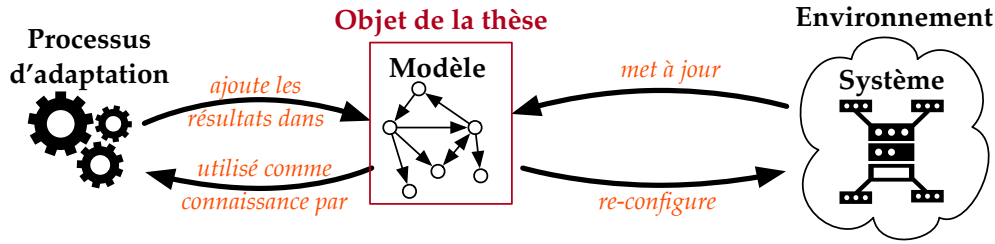


FIGURE 8.1 : Aperçu du *models@run.time* et objet de la thèse

des logiciels actuels [Sch06 ; Ken02 ; BCW17]. Cette méthodologie peut être appliquée à différentes étapes du développement logiciel. Dans cette thèse, nous nous concentrons sur l'un de ses paradigmes : *models@run.time* [BBF09 ; MBJ⁺09]. Comme nous l'illustrons par la Figure 8.1, en utilisant ce paradigme, le processus d'adaptation s'appuie sur un modèle pour analyser la situation et déclencher l'adaptation. Dans ce document, nous supposons que le modèle représente la connaissance du processus d'adaptation. Les développeurs peuvent utiliser ce paradigme pour mettre en œuvre des systèmes adaptatifs [MBJ⁺09 ; HFK⁺14a]. Cette thèse contribue à cette couche de modélisation.

8.4 Dilemmes

Au cours de notre étude, nous avons identifié cinq caractéristiques des systèmes adaptatifs qui posent des dilemmes pour la recherche en génie logiciel. Premièrement, l'information recueillie n'est pas toujours connue avec une confiance absolue. Deuxièmement, les reconfigurations peuvent ne pas être immédiates et leurs effets ne sont pas instantanément mesurés. Troisièmement, le comportement du système peut être émergent [ZA11], c'est-à-dire qu'il ne peut être entièrement connu au moment de la conception. Quatrièmement, les différentes sous-parties du système n'évoluent pas au même rythme. Cinquièmement, la structure et le comportement des systèmes ont une dimension temporelle. La dernière a été publiée dans notre document de vision concernant la prise de conscience du temps dans l'IDM [BHM⁺17]. Nous les avons détaillés dans cette section.

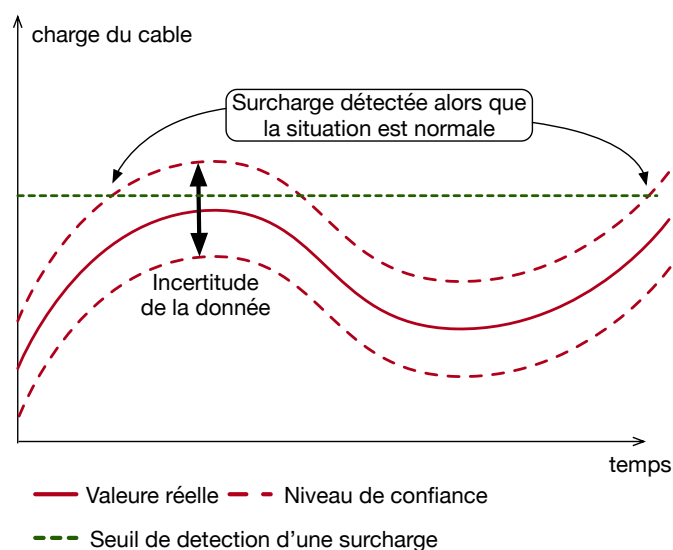


FIGURE 8.2 : Illustration du problème causé par l'incertitude des données

2 8.4.1 Ingénierie de logiciels sensibles à l'incertitude

3 La plupart des fusibles sont ouverts et fermés manuellement par les techni-
 4 ciens plutôt que modifiés automatiquement. Ensuite, les techniciens rapportent
 5 manuellement les modifications effectuées sur la grille. En raison d'erreurs hu-
 6 maines, il en résulte des erreurs. La topologie de la grille est donc incertaine. Cette
 7 incertitude se propage à l'approximation de la charge, utilisée pour détecter
 8 les surcharges dans le réseau. De mauvaises reconfigurations pourraient être
 9 déclenchées, ce qui pourrait être encore pire que si aucun changement n'avait
 10 été appliqué.

11 Plus généralement, les données sont, presque par définition, incertaines et
 12 les développeurs travaillent avec des estimations dans la plupart des cas [BMM14 ;
 13 Met08 ; AY09]. L'incertitude peut s'expliquer par la façon dont les données sont
 14 recueillies. On peut distinguer trois catégories : les capteurs, les humains et les
 15 résultats des calculs. Les capteurs (logiciels ou matériels) estiment toujours la
 16 valeur et ont une valeur de précision grâce à la méthode de mesure [Met08 ;
 17 BMM14]. Les humains sont sujets à l'erreur. Les calculs peuvent soit donner une
 18 approximation, soit être basés sur des données incertaines. Cette incertitude se
 1 propage ensuite à travers toutes les étapes jusqu'au résultat final.

Pour un domaine spécifique, cette incertitude peut avoir une incidence sur la compréhension de la situation réelle, comme le montre la Figure 8.2. Par exemple, l'incertitude de l'horloge CPU est trop faible pour endommager le pourcentage de charge du processeur. Cependant, l'incertitude de la charge du câble dans un réseau intelligent peut déclencher une fausse détection d'une surcharge, comme le montre la Figure 8.2. Si l'incertitude des données peut induire en erreur la compréhension du comportement ou de l'état d'un système, les développeurs doivent mettre en œuvre un système conscient de l'incertitude. Pour les systèmes adaptatifs, ce manque de confiance peut entraîner des adaptations sous-optimales.

Par conséquent, nous soutenons que l'incertitude des données a une incidence sur toutes les étapes du développement d'un logiciel, de la conception à l'exécution. Parmi les différentes étapes de cette thèse, nous nous concentrons sur celle de la conception. Nous croyons fermement que les techniques de conception devraient fournir des mécanismes pour aider les développeurs à abstraire et à manipuler des données incertaines.

La littérature fournit des approches pour aider les ingénieurs à raisonner ou à manipuler les données sans certitude, ou du moins les distributions de probabilité. Par exemple, les fonctions de croyances [Sha76] aident à réduire cette incertitude en combinant plusieurs sources de données. La communauté de programmation probabiliste [GHN⁺14] fournit des *frameworks* et des langages de programmation [MWG⁺18; BDI⁺17] pour propager les probabilités lors des calculs.

Cependant, à notre connaissance, aucune étude n'a été effectuée pour évaluer l'impact de l'incertitude des données sur le développement des logiciels. Le défi suivant demeure une question ouverte pour la communauté du génie logiciel :

Comment concevoir des logiciels sensibles à l'incertitude (conception, mise en œuvre, test et validation) ?

8.4.2 Raisonnement sur les actions à long terme

La reconfiguration d'un réseau intelligent implique de modifier le flux de puissance en ouvrant ou en fermant les fusibles. Comme nous l'avons déjà dit, les techniciens doivent se rendre physiquement sur les lieux des fusibles pour modifier

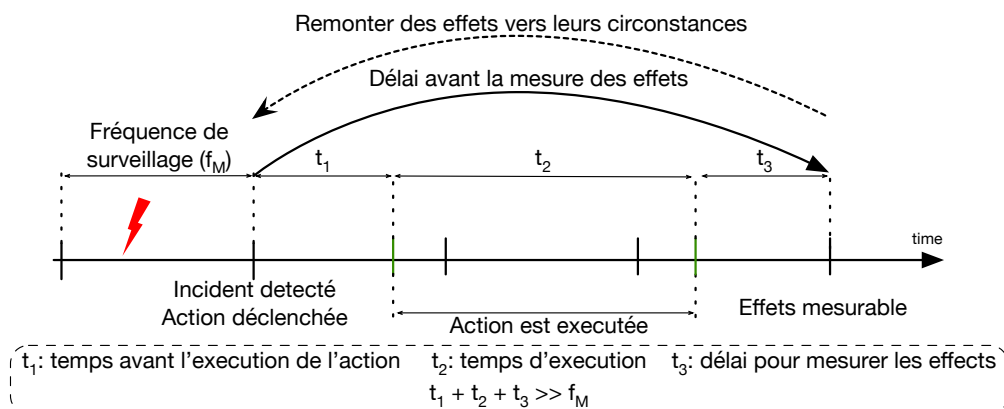


FIGURE 8.3 : Illustration d'une action à long terme

leur États. Dans le cas du réseau intelligent luxembourgeois, les compteurs envoient des mesures d'énergie toutes les 15 minutes, de manière non synchrone. Par conséquent, entre le moment où une reconfiguration du réseau intelligent est décidée et le moment où les effets sont mesurés, un retard d'au moins 15 minutes se produit. D'autre part, un incident devrait être détecté dans les 15 minutes qui suivent. Si le processus d'adaptation ne tient pas compte de cette différence de rythme, il peut entraîner des décisions irréversibles.

De façon plus générale, il peut exister une différence entre la fréquence de surveillance et le temps nécessaire pour mesurer les effets de l'action. L'une des causes de cette différence est ce que nous appelons les actions à long terme dans ce document, illustrées à la Figure 8.3. Une action à long terme est définie comme une action qui prend du temps à être exécutée (retard à exécuter et temps d'exécution) ou qui a des effets à long terme. Une deuxième cause est l'impossibilité de réduire la fréquence de surveillance puisque les systèmes doivent être réactifs dans certains cas. Cette différence de taux peut nuire au processus décisionnel.

Par conséquent, nous soutenons que les processus décisionnels devraient tenir compte de ce délai si la fréquence de l'étape de surveillance est inférieure à la durée des effets de l'action pour pouvoir être mesurée. A notre connaissance, aucune des approches ne permet aux développeurs d'implémenter de tels outils. L'un des défis ouverts pour la communauté des chercheurs est donc le suivant :

Comment modéliser, stocker et interroger les actions à long terme avec leurs effets ?

8.4.3 Diagnostic du processus d'adaptation

Le comportement du réseau intelligent est affecté par plusieurs facteurs qui ne peuvent pas être contrôlés par le gestionnaire de réseau. Les conditions météorologiques en sont un exemple. Les réseaux intelligents reposent sur une production d'énergie répartie entre plusieurs acteurs. Par exemple, les utilisateurs, qui étaient auparavant principalement des consommateurs, peuvent maintenant produire de l'énergie en ajoutant des panneaux solaires sur le toit de leur maison. La production de cette énergie dépend de la météo, et même de la saison⁴. Malgré cette stochasticité du comportement, les ingénieurs doivent mettre en œuvre un processus d'adaptation qui peut conduire à une configuration de grille sous-optimale.

Face à des systèmes logiciels de plus en plus complexes et à grande échelle (par exemple les systèmes de réseaux intelligents), nous pouvons tous convenir que la présence de défauts résiduels devient inévitable [BdMM⁺17; MPS15; HBB15]. Même avec un processus itératif de vérification ou de validation, il est très probable de se heurter à un comportement inattendu qui n'était pas prévu au moment de la conception. À elles seules, les approches formelles de modélisation et de vérification existantes peuvent ne pas être suffisantes pour anticiper ces défaillances [TOH17]. À ce titre, des techniques complémentaires doivent être proposées pour localiser le comportement anormal et son origine afin de le manipuler en toute sécurité.

Bencomo *et al.*, [BWS⁺12] soutiennent qu'une explication complète du comportement du système contribue de manière drastique à la qualité du diagnostic et facilite la tâche de débogage du comportement du système. Pour ce faire, comme le montre la Figure 8.3, nous pensons que les systèmes logiciels adaptatifs devraient être dotés d'installations de gestion de la traçabilité pour relier les décisions prises à (i) leur situation, c'est-à-dire l'historique des états du système et les exigences visées, et (ii) les actions réalisées et leur(s) impact(s)

⁴L'angle du soleil a un impact sur la quantité d'énergie produite par les panneaux solaires. Cet angle varie selon la saison.

2 sur le système. En particulier, un système adaptatif devrait conserver une trace
3 des événements historiques pertinents. En outre, il devrait être en mesure de
4 lier les objectifs que le système entend atteindre aux adaptations et aux décisions
5 qui ont été prises, et vice versa. Enfin, afin de permettre aux développeurs
6 d'interagir avec le système d'une manière claire et compréhensible, une abstraction
7 appropriée pour permettre la navigation des traces et leur historique devrait
8 également être fournie. En d'autres termes, un défi qui n'a pas encore été
10 relevé est le suivant :

11

Comment faire remonter les effets des décisions d'adaptation à leur situation ?

12 8.4.4 Modélisation des états incohérents des systèmes

13 Chaque compteur envoie des données de consommation et de production toutes
14 les 15 minutes. Cependant, cette collection n'est pas synchrone. Autrement dit,
15 tous les compteurs n'envoient pas leurs données au même moment. Le système
16 global, qui reçoit toutes les données, n'a donc pas une vision globale avec la
17 même fraîcheur pour toute la partie de la grille. Les données électriques
18 sont volatiles : un pic ou une baisse peut survenir en moins d'une minute en
19 raison, par exemple, du démarrage ou de la finition d'une machine à laver. La
20 reconfiguration de la grille peut donc être sous-optimale en raison d'informations
21 périmées.

22 Les différentes parties d'un système peuvent évoluer à des rythmes diffé-
23 rents. Certains systèmes sont hétérogènes en termes de matériel et de logiciels.
24 Cette diversité se traduit par des évolutions ou des taux de réaction différents.
25 Par exemple, si certains composants fonctionnent sur batteries, ils auront un cycle
26 de sommeil pour économiser de l'énergie. A l'inverse, si d'autres sont connectés
27 directement à une source d'énergie, ils peuvent réagir plus rapidement.

28 Malgré cette différence de rythmes, une vision globale d'un système à un
29 moment précis peut encore être nécessaire. La vision devrait porter sur des
30 données qui ont une fraîcheur différente. Par exemple, le processus d'adaptation
31 peut nécessiter une vision globale du système. Dans le pire des cas, certaines
32 données seraient périmées et ne pourraient être utilisées.

33 Lors de la conception du processus d'adaptation, les ingénieurs ont donc besoin
1 de solutions pour faire face à un état inconsistant du système. Une solution

peut, par exemple, estimer de façon transparente ce que devrait être la valeur actuelle de données primaires. Un défi pour la communauté du génie logiciel est donc :

Comment représenter, interroger et stocker des états et des comportements système incohérents ?

8.4.5 Modélisation des données temporelles et interconnectées

Le flux d'énergie est influencé par la consommation et la production des utilisateurs, ainsi que par les modifications de la topologie. Connaître le dernier statut de la grille est aussi important que savoir comment elle évolue. En fonction de l'évolution, l'exploitant du réseau peut prévoir tout incident futur, comme une surcharge. Il pourrait également comparer cette évolution de comportement avec une évolution normale pour détecter, par exemple, tout comportement malveillant.

L'évolution des systèmes est intrinsèquement liée à une dimension temporelle. Evolution et temps sont deux concepts liés. Pour certains systèmes, non seulement les derniers états sont importants, mais aussi leur évolution. Ensuite, les processus d'analyse examineront cette évolution pour savoir si elle est normale ou non. Ils peuvent aussi utiliser cette évolution pour prédire comment les systèmes vont évoluer. En se basant sur ces prévisions, ils peuvent prévoir les incidents futurs dans le système.

Les décisions ne sont pas prises en fonction du dernier état du système, mais en fonction de son évolution. Le processus d'analyse doit donc naviguer à la fois dans la structure du système et dans son comportement dans le temps. Les ingénieurs ont besoin d'outils efficaces pour structurer, représenter, interroger et stocker des données temporelles et interconnectées à grande échelle.

Le temps n'est pas un nouveau défi pour le génie logiciel. Par exemple, Riviera *et al.*, [RRV08] ont déjà identifié le temps comme un défi pour la communauté de l'IDM. Différentes approches ont été définies [BCC⁺15; KT12; KH10; HFN⁺14b].

Cependant, nous remarquons que les efforts de recherche déployés par la communauté des EMM n'étaient pas axés sur la modélisation, la persistance

2 et le traitement des donn  es en   volution. Thomas Hartmann a commenc     
3 aborder ces d  fis dans sa th  se de doctorat [Har16]. Le dernier d  fi, qui n'est
4 pas enti  rement relev  , est donc :

6 Comment structurer, repr  senter, requ  ter et stocker efficacement des
donn  es temporelles    grande   chelle ?

7 8.5 P  rim  tre de la th  se

8 Parmi tous les d  fis d  crits dans la section pr  c  dente, cette th  se se
9 concentre sur trois d'entre eux : l'incertitude des donn  es (Section 8.4.2), les
10 actions    long terme (Section 8.4.3) et le processus d'adaptation sujet aux erreurs
11 (Section 8.4.3). Plus pr  cis  ment, nous abordons trois sous-probl  mes de ces
12 d  fis.

13 La gestion de l'incertitude exige une grande expertise en probabilit   et en
14 th  orie statistique. La litt  rature propose diff  rentes solutions pour g  rer
15 l'incertitude [Zad ; Met08 ; Sha76]. L'application de ces techniques exige une
16 compr  hension approfondie des th  ories sous-jacentes et prend beaucoup de
17 temps [VMO16]. De plus, il est difficile    tester et peut-  tre plus important
18 encore, tr  s sujet aux erreurs. Dans cette th  se, nous abordons donc le probl  me
19 suivant :

21 **Sous-d  fi #1:** Comment faciliter la manipulation de l'incertitude des donn  es
pour les ing  nieurs logiciels ?

22 Les processus d'adaptation peuvent reposer sur des actions    long terme
23 comme la migration des ressources dans l'infrastructure *cloud*. A cause du manque
24 d'information sur les actions inachev  es et leurs effets pr  vus sur le syst  me,
25 la composante raisonnement peut prendre des d  cisions r  p  t  es ou sous-
26 optimales. Une   tape pour permettre ce m  canisme de raisonnement est d'avoir
27 une couche d'abstraction qui peut repr  senter efficacement ces actions    long
28 terme. Dans cette th  se, nous relevons donc le d  fi suivant :

30 **Sous-d  fi #2:** Comment permettre de raisonner sur les actions inachev  es
et leurs effets attendus ?

31 En raison de la complexit   croissante des syst  mes, les d  veloppeurs ont
1 des difficult  s    livrer des logiciels sans erreur [BdMM⁺17 ; MPS15 ; HBB15].

```

Gaussian<double> average(Gaussian<double>[] arrays) {
    Gaussian<int> sum = arrays[0];
    for(int i = 0; i < arrays.length; i++) {
        sum = sum + arrays[i];
    }
    return sum / arrays.length;
}

```

Gaussian distribution can only be applied on (float, double). Actual: int

FIGURE 8.4 : Overview of the language proposed, Ain'tea

De plus, les systèmes complexes ou les systèmes à grande échelle peuvent avoir des comportements émergents. Les systèmes ont très probablement un comportement anormal qui n'était pas prévu au moment de la conception. Les approches formelles de modélisation et de vérification existantes peuvent ne pas être suffisantes pour vérifier et valider de tels processus [TOH17]. Dans de telles situations, les développeurs appliquent habituellement des routines de diagnostic pour identifier les causes des défaillances. Au cours de nos études, nous relevons le défi suivant :

Sous-défi #3: Comment modéliser les décisions d'un processus d'adaptation pour le diagnostiquer ?

8.6 Contribution & validation

Dans cette thèse, nous soutenons que les cadres de modélisation modernes devraient considérer l'incertitude et le temps comme des concepts de première classe. Dans ce mémoire, je présente deux contributions qui appuient cette vision. Tout d'abord, nous définissons une langue avec incertitude chez un citoyen de première classe : Ain'tea. Deuxièmement, nous définissons un métamodèle, et nous le formalisons, de la connaissance des systèmes adaptatifs.

Ain'tea : Gérer l'incertitude des données au niveau du langage

Cette contribution aborde le défi de la manipulation des données incertaines (cf. Sous-défi #1). Nous proposons Ain'tea, un langage capable de représenter des données incertaines comme des types de données intégrés au langage avec leurs opérations supportées. La Figure 8.4 donne un aperçu du langage utilisé. Il contient un échantillon des lois de probabilités (Gaussiennes, Bernoulli, binomiales, Rayleigh, et la fonction delta de Dirac) qui couvre les différents types

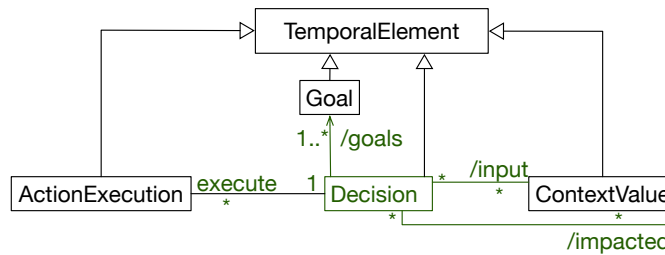


FIGURE 8.5 : Overview of the temporal knowledge model

de données (booléens, nombres et références). Nous avons implémenté un prototype du langage, disponible publiquement sur GitHub⁵. Nous utilisons une étude de cas basée sur un réseau intelligent, réalisée avec notre partenaire Creos S.A.. L'évaluation montre d'abord que notre approche n'a pas d'impact sur la concision de la langue. Deuxièmement, elle souligne la faisabilité et les avantages des systèmes de contrôle de type sensibles à l'incertitude au niveau linguistique.

Cette contribution a été soumise au journal JOT⁶ :

- “Ain'tea : Managing Data Uncertainty at the Language Level”, MOULINE, BENELALLAM, HARTMANN, BOURCIER, BARAIS et CORDY

Un métamodèle temporel de la connaissance Cette contribution aborde le défi du raisonnement sur les actions inachevées et de la compréhension du comportement des systèmes adaptatifs (cf. Sous-chapitre #2 et #3). Tout d'abord, nous formalisons les concepts de base communs impliqués dans les processus d'adaptation, également appelés connaissances. La formalisation est basée sur des graphes temporels et un ensemble de relations qui retracent l'impact des décisions sur les circonstances. Deuxièmement, nous proposons un *framework* pour structurer et stocker l'état et le comportement d'un système adaptatif en fonctionnement, ainsi qu'une API de haut niveau pour effectuer efficacement des routines de diagnostic. Notre *framework* s'appuie sur une solution basée sur un modèle temporel qui résume efficacement les décisions, leurs circonstances correspondantes et leurs effets. Nous donnons un aperçu du métamodèle dans la Figure 8.5. Nous démontrons l'applicabilité de notre approche en l'appliquant

⁵<https://github.com/lmouline/aintea/>

⁶<http://www.jot.fm/>

2 C'est un exemple basique sur un réseau intelligent. Nous montrons également que
3 notre approche peut être utilisée pour diagnostiquer le comportement d'au plus
4 les cinq derniers jours d'une période de sur le réseau intelligent luxembourgeois
5 en ~ 2.4 secondes.

6 Une partie de cette contribution a été publiée à la conférence inter-
7 nationale de l'IEEE. sur l'informatique autonome⁷ (ICAC) et au Symposium
8 ACM/SIGAPP sur l'informatique appliquée⁸ (SAC) :

- 9 • “Enabling temporal-aware contexts for adaptive distributed systems”, MOU-
10 LINE, BENELALLAM, HARTMANN, FOUQUET, BOURCIER, MORIN et BARAIS
- 11 • “A Temporal Model for Interactive Diagnosis of Adaptive Systems”, MOULINE,
1 BENELALLAM, FOUQUET, BOURCIER et BARAIS

⁷<http://icac2018.informatik.uni-wuerzburg.de/>

⁸<http://www.sigapp.org/sac/sac2018/>

Bibliography

- [AdLM⁺09] Jesper Andersson, Rogério de Lemos, Sam Malek, and Danny Weyns. Reflecting on self-adaptive software systems. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2009. URL: <https://doi.org/10.1109/SEAMS.2009.5069072> (cited on page 16).
- [AFR⁺10] Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic, and Rudi Studer. A rule-based language for complex event processing and reasoning. In *Web Reasoning and Rule Systems RR*, 2010. URL: https://doi.org/10.1007/978-3-642-15918-3%5C_5 (cited on pages 64, 65).
- [AGR11] Paolo Arcaini, Angelo Gargantini, and Elvinia Riccobene. CoMA: conformance monitoring of java programs by abstract state machines. In *Runtime Verification RV*, 2011. URL: https://doi.org/10.1007/978-3-642-29860-8_17 (cited on pages 55, 57).
- [AK03] Colin Atkinson and Thomas Kühne. Model-driven development: a metamodeling foundation. *IEEE Software*, 2003. URL: <https://doi.org/10.1109/MS.2003.1231149> (cited on page 23).
- [ARS15] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. Modeling and analyzing MAPE-K feedback loops for self-adaptation. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2015. URL: <https://doi.org/10.1109/SEAMS.2015.10> (cited on pages 55, 57, 59, 60, 63, 64).

- 3 [AY09] Charu C. Aggarwal and Philip S. Yu. A survey of uncertain data
4 algorithms and applications. *IEEE Transactions on Knowledge and*
5 *Data Engineering*, 2009. URL: [https://doi.org/10.1109/TKDE.](https://doi.org/10.1109/TKDE.2008.190)
6 2008.190 (cited on pages 5, xvi).
- 7 [BAV⁺12] Sooraj Bhat, Ashish Agarwal, Richard W. Vuduc, and Alexander G.
8 Gray. A type theory for probability density functions. In *Proceedings*
9 *of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of*
10 *Programming Languages, POPL*, 2012. URL: [https://doi.org/10.](https://doi.org/10.1145/2103656.2103721)
11 1145/2103656.2103721 (cited on pages 67, 68, 70, 73, 74, 76, 77).
- 12 [BBF09] Gordon S. Blair, Nelly Bencomo, and Robert B. France. Mod-
13 els@run.time. *IEEE Computer*, 2009. URL: [https://doi.org/](https://doi.org/10.1109/MC.2009.326)
14 10.1109/MC.2009.326 (cited on pages 4, 19, 47, 54, 55, 63, 64, ii,
15 xv).
- 16 [BBG⁺13] Nelly Bencomo, Amel Bennaceur, Paul Grace, Gordon S. Blair, and
17 Valérie Issarny. The role of models@run.time in supporting on-the-
18 fly interoperability. *Computing*, 2013. URL: [https://doi.org/10.](https://doi.org/10.1007/s00607-012-0224-x)
19 1007/s00607-012-0224-x (cited on pages 55, 57).
- 20 [BBH⁺10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indul-
21 ska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A
22 survey of context modelling and reasoning techniques. *Pervasive and*
23 *Mobile Computing*, 2010. URL: [https://doi.org/10.1016/j.pmcj.](https://doi.org/10.1016/j.pmcj.2009.06.002)
24 2009.06.002 (cited on pages 20, 21).
- 25 [BBM⁺18] Loli Burgueño, Manuel F. Bertoa, Nathalie Moreno, and Antonio Val-
26 lecillo. Expressing confidence in models and in model transformation
27 elements. In *Proceedings of the 21th ACM/IEEE International Con-*
28 *ference on Model Driven Engineering Languages and Systems, MOD-*
29 *ELS*, 2018. URL: <https://doi.org/10.1145/3239372.3239394>
30 (cited on pages 67–70, 73, 74, 76, 77).
- 31 [BCC⁺15] Erwan Bousse, Jonathan Corley, Benoît Combemale, Jeffrey G. Gray,
1 and Benoit Baudry. Supporting efficient and advanced omniscient
2 debugging for xdsmls. In *Proceedings of the 2015 ACM SIGPLAN*

International Conference on Software Language Engineering, SLE, 2015. URL: <https://dl.acm.org/citation.cfm?id=2814262> (cited on pages 10, xxi).

[BCG⁺12] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. Modelling and analyzing adaptive self-assembly strategies with maude. In *Rewriting Logic and Its Applications WRLA 2012*, 2012. URL: https://doi.org/10.1007/978-3-642-34005-5_7 (cited on pages 59, 60).

[BCW17] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice, Second Edition*. 2017. URL: <https://doi.org/10.2200/S00751ED2V01Y201701SWE004> (cited on pages 4, 23, ii, xv).

[BDI⁺17] Michaël Baudin, Anne Dutfoy, Bertrand Iooss, and Anne-Laure Popelin. OpenTURNS: an industrial software for uncertainty quantification in simulation. *Handbook of uncertainty quantification*, 2017. URL: https://doi.org/10.1007/978-3-319-12385-1_64 (cited on pages 7, 67, 68, 70, 73, 74, 76, 77, 106, xvii).

[BdMM⁺17] Davi Monteiro Barbosa, Rômulo Gadelha de Moura Lima, Paulo Henrique Mendes Maia, and Evilásio Costa Junior. Lotus@runtime: a tool for runtime monitoring and verification of self-adaptive systems. In *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2017. URL: <https://doi.org/10.1109/SEAMS.2017.18> (cited on pages 8, 11, 55, 57, 59, 60, 63, 64, xix, xxii).

[BGG⁺13] Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. Measure transformer semantics for bayesian machine learning. *Logical Methods in Computer Science*, 2013. URL: [https://doi.org/10.2168/LMCS-9\(3:11\)2013](https://doi.org/10.2168/LMCS-9(3:11)2013) (cited on pages 67, 68, 70, 73, 74, 76, 77).

[BGP92] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Transactions on Knowledge*

- 4 *and Data Engineering*, 1992. URL: [https://doi.org/10.1109/69.](https://doi.org/10.1109/69.166990)
5 166990 (cited on pages 67–70, 73, 76, 77).
- 6 [BGS⁺14] Amine Benelallam, Abel Gómez, Gerson Sunyé, Massimo Tisi, and
7 David Launay. Neo4EMF, a scalable persistence layer for EMF
8 models. In *Modelling Foundations and Applications ECMFA*, 2014.
9 URL: https://doi.org/10.1007/978-3-319-09195-2%5C_15
10 (cited on page 24).
- 11 [BHH05] George E. P. Box, J. Stuart Hunter, and William G. Hunter. *Statistics*
12 *for Experiments: Design, Innovation, and Discovery, 2nd Edition*.
13 Wiley-Interscience, 2005. ISBN: 978-0-471-71813-0 (cited on page 32).
- 14 [BHL⁺01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic
15 web. *Scientific american*, 2001. URL: [https://doi.org/10.1038/](https://doi.org/10.1038/scientificamerican0501-34)
16 [scientificamerican0501-34](https://doi.org/10.1038/scientificamerican0501-34) (cited on page 23).
- 17 [BHM⁺17] Amine Benelallam, Thomas Hartmann, Ludovic Mouline, François
18 Fouquet, Johann Bourcier, Olivier Barais, and Yves Le Traon. Raising
19 time awareness in model-driven engineering: vision paper. In *20th*
20 *ACM/IEEE International Conference on Model Driven Engineering*
21 *Languages and Systems, MODELS*, 2017. URL: [https://doi.org/](https://doi.org/10.1109/MODELS.2017.11)
22 [10.1109/MODELS.2017.11](https://doi.org/10.1109/MODELS.2017.11) (cited on pages 5, vii, xv).
- 23 [BJT05] Jean Bézivin, Frédéric Jouault, and David Touzet. Principles, stan-
24 dards and tools for model engineering. In *10th International Con-*
25 *ference on Engineering of Complex Computer Systems (ICECCS)*,
26 2005. URL: <https://doi.org/10.1109/ICECCS.2005.68> (cited on
27 pages 25, ii).
- 28 [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*.
29 MIT Press, 2008. ISBN: 978-0-262-02649-9 (cited on page 24).
- 30 [BK11] Björn Bartels and Moritz Kleine. A CSP-based framework for the
1 specification, verification, and implementation of adaptive systems.
2 In *2011 ICSE Symposium on Software Engineering for Adaptive and*
3 *Self-Managing Systems, SEAMS*, 2011. URL: [https://doi.org/10.](https://doi.org/10.1145/1988008.1988030)
4 [1145/1988008.1988030](https://doi.org/10.1145/1988008.1988030) (cited on pages 55, 56, 59, 62, 64, 65).

- [BKF⁺17] Cornel Barna, Hamzeh Khazaei, Marios Fokaefs, and Marin Litoiu. Delivering elastic containerized cloud applications to enable devops. In *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2017. URL: <https://doi.org/10.1109/SEAMS.2017.12> (cited on pages 4, xiv).
- [BLC⁺18] Erwan Bousse, Dorian Leroy, Benoît Combemale, Manuel Wimmer, and Benoit Baudry. Omniscient debugging for executable dsls. *Journal of Systems and Software*, 2018. URL: <https://doi.org/10.1016/j.jss.2017.11.025> (cited on pages 25, 29).
- [BLW05] Paul Baker, Shiou Loh, and Frank Weil. Model-driven engineering in a large industrial context - motorola case study. In *Model Driven Engineering Languages and Systems, MoDELS*, 2005. URL: https://doi.org/10.1007/11557432%5C_36 (cited on pages 23, 24).
- [BMB⁺18] Manuel F. Bertoa, Nathalie Moreno, Gala Barquero, Loli Burgueño, Javier Troya, and Antonio Vallecillo. Expressing measurement uncertainty in OCL/UML datatypes. In *Modelling Foundations and Applications, ECMFA*, 2018. URL: https://doi.org/10.1007/978-3-319-92997-2_4 (cited on pages 67–70, 73, 76, 77, 89).
- [BMM14] James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. Uncertain: a first-order type for uncertain data. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS*, 2014. URL: <https://doi.org/10.1145/2541940.2541958> (cited on pages 5, 6, 67, 68, 70, 71, 73, 74, 76, 77, xvi).
- [Bor13] James Bornholt. *Abstractions and techniques for programming with uncertain data*. PhD thesis, Honors thesis, Australian National University, 2013. URL: <https://homes.cs.washington.edu/~bornholt/papers/thesis13.pdf> (cited on pages 33, 70).
- [BPG⁺04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: an agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 2004.

- 5 URL: <https://doi.org/10.1023/B:AGNT.0000018806.20944.ef>
6 (cited on page 131).
- 7 [BPS10] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. Fuzzy goals
8 for requirements-driven adaptation. In *RE, 18th IEEE International*
9 *Requirements Engineering Conference*, 2010. URL: <https://doi.org/10.1109/RE.2010.25> (cited on pages 18, 55, 57, 59, 61, 64, 66,
10 76).
11
- 12 [BSG⁺09] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger
13 Giese, Holger M. Kienle, Marin Litoiu, Hausi A. Müller, Mauro Pezzè,
14 and Mary Shaw. Engineering self-adaptive systems through feedback
15 loops. In *Software Engineering for Self-Adaptive Systems [outcome of*
16 *a Dagstuhl Seminar]*, 2009. URL: [https://doi.org/10.1007/978-](https://doi.org/10.1007/978-3-642-02161-9%5C_3)
17 [3-642-02161-9%5C_3](https://doi.org/10.1007/978-3-642-02161-9%5C_3) (cited on pages 16, 18).
- 18 [BSH⁺06] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer
19 Widom. ULDBs: databases with uncertainty and lineage. In *Proceed-*
20 *ings of the 32nd International Conference on Very Large Data Bases*,
21 2006. URL: <http://dl.acm.org/citation.cfm?id=1164209> (cited
22 on pages 67, 68, 70, 71, 76).
- 23 [BWS⁺12] Nelly Bencomo, Kristopher Welsh, Pete Sawyer, and Jon Whittle.
24 Self-explanation in adaptive systems. In *17th IEEE International*
25 *Conference on Engineering of Complex Computer Systems, ICECCS*,
26 2012. URL: [http://doi.ieeecomputersociety.org/10.1109/](http://doi.ieeecomputersociety.org/10.1109/ICECCS.2012.34)
27 [ICECCS.2012.34](http://doi.ieeecomputersociety.org/10.1109/ICECCS.2012.34) (cited on pages 8, 59, 61, xix).
- 28 [CA07] Betty H. C. Cheng and Joanne M. Atlee. Research directions in
29 requirements engineering. In *Workshop on the Future of Software*
30 *Engineering, FOSE*, 2007. URL: [https://doi.org/10.1109/FOSE.](https://doi.org/10.1109/FOSE.2007.17)
31 [2007.17](https://doi.org/10.1109/FOSE.2007.17) (cited on page 22).
- 1 [CCH⁺13] Maxime Cordy, Andreas Classen, Patrick Heymans, Axel Legay,
2 and Pierre-Yves Schobbens. Model checking adaptive software with
3 featured transition systems. In *Assurances for Self-Adaptive Systems*

- *Principles, Models, and Techniques*. 2013. URL: https://doi.org/10.1007/978-3-642-36249-1_1 (cited on pages 59, 62, 64, 66).

[CDE⁺02] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Jose F. Quesada. Maude: specification and programming in rewriting logic, 2002. URL: [https://doi.org/10.1016/S0304-3975\(01\)00359-0](https://doi.org/10.1016/S0304-3975(01)00359-0) (cited on page 56).

[CdLG⁺09] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaella Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: a research roadmap. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, 2009. URL: https://doi.org/10.1007/978-3-642-02161-9_1 (cited on pages 4, 16, i, xiv).

[CDM09] Anis Charfi, Tom Dinkelaker, and Mira Mezini. A plug-in architecture for self-adaptive web service compositions. In *IEEE International Conference on Web Services, ICWS*, 2009. URL: <https://doi.org/10.1109/ICWS.2009.125> (cited on pages 59, 61, 64, 65).

[CG12] Shang-Wen Cheng and David Garlan. Stitch: a language for architecture-based self-adaptation. *Journal of Systems and Software*, 2012. URL: <https://doi.org/10.1016/j.jss.2012.02.060> (cited on pages 59, 61, 63–65, 76, 133).

[CGF⁺09] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Autonomic computing through reuse of variability models at runtime: the case of smart homes. *IEEE Computer*, 42(10):37–43, 2009. DOI: 10.1109/MC.2009.309. URL: <https://doi.org/10.1109/MC.2009.309> (cited on page 19).

- 5 [CGH⁺17] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee,
6 Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo,
7 Peter Li, and Allen Riddell. Stan : a probabilistic programming
8 language. *Journal of Statistical Software*, 2017. ISSN: 1548-7660. URL:
9 <https://doi.org/10.18637/jss.v076.i01> (cited on pages 67, 68,
10 70, 73, 74, 76, 77).
- 11 [CGK⁺] Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska, Raffaella
12 Mirandola, and Giordano Tamburrelli. Dynamic QoS management
13 and optimization in service-based systems. *IEEE Transactions on*
14 *Software Engineering*. URL: [https://doi.org/10.1109/TSE.2010.](https://doi.org/10.1109/TSE.2010.92)
15 92 (cited on page 18).
- 16 [Che76] Peter P. Chen. The entity-relationship model - toward a unified
17 view of data. *ACM Transactions on Database Systems*, 1976. URL:
18 <https://doi.org/10.1145/320434.320440> (cited on page 23).
- 19 [CMR13] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying
20 quantitative reliability for programs that execute on unreliable hard-
21 ware. In *Proceedings of the ACM SIGPLAN International Conference*
22 *on Object Oriented Programming Systems Languages & Applica-*
23 *tions, OOPSLA*, 2013. URL: [https://doi.org/10.1145/2509136.](https://doi.org/10.1145/2509136.2509546)
24 2509546 (cited on pages 68–70, 72).
- 25 [CMR15] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Strong tem-
26 poral planning with uncontrollable durations: a state-space approach.
27 In *Proceedings of the AAAI Conference on Artificial Intelligence*,
28 2015. URL: [http://www.aaai.org/ocs/index.php/AAAI/AAAI15/](http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9728)
29 [paper/view/9728](http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9728) (cited on pages 59, 62).
- 30 [CNR13] Arun Tejasvi Chaganty, Aditya V. Nori, and Sriram K. Rajamani. Effi-
1 ciently sampling probabilistic programs via program analysis. In *Pro-*
2 *ceedings of the Sixteenth International Conference on Artificial Intel-*
3 *ligence and Statistics, AISTATS*, 2013. URL: [http://proceedings.](http://proceedings.mlr.press/v31/chaganty13a.html)
4 [mlr.press/v31/chaganty13a.html](http://proceedings.mlr.press/v31/chaganty13a.html) (cited on pages 67, 68, 70, 73,
5 74, 76, 77).

- [Com⁺06] Autonomic Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 31, 2006. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.1011&rep=rep1&type=pdf> (cited on pages 4, 16–18, 20, xiv).
- [CPY⁺14] Bihuan Chen, Xin Peng, Yijun Yu, Bashar Nuseibeh, and Wenyun Zhao. Self-adaptation through incremental generative model transformations at runtime. In *36th International Conference on Software Engineering, ICSE*, 2014. URL: <https://doi.org/10.1145/2568225.2568310> (cited on pages 55, 57, 59, 61, 63, 64, 76).
- [CvL17] Antoine Cailliau and Axel van Lamsweerde. Runtime monitoring and resolution of probabilistic obstacles to system goals. In *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2017. URL: <https://doi.org/10.1109/SEAMS.2017.5> (cited on pages 55, 57, 76).
- [DBZ14] Brahim Djoudi, Chafia Bouanaka, and Nadia Zeghib. Model checking pervasive context-aware systems. In *2014 IEEE 23rd International WETICE Conference, WETICE*, 2014. URL: <https://doi.org/10.1109/WETICE.2014.11> (cited on pages 55, 57–60, 64, 66, 76).
- [Deg16] Thomas Degueule. *Composition and Interoperability for External Domain-Specific Language Engineering*. PhD thesis, University of Rennes 1, France, 2016. URL: <https://tel.archives-ouvertes.fr/tel-01427009> (cited on page 87).
- [Dem98] M Beth L Dempster. *A self-organizing systems perspective on planning for sustainability*. PhD thesis, Citeseer, 1998. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.180.6090> (cited on page 17).
- [Dey01] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 2001. URL: <https://doi.org/10.1007/s007790170019> (cited on pages 20, i).

- 4 [dL04] Mark d’Inverno and Michael Luck. *Understanding agent systems,*
5 *Second Edition*. Springer series on agent technology. 2004. ISBN:
6 978-3-540-40700-3 (cited on page 56).
- 7 [DL06] Pierre-Charles David and Thomas Ledoux. An aspect-oriented ap-
8 proach for developing self-adaptive fractal components. In *Soft-*
9 *ware Composition, SC*, 2006. URL: [https://doi.org/10.1007/](https://doi.org/10.1007/11821946_6)
10 [11821946_6](https://doi.org/10.1007/11821946_6) (cited on pages 55, 58, 59, 61, 64–66).
- 11 [dLGM⁺10] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jes-
12 per Andersson, Marin Litoiu, Bradley R. Schmerl, Gabriel Tamura,
13 Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi,
14 Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ronald J.
15 Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl
16 M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi,
17 Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek,
18 Serge Mankovski, Raffaella Mirandola, John Mylopoulos, Oscar Nier-
19 strasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Richard D.
20 Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari,
21 Kenny Wong, and Jochen Wuttke. Software engineering for self-
22 adaptive systems: a second research roadmap. In *Software Engineer-*
23 *ing for Self-Adaptive Systems II - International Seminar, Dagstuhl*
24 *Castle*, 2010. URL: [https://doi.org/10.1007/978-3-642-35813-](https://doi.org/10.1007/978-3-642-35813-5_1)
25 [5_1](https://doi.org/10.1007/978-3-642-35813-5_1) (cited on page 21).
- 26 [DMS18] Lucio Mauro Duarte, Paulo Henrique Mendes Maia, and Ana Car-
27 olina Sanchotene Silva. Extraction of probabilistic behaviour models
28 based on contexts. In *Proceedings of the 10th International Workshop*
29 *on Modelling in Software Engineering, MiSE*, 2018. URL: <https://doi.org/10.1145/3193954.3193963> (cited on pages 55, 57).
- 1 [DSB⁺17] Gwendal Daniel, Gerson Sunyé, Amine Benelallam, Massimo Tisi,
2 Yoann Vernageau, Abel Gómez, and Jordi Cabot. NeoEMF: a multi-
3 database model persistence framework for very large models. *Science*
4 *of Computer Programming*, 2017. URL: [https://doi.org/10.1016/](https://doi.org/10.1016/j.scico.2017.08.002)
5 [j.scico.2017.08.002](https://doi.org/10.1016/j.scico.2017.08.002) (cited on page 24).

- [DSC16] Gwendal Daniel, Gerson Sunyé, and Jordi Cabot. Mogwai: a framework to handle complex queries on large models. In *IEEE International Conference on Research Challenges in Information Science, RCIS*, 2016. URL: <https://doi.org/10.1109/RCIS.2016.7549343> (cited on page 24).
- [DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 1993. URL: [https://doi.org/10.1016/0167-6423\(93\)90021-G](https://doi.org/10.1016/0167-6423(93)90021-G) (cited on page 131).
- [EB10] Moritz Eysholdt and Heiko Behrens. Xtext: implement your language faster than the quick and dirty way. In *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA*, 2010. URL: <https://doi.org/10.1145/1869542.1869625> (cited on page 30).
- [Egy01] Alexander Egyed. A scenario-driven approach to traceability. In *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001*, 2001. URL: <https://doi.org/10.1109/ICSE.2001.919087> (cited on page 132).
- [EM10] Naeem Esfahani and Sam Malek. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle*, 2010. URL: https://doi.org/10.1007/978-3-642-35813-5_9 (cited on pages 68, 69).
- [EPR14] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Uncertainty in bidirectional transformations. In *6th International Workshop on Modeling in Software Engineering, MiSE 2014*, 2014. URL: <https://doi.org/10.1145/2593770.2593772> (cited on pages 67, 68, 70, 71).
- [EPR15] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Managing uncertainty in bidirectional model transformations. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software*

- 9 *Language Engineering, SLE*, 2015. URL: [https://dl.acm.org/](https://dl.acm.org/citation.cfm?id=2814259)
10 [citation.cfm?id=2814259](https://dl.acm.org/citation.cfm?id=2814259) (cited on pages 67, 68, 70, 71).
- 11 [FA04] Paolo Falcarin and Gustavo Alonso. Software architecture evolution
12 through dynamic AOP. In *Software Architecture, First European*
13 *Workshop, EWSA*, 2004. DOI: 10.1007/978-3-540-24769-2_5
14 (cited on pages 64, 65).
- 15 [Far10] Hassan Farhangi. The path of the smart grid. *IEEE power and*
16 *energy magazine*, 2010. URL: [https://doi.org/10.1109/MPE.2009.](https://doi.org/10.1109/MPE.2009.934876)
17 [934876](https://doi.org/10.1109/MPE.2009.934876) (cited on pages 4, xiv).
- 18 [FC19] Michalis Famelis and Marsha Chechik. Managing design-time uncer-
19 tainty. *Software and Systems Modeling*, 2019. URL: [https://doi.](https://doi.org/10.1007/s10270-017-0594-9)
20 [org/10.1007/s10270-017-0594-9](https://doi.org/10.1007/s10270-017-0594-9) (cited on pages 67, 68, 70, 71).
- 21 [FGL⁺10] Jean-Marie Favre, Dragan Gasevic, Ralf Lämmel, and Ekaterina
22 Pek. Empirical language analysis in software linguistics. In *Software*
23 *Language Engineering, SLE*, 2010. URL: [https://doi.org/10.](https://doi.org/10.1007/978-3-642-19440-5%5C_21)
24 [1007/978-3-642-19440-5%5C_21](https://doi.org/10.1007/978-3-642-19440-5%5C_21) (cited on page 28).
- 25 [FGL⁺11] Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio.
26 Self-adaptive software meets control theory: a preliminary approach
27 supporting reliability requirements. In *26th IEEE/ACM International*
28 *Conference on Automated Software Engineering (ASE)*, 2011. URL:
29 <https://doi.org/10.1109/ASE.2011.6100064> (cited on pages 55,
30 57, 59, 60, 64, 66, 76).
- 31 [FHM14] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated
1 design of self-adaptive software with control-theoretical formal guar-
2 antees. In *36th International Conference on Software Engineering,*
3 *ICSE*, 2014. URL: <https://doi.org/10.1145/2568225.2568272>
4 (cited on pages 17, 18).
- 5 [FMF⁺12] François Fouquet, Brice Morin, Franck Fleurey, Olivier Barais, Noël
6 Plouzeau, and Jean-Marc Jézéquel. A dynamic component model for
7 cyber physical systems. In *Proceedings of the 15th ACM SIGSOFT*
8 *Symposium on Component Based Software Engineering, CBSE*, 2012.

URL: <https://doi.org/10.1145/2304736.2304759> (cited on pages 4, 55, 58, 63–65, xiv).

[FMX⁺12] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart grid - the new and improved power grid: a survey. *IEEE Communications Surveys and Tutorials*, 2012. URL: <https://doi.org/10.1109/SURV.2011.101911.00087> (cited on pages 4, xiv).

[FNM⁺12] François Fouquet, Grégory Nain, Brice Morin, Erwan Daubert, Olivier Barais, Noël Plouzeau, and Jean-Marc Jézéquel. An eclipse modelling framework alternative to meet the models@runtime requirements. In *Model Driven Engineering Languages and Systems, MODELS*, 2012. URL: https://doi.org/10.1007/978-3-642-33666-9_7 (cited on page 27).

[FNM⁺14] François Fouquet, Grégory Nain, Brice Morin, Erwan Daubert, Olivier Barais, Noël Plouzeau, and Jean-Marc Jézéquel. Kevoree modeling framework (KMF): efficient modeling techniques for runtime use. *CoRR*, 2014. arXiv: 1405.6817. URL: <http://arxiv.org/abs/1405.6817> (cited on page 27).

[Fou10] Eclipse Foundation. Ecore. <https://wiki.eclipse.org/Ecore>, 2010. [Accessed July 2019] (cited on page 27).

[Fow10] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010. ISBN: 978-0321712943. URL: <https://www.martinfowler.com/books/dsl.html> (cited on page 42).

[FR07] Robert B. France and Bernhard Rumpe. Model-driven development of complex software: a research roadmap. In *International Conference on Software Engineering, ISCE*, 2007. URL: <https://doi.org/10.1109/F0SE.2007.14> (cited on pages 23, 26).

[FSC12] Michalis Famelis, Rick Salay, and Marsha Chechik. Partial models: towards modeling and reasoning with uncertainty. In *34th International Conference on Software Engineering, ICSE*, 2012. URL: <https://doi.org/10.1109/ICSE.2012.6227159> (cited on pages 67, 68, 70, 71).

- [fSta16] International Organization for Standardization (ISO). Structured query language (SQL). <https://www.iso.org/standard/63555.html>, 2016. [Accessed July 2019] (cited on page 24).
- [Gar08] Vahid Garousi. Traffic-aware stress testing of distributed real-time systems based on UML models in the presence of time uncertainty. In *First International Conference on Software Testing, Verification, and Validation, ICST*, 2008. URL: <https://doi.org/10.1109/ICST.2008.7> (cited on pages 68, 70, 72).
- [GB] Phil Greenwood and Lynne Blair. A framework for policy driven auto-adaptive systems using dynamic framed aspects. URL: https://doi.org/10.1007/11922827_2 (cited on pages 64, 65).
- [GBM⁺13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (IoT): a vision, architectural elements, and future directions. *Future Generation Computer Systems*, 2013. URL: <https://doi.org/10.1016/j.future.2013.01.010> (cited on page 27).
- [GBM⁺18] Alejandro Sánchez Guinea, Andrey Boytsov, Ludovic Mouline, and Yves Le Traon. Continuous identification in smart environments using wrist-worn inertial sensors. In *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous*, 2018. URL: <https://doi.org/10.1145/3286978.3287001> (cited on page viii).
- [GCH⁺04] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley R. Schmerl, and Peter Steenkiste. Rainbow: architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 2004. URL: <https://doi.org/10.1109/MC.2004.175> (cited on pages 17, 18, 55, 58–60, 63, 64).
- [GHN⁺14] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE*, 2014. URL: <https://doi.org/10.1145/2592064.2592065> (cited on page 10).

[//doi.org/10.1145/2593882.2593900](https://doi.org/10.1145/2593882.2593900) (cited on pages 7, 70, 71, xvii).

[GHP⁺08] Paul Grace, Danny Hughes, Barry Porter, Gordon S. Blair, Geoff Coulson, and François Taïani. Experiences with open overlays: a middleware approach to network heterogeneity. In *Proceedings of the 2008 EuroSys Conference*, 2008. URL: <https://doi.org/10.1145/1352592.1352606> (cited on pages 59, 60, 63, 64).

[Gli07] Martin Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference, RE*, 2007. URL: <https://doi.org/10.1109/RE.2007.45> (cited on page 22).

[GPS⁺13] Carlo Ghezzi, Leandro Sales Pinto, Paola Spoletini, and Giordano Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *35th International Conference on Software Engineering, ICSE*, 2013. URL: <https://doi.org/10.1109/ICSE.2013.6606549> (cited on pages 55, 57, 59, 60, 64, 66, 76).

[Gro14] Object Management Group. Object constraint language (OCL), version 2.4. <http://www.omg.org/spec/OCL/>, 2014. [Accessed July 2019] (cited on page 26).

[Gro15] Object Management Group. XML metadata interchange (XMI), version 2.5.1. <http://www.omg.org/spec/XMI/>, 2015. [Accessed July 2019] (cited on page 26).

[Gro16a] Object Management Group. Meta object facility (MOF), version 2.5.1. <http://www.omg.org/spec/MOF/>, 2016. [Accessed July 2019] (cited on pages 26, 27).

[Gro16b] Object Management Group. Query/view/transformation (OCL), version 1.3. <https://www.omg.org/spec/QVT>, 2016. [Accessed July 2019] (cited on page 61).

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 1995. URL: <https://doi.org/10.1006/ijhc.1995.1081> (cited on page 23).

- 11 [GS10] Carlo Ghezzi and Amir Molzam Sharifloo. Dealing with non-functional
12 requirements for adaptive systems via dynamic software product-lines.
13 In *Software Engineering for Self-Adaptive Systems II - International*
14 *Seminar, Dagstuhl Castle*, 2010. URL: [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-642-35813-5_8)
15 [978-3-642-35813-5_8](https://doi.org/10.1007/978-3-642-35813-5_8) (cited on pages 55, 57, 59, 62, 64, 66).
- 16 [GS64] Israel M. Gelfand and Georgi E. Shilov. *Generalized Functions, Vol-*
17 *ume 1, Properties and Operations*. American Mathematical Society,
18 1964. ISBN: 978-1-4704-2658-3 (cited on page 95).
- 19 [GVD19] Simos Gerasimou, Thomas Vogel, and Ada Diaconescu. Software
20 engineering for intelligent and autonomous systems: report from the
21 GI dagstuhl seminar 18343. *CoRR*, 2019. URL: [http://arxiv.org/](http://arxiv.org/abs/1904.01518)
22 [abs/1904.01518](http://arxiv.org/abs/1904.01518) (cited on pages 4, xiv).
- 23 [GvdHT09] John C. Georgas, André van der Hoek, and Richard N. Taylor. Using
24 architectural models to manage and visualize runtime adaptation.
25 *IEEE Computer*, 2009. URL: [https://doi.org/10.1109/MC.2009.](https://doi.org/10.1109/MC.2009.335)
26 [335](https://doi.org/10.1109/MC.2009.335) (cited on pages 55, 58, 59, 62–64).
- 27 [Hal06] B. D. Hall. Component interfaces that support measurement un-
28 certainty. *Computer Standards & Interfaces*, 2006. URL: <https://doi.org/10.1016/j.csi.2005.07.009> (cited on pages 67,
29 68, 70, 72, 76).
- 1 [Har16] Thomas Hartmann. *Enabling Model-Driven Live Analytics For Cyber-*
2 *Physical Systems: The Case of Smart Grids*. PhD thesis, University of
3 Luxembourg, 2016. URL: [http://orbilu.uni.lu/handle/10993/](http://orbilu.uni.lu/handle/10993/28924)
4 [28924](http://orbilu.uni.lu/handle/10993/28924) (cited on pages 10, 27, 139, xxii).
- 5 [HBB15] Sara Hassan, Nelly Bencomo, and Rami Bahsoon. Minimizing nasty
6 surprises with better informed decision-making in self-adaptive sys-
7 tems. In *10th IEEE/ACM International Symposium on Software*
8 *Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2015.
9 URL: <https://doi.org/10.1109/SEAMS.2015.13> (cited on pages 8,
10 11, xix, xxii).

- [HFJ⁺17] Thomas Hartmann, François Fouquet, Matthieu Jimenez, Romain Rouvoy, and Yves Le Traon. Analyzing complex data in motion at scale with temporal graphs. In *The 29th International Conference on Software Engineering and Knowledge Engineering*, 2017. URL: <https://doi.org/10.18293/SEKE2017-048> (cited on page 139).
- [HFK⁺14a] Thomas Hartmann, François Fouquet, Jacques Klein, Grégory Nain, and Yves Le Traon. Reactive security for smart grids using models@run. time-based simulation and reasoning. In *Smart Grid Security - Second International Workshop, SmartGridSec*, 2014. URL: https://doi.org/10.1007/978-3-319-10329-7_9 (cited on pages 4, 5, 37, 48, 55–60, 64, 65, 76, xiv, xv).
- [HFK⁺14b] Thomas Hartmann, François Fouquet, Jacques Klein, Yves Le Traon, Alexander Pelov, Laurent Toutain, and Tanguy Ropitault. Generating realistic smart grid communication topologies based on real-data. In *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm*, 2014. URL: <https://doi.org/10.1109/SmartGridComm.2014.7007684> (cited on pages 4, 36, 142, xiv).
- [HFM⁺16] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 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. URL: <http://dl.acm.org/citation.cfm?id=2976812> (cited on page 24).
- [HFM⁺19] Thomas Hartmann, François Fouquet, Assaad Moawad, Romain Rouvoy, and Yves Le Traon. GreyCat: efficient what-if analytics for data in motion at scale. *Information Systems*, 2019. URL: <https://doi.org/10.1016/j.is.2019.03.004> (cited on pages 27, 58, 83).
- [HFN⁺14a] Thomas Hartmann, François Fouquet, Grégory Nain, Brice Morin, Jacques Klein, Olivier Barais, and Yves Le Traon. A native versioning concept to support historized models at runtime. In *Model-Driven Engineering Languages and Systems, MODELS*, 2014. URL: https://doi.org/10.1007/978-3-319-10329-7_9 (cited on pages 4, 5, 37, 48, 55–60, 64, 65, 76, xiv, xv).

- 13 [//doi.org/10.1007/978-3-319-11653-2_16](https://doi.org/10.1007/978-3-319-11653-2_16) (cited on pages 55,
14 56, 58, 63, 64, 76).
- 15 [HFN⁺14b] Thomas Hartmann, François Fouquet, Grégory Nain, Brice Morin,
16 Jacques Klein, and Yves Le Traon. Reasoning at runtime using time-
17 distorted contexts: a models@run.time based approach. In *The 26th*
18 *International Conference on Software Engineering and Knowledge*
19 *Engineering*, 2014. URL: [http://ksiresearchorg.ipage.com/](http://ksiresearchorg.ipage.com/seke/Proceedings/seke/SEKE2014_Proceedings.pdf)
20 [seke/Proceedings/seke/SEKE2014_Proceedings.pdf](http://ksiresearchorg.ipage.com/seke/Proceedings/seke/SEKE2014_Proceedings.pdf) (cited on
21 pages 10, 20, 21, 27, 55, 56, 58, 63, 64, 76, xxi).
- 22 [HHP⁺08] Svein O. Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus
23 Schmid. Dynamic software product lines. *IEEE Computer*, 2008.
24 URL: <https://doi.org/10.1109/MC.2008.123> (cited on page 19).
- 25 [HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy.
26 Modeling context information in pervasive computing systems. In
27 *Pervasive Computing, First International Conference, Pervasive*,
28 2002. URL: https://doi.org/10.1007/3-540-45866-2_14 (cited
29 on pages 20, 21, 55, 56, 58, 76).
- 30 [HMF⁺16] Thomas Hartmann, Assaad Moawad, François Fouquet, Yves Reckinger,
1 Jacques Klein, and Yves Le Traon. Near real-time electric load approx-
2 imation in low voltage cables of smart grids with models@run.time.
3 In *Proceedings of the 31st Annual ACM Symposium on Applied Com-*
4 *puting*, 2016. URL: <https://doi.org/10.1145/2851613.2853125>
5 (cited on pages 36, 37).
- 6 [HMF⁺19] Thomas Hartmann, Assaad Moawad, François Fouquet, and Yves Le
7 Traon. The next evolution of MDE: a seamless integration of machine
8 learning into domain modeling. *Software and System Modeling*, 2019.
9 URL: <https://doi.org/10.1007/s10270-017-0600-2> (cited on
10 page 27).
- 11 [Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communica-*
12 *tions of the ACM*, 1978. URL: [https://doi.org/10.1145/359576.](https://doi.org/10.1145/359576.359585)
13 [359585](https://doi.org/10.1145/359576.359585) (cited on page 56).

- [HR04] David Harel and Bernhard Rumpe. Meaningful modeling: what's the semantics of "semantics"? *IEEE Computer*, 2004. URL: <https://doi.org/10.1109/MC.2004.172> (cited on pages 29, 87).
- [HRW11] John Edward Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE*, 2011. URL: <https://doi.org/10.1145/1985793.1985882> (cited on pages 23, 24).
- [HWR⁺11] John Edward Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE*, 2011. URL: <https://doi.org/10.1145/1985793.1985858> (cited on page 23).
- [IA09] Ali Ipakchi and Farrokh Albuyeh. Grid of the future. *IEEE power and energy magazine*, 2009. URL: <https://doi.org/10.1109/MPE.2008.931384> (cited on pages 4, xiv).
- [III17] ISO, IEC, and IEEE. Systems and software engineering – vocabulary. In *ISO/IEC/IEEE 24765: 2017 (E)*. 2017. URL: <https://doi.org/10.1109/IEEESTD.2017.8016712> (cited on pages 21, 22, i, ii).
- [IW14] M. Usman Iftikhar and Danny Weyns. ActivFORMS: active formal models for self-adaptation. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2014. URL: <https://doi.org/10.1145/2593929.2593944> (cited on pages 55, 57, 59, 60, 64, 65, 76).
- [Jac97] Michael Jackson. The meaning of requirements. *Annals of Software Engineering*, 1997. URL: <https://doi.org/10.1023/A:10189900005598> (cited on page 17).
- [JCB⁺15] Jean-Marc Jézéquel, Benoît Combemale, Olivier Barais, Martin Monperrus, and François Fouquet. Mashup of metalanguages and its implementation in the kermeta language workbench. *Software and*

- 14 *Systems Modeling*, 2015. URL: [https://doi.org/10.1007/s10270-](https://doi.org/10.1007/s10270-013-0354-4)
15 013-0354-4 (cited on page 106).
- 16 [JG17] Seyyed Ahmad Javadi and Anshul Gandhi. DIAL: reducing tail
17 latencies for cloud applications via dynamic interference-aware load
18 balancing. In *2017 IEEE International Conference on Autonomic*
19 *Computing, ICAC*, 2017. URL: [https://doi.org/10.1109/ICAC.](https://doi.org/10.1109/ICAC.2017.17)
20 2017.17 (cited on pages 4, xiv).
- 21 [JK12] Szymon Jaroszewicz and Marcin Korzen. Arithmetic operations
22 on independent random variables: a numerical approach. *SIAM J.*
23 *Scientific Computing*, 2012. URL: [https://doi.org/10.1137/](https://doi.org/10.1137/110839680)
24 110839680 (cited on pages 67, 68, 70, 73, 74, 76, 77).
- 25 [JWB⁺15] Andrés Jiménez-Ramírez, Barbara Weber, Irene Barba, and Carmelo
26 Del Valle. Generating optimized configurable business process models
27 in scenarios subject to uncertainty. *Information & Software Tech-*
28 *nology*, 2015. URL: [https://doi.org/10.1016/j.infsof.2014.06.](https://doi.org/10.1016/j.infsof.2014.06.006)
29 006 (cited on pages 67, 68, 70, 72, 76).
- 30 [KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic
31 computing. *IEEE Computer*, 2003. URL: [https://doi.org/10.](https://doi.org/10.1109/MC.2003.1160055)
1 1109/MC.2003.1160055 (cited on pages 4, 16–18, 20, ii, iii, xiv).
- 2 [Kel76] Robert M. Keller. Formal verification of parallel programs. *Commu-*
3 *nications of ACM*, 1976. URL: [https://doi.org/10.1145/360248.](https://doi.org/10.1145/360248.360251)
4 360251 (cited on page 57).
- 5 [Ken02] Stuart Kent. Model driven engineering. In *Integrated Formal Methods,*
6 *Third International Conference, IFM*, 2002. URL: [https://doi.org/](https://doi.org/10.1007/3-540-47884-1_16)
7 10.1007/3-540-47884-1_16 (cited on pages 4, 23, 24, xv).
- 8 [KH10] Maximilian Koegel and Jonas Helming. EMFStore: a model repos-
9 itory for EMF models. In *Proceedings of the 32nd ACM/IEEE In-*
10 *ternational Conference on Software Engineering, ICSE*, 2010. URL:
11 <https://doi.org/10.1145/1810295.1810364> (cited on pages 10,
12 xxi).

- [Kle08] Anneke Kleppe. *Software language engineering: creating domain-specific languages using metamodels*. 2008. ISBN: 0-321-60646-9. URL: <https://www.pearson.com/us/higher-education/program/Kleppe-Software-Language-Engineering-Creating-Domain-Specific-Languages-Using-Metamodels/PGM162096.html> (cited on pages 28, iii).
- [KM90] Jeff Kramer and Jeff Magee. The evolving philosophers problem: dynamic change management. *IEEE Transactions on Software Engineering*, 1990. URL: <https://doi.org/10.1109/32.60317> (cited on pages 17, 55, 58, 59, 61, 64, 65).
- [Kra07] Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 2007. URL: <https://doi.org/10.1145/1232743.1232745> (cited on page 23).
- [KT12] Bilal Kanso and Safouan Taha. Temporal constraint support for OCL. In *Software Language Engineering, SLE*, 2012. URL: https://doi.org/10.1007/978-3-642-36089-3_6 (cited on pages 10, xxi).
- [LEB18] Eric O. LEBIGOT. Uncertainties: a python package for calculations with uncertainties. <http://pythonhosted.org/uncertainties/>, 2018. Accessed: 2018-10-11 (cited on page 41).
- [Lew03] Bil Lewis. Debugging backwards in time. *CoRR*, cs.SE/0310016, 2003. URL: <http://arxiv.org/abs/cs.SE/0310016> (cited on page 25).
- [LGC17] Philippe Lalanda, Eva Gerbert-Gaillard, and Stéphanie Chollet. Self-aware context in smart home pervasive platforms. In *2017 IEEE International Conference on Autonomic Computing, ICAC*, 2017. URL: <https://doi.org/10.1109/ICAC.2017.1> (cited on pages 4, xiv).
- [LTB⁺00] David J. Lunn, Andrew Thomas, Nicky Best, and David J. Spiegelhalter. WinBUGS - a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and Computing*, 2000. URL: <https://>

- 12 //doi.org/10.1023/A:1008929526011 (cited on pages 67, 68, 70,
13 73, 74, 76, 77).
- 14 [Mao09] Shahar Maoz. Using model-based traces as runtime models. *IEEE*
15 *Computer*, 2009. URL: <https://doi.org/10.1109/MC.2009.336>
16 (cited on pages 55, 58).
- 17 [MAR14] Danilo F. Mendonça, Raian Ali, and Genáina Nunes Rodrigues.
18 Modelling and analysing contextual failures for dependability re-
19 quirements. In *9th International Symposium on Software Engineer-*
20 *ing for Adaptive and Self-Managing Systems, SEAMS*, 2014. URL:
21 <https://doi.org/10.1145/2593929.2593947> (cited on pages 55,
22 57, 59, 61, 76).
- 23 [MBE⁺11] Michael Maurer, Ivan Breskovic, Vincent C. Emeakaroha, and Ivona
24 Brandic. Revealing the MAPE loop for the autonomic management
25 of cloud infrastructures. In *Proceedings of the 16th IEEE Symposium*
26 *on Computers and Communications, ISCC*, 2011. URL: <https://doi.org/10.1109/ISCC.2011.5984008> (cited on pages 64, 66).
- 27 [MBF⁺18] Ludovic Mouline, Amine Benelallam, François Fouquet, Johann Bour-
28 cier, and Olivier Barais. A temporal model for interactive diagnosis
29 of adaptive systems. In *2018 IEEE International Conference on*
30 *Autonomic Computing, ICAC*, 2018. URL: <https://doi.org/10.1109/ICAC.2018.00029> (cited on pages 13, vii, xxv).
- 1 [MBH⁺18] Ludovic Mouline, Amine Benelallam, Thomas Hartmann, François
2 Fouquet, Johann Bourcier, Brice Morin, and Olivier Barais. Enabling
3 temporal-aware contexts for adaptative distributed systems. In *Pro-*
4 *ceedings of the 33rd Annual ACM Symposium on Applied Computing,*
5 *SAC*, 2018. URL: <https://doi.org/10.1145/3167132.3167286>
6 (cited on pages 13, vii, xxv).
- 7 [MBH⁺ng] Ludovic Mouline, Amine Benelallam, Thomas Hartmann, Johann
8 Bourcier, Olivier Barais, and Maxime Cordy. Ain'tea: managing data
9 uncertainty at the language level. *Forthcoming*, forthcoming (cited
10 on pages 12, vii, xxiv).

- [MBJ⁺09] Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey, and Arnor Solberg. Models@run.time to support dynamic adaptation. *IEEE Computer*, 2009. URL: <https://doi.org/10.1109/MC.2009.327> (cited on pages 4, 5, 19, 47, 54, 55, 63, 64, xv).
- [MBN⁺09] Brice Morin, Olivier Barais, Grégory Nain, and Jean-Marc Jézéquel. Taming dynamically adaptive systems using models and aspects. In *31st International Conference on Software Engineering, ICSE*, 2009. URL: <https://doi.org/10.1109/ICSE.2009.5070514> (cited on pages 64, 66).
- [MCG⁺15] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley R. Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE*, 2015. URL: <https://doi.org/10.1145/2786805.2786853> (cited on pages 55, 57, 59, 60, 64, 66, 76).
- [MD04] Erik Meijer and Peter Drayton. Static typing where possible, dynamic typing when needed: the end of the cold war between programming languages, 2004 (cited on page 71).
- [Met08] JCGi Metrology. Evaluation of measurement data - guide to the expression of uncertainty in measurement. *Bureau International des Poids et Mesures*, 2008. URL: <https://www.bipm.org/en/publications/guides/gum.html> (cited on pages 5, 6, 11, 33, 70, 87, 153, xvi, xxii).
- [MGB63] Alexander M. Mood, Franklin A. Graybill, and Duane C. Boes. *Introduction to the Theory of Statistics*. McGraw Hill; 3rd edition, 1963. ISBN: 978-0070854659 (cited on page 96).
- [MHF⁺15] Assaad Moawad, Thomas Hartmann, François Fouquet, Grégory Nain, Jacques Klein, and Yves Le Traon. Beyond discrete modeling: a continuous and efficient model for iot. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and*

- 13 *Systems, MoDELS*, 2015. URL: [https://doi.org/10.1109/MODELS.](https://doi.org/10.1109/MODELS.2015.7338239)
14 2015.7338239 (cited on page 27).
- 15 [MHF⁺17] Ludovic Mouline, Thomas Hartmann, François Fouquet, Yves Le
16 Traon, Johann Bourcier, and Olivier Barais. Weaving rules into
17 models@run.time for embedded smart systems. In *Companion to the*
18 *first International Conference on the Art, Science and Engineering*
19 *of Programming, Programming*, 2017. URL: [https://doi.org/10.](https://doi.org/10.1145/3079368.3079394)
20 1145/3079368.3079394 (cited on page viii).
- 21 [Mos01] Peter D. Mosses. The varieties of programming language semantics.
22 In *Perspectives of System Informatics, PSI*, 2001. URL: [https://](https://doi.org/10.1007/3-540-45575-2_18)
23 doi.org/10.1007/3-540-45575-2_18 (cited on page 87).
- 24 [MPS15] Marina Mongiello, Patrizio Pelliccione, and Massimo Sciancalepore.
25 AC-Contract: run-time verification of context-aware applications. In
26 *10th IEEE/ACM International Symposium on Software Engineering*
27 *for Adaptive and Self-Managing Systems, SEAMS*, 2015. URL: [https://](https://doi.org/10.1109/SEAMS.2015.11)
28 doi.org/10.1109/SEAMS.2015.11 (cited on pages 8, 11, xix,
29 xxii).
- 30 [MS17] Vera Zaychik Moffitt and Julia Stoyanovich. Temporal graph algebra.
1 In *Proceedings of The 16th International Symposium on Database*
2 *Programming Languages, DBPL*, 2017. URL: [https://doi.org/10.](https://doi.org/10.1145/3122831.3122838)
3 1145/3122831.3122838 (cited on pages 55, 58, 64, 65, 76).
- 4 [MSS13] Prodromos Makris, Dimitrios N. Skoutas, and Charalabos Skianis. A
5 survey on context-aware mobile and wireless networking: on network-
6 ing and computing environments’ integration. *IEEE Communications*
7 *Surveys and Tutorials*, 2013. URL: [https://doi.org/10.1109/SURV.](https://doi.org/10.1109/SURV.2012.040912.00180)
8 2012.040912.00180 (cited on pages 20, 21).
- 9 [MWG⁺18] T. Minka, J.M. Winn, J.P. Guiver, Y. Zaykov, D. Fabian, and J.
10 Bronskill. Infer.net 0.3, 2018. URL: [https://dotnet.github.io/](https://dotnet.github.io/infer/)
11 [infer/](https://dotnet.github.io/infer/). [Accessed July 2019] (cited on pages 7, 41, 106, xvii).

- [MWV16] Tanja Mayerhofer, Manuel Wimmer, and Antonio Vallecillo. Adding uncertainty and units to quantity types in software models. In *International Conference on Software Language Engineering*, 2016. URL: <http://dl.acm.org/citation.cfm?id=2997376> (cited on pages 67–70, 73, 76, 77, 115).
- [OMG17] Object Management Group (OMG). OMG Unified Modeling Language, version 2.5.1, 2017. URL: <https://www.omg.org/spec/UML/2.5.1>. [Accessed July 2019] (cited on pages 23, 25, ii).
- [oSN] National Institute of Standards and Technology (NIST). Smart grid: a beginner’s guide. <https://www.nist.gov/engineering-laboratory/smart-grid/smart-grid-beginners-guide#what>. [Accessed June 2019] (cited on page iii).
- [PBC⁺11] Carlos Parra, Xavier Blanc, Anthony Cleve, and Laurence Duchien. Unifying design and runtime software adaptation using aspect models. *Science of Computer Programming*, 2011. URL: <https://doi.org/10.1016/j.scico.2010.12.005> (cited on pages 59, 61, 64, 65).
- [Pfe01] Avi Pfeffer. IBAL: a probabilistic rational programming language. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI*, 2001 (cited on pages 67, 68, 70, 73, 74, 76, 77).
- [PFT03] Mónica Pinto, Lidia Fuentes, and José M. Troya. DAOP-ADL: an architecture description language for dynamic component and aspect-based development. In Frank Pfenning and Yannis Smaragdakis, editors, *Generative Programming and Component Engineering, Second International Conference, GPCE*, 2003. URL: https://doi.org/10.1007/978-3-540-39815-8_8 (cited on pages 59, 60, 64, 65).
- [Plu⁺03] Martyn Plummer et al. Jags: a program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, 2003 (cited on pages 67, 68, 70, 73, 74, 76, 77).

- 11 [PPT08] Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic
12 language based on sampling functions. *ACM Transactions on*
13 *Programming Languages and Systems*, 2008. URL: <https://doi.org/10.1145/1452044.1452048> (cited on pages 67, 68, 70, 73, 74,
14 76, 77).
15
- 16 [PSR10] Dhirendra Pandey, Ugrasen Suman, and AK Ramani. An effective
17 requirement engineering process model for software development
18 and requirements management. In *Advances in Recent Technologies*
19 *in Communication and Computing (ARTCom)*, 2010, 2010. URL:
20 <https://doi.org/10.1109/ARTCom.2010.24> (cited on page 22).
- 21 [PZC⁺14] Charith Perera, Arkady B. Zaslavsky, Peter Christen, and Dimitrios
22 Georgakopoulos. Context aware computing for the internet of things:
23 a survey. *IEEE Communications Surveys and Tutorials*, 2014. URL:
24 <https://doi.org/10.1109/SURV.2013.042313.00197> (cited on
25 page 20).
- 26 [RP02] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and
27 monads of probability distributions. In *Conference Record of POPL*
28 *2002: The 29th SIGPLAN-SIGACT Symposium on Principles of*
29 *Programming Languages*, 2002. URL: [https://doi.org/10.1145/](https://doi.org/10.1145/503272.503288)
1 [503272.503288](https://doi.org/10.1145/503272.503288) (cited on pages 67, 68, 70, 73, 74, 76, 77).
- 2 [RRV08] José Eduardo Rivera, José Raúl Romero, and Antonio Vallecillo.
3 Behavior, time and viewpoint consistency: three challenges for MDE.
4 In *Models in Software Engineering, Workshops and Symposia at*
5 *MODELS*, 2008. URL: [https://doi.org/10.1007/978-3-642-](https://doi.org/10.1007/978-3-642-01648-6_7)
6 [01648-6_7](https://doi.org/10.1007/978-3-642-01648-6_7) (cited on pages 10, xxi).
- 7 [SBM⁺08] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro.
8 *EMF: eclipse modeling framework, 2nd Edition*. Addison-Wesley
9 Professional, 2008. ISBN: 0-321-33188-5. URL: [http://www.informit.](http://www.informit.com/store/emf-eclipse-modeling-framework-9780321331885)
10 [com/store/emf-eclipse-modeling-framework-9780321331885](http://www.informit.com/store/emf-eclipse-modeling-framework-9780321331885)
11 (cited on page 27).

- [SCG13] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI*, 2013. URL: <https://doi.org/10.1145/2491956.2462179> (cited on pages 67, 68, 70, 73, 74, 76, 77).
- [SCH⁺13] Rick Salay, Marsha Chechik, Jennifer Horkoff, and Alessio Di Sandro. Managing requirements uncertainty with partial models. *Requirements Engineering*, 2013. URL: <https://doi.org/10.1007/s00766-013-0170-y> (cited on pages 67, 68, 70, 71).
- [Sch06] Douglas C. Schmidt. Guest editor’s introduction: model-driven engineering. *IEEE Computer*, 2006. URL: <https://doi.org/10.1109/MC.2006.58> (cited on pages 4, 23–25, ii, xv).
- [Sha76] Glenn Shafer. *A mathematical theory of evidence*, volume 42. Princeton university press, 1976. ISBN: 9780691100425 (cited on pages 6, 11, 153, xvii, xxii).
- [SMH11] Julia Schwarz, Jennifer Mankoff, and Scott E. Hudson. Monte carlo methods for managing interactive state, action and feedback under uncertainty. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 2011. URL: <https://doi.org/10.1145/2047196.2047227> (cited on pages 67–70, 73, 74, 76, 77).
- [ST09] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: landscape and research challenges. *TAAS*, 2009. URL: <https://doi.org/10.1145/1516533.1516538> (cited on pages 17, 18).
- [SWF16] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using PyMC3. *PeerJ Computer Science*, 2016. URL: <https://doi.org/10.7717/peerj-cs.55> (cited on pages 41, 67, 68, 70, 73, 74, 76, 77).

- [Tea15] Watcher Drivers Team. Openstack watcher. <https://wiki.openstack.org/wiki/Watcher>, 2015. [Accessed April 2019] (cited on pages 4, xiv).
- [TGE⁺10] Hossein Tajalli, Joshua Garcia, George Edwards, and Nenad Medvidovic. PLASMA: a plan-based layered architecture for software model-driven adaptation. In *25th IEEE/ACM International Conference on Automated Software Engineering*, 2010. URL: <https://doi.org/10.1145/1858996.1859092> (cited on pages 55, 57, 59, 60, 64, 66, 76).
- [Thr00] Sebastian Thrun. Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA*, 2000. URL: <https://doi.org/10.1109/ROBOT.2000.844075> (cited on pages 67, 68, 70, 73, 74, 76, 77).
- [TOH17] Yasuyuki Tahara, Akihiko Ohsuga, and Shinichi Honiden. Formal verification of dynamic evolution processes of UML models using aspects. In *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2017. URL: <https://doi.org/10.1109/SEAMS.2017.4> (cited on pages 8, 11, 55–60, 63, 64, 76, xix, xxiii).
- [vDK98] Arie van Deursen and Paul Klint. Little languages: little maintenance? *Journal of Software Maintenance*, 1998. URL: [https://doi.org/10.1002/\(SICI\)1096-908X\(199803/04\)10:2%3C75::AID-SMR168%3E3.0.CO;2-5](https://doi.org/10.1002/(SICI)1096-908X(199803/04)10:2%3C75::AID-SMR168%3E3.0.CO;2-5) (cited on page 29).
- [vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Notices*, 2000. URL: <https://doi.org/10.1145/352029.352035> (cited on pages 29, i).
- [VMO16] Antonio Vallecillo, Carmen Morcillo, and Priscill Orue. Expressing measurement uncertainty in software models. In *10th International Conference on the Quality of Information and Communications Technology, QUATIC*, 2016. URL: <http://doi.ieeecomputersociety.org>.

org/10.1109/QUATIC.2016.013 (cited on pages 11, 67–70, 73–77, xxii).

[Voe14] Markus Voelter. *Generic tools, specific languages*. Citeseer, 2014. ISBN: 978-94-6203-586-7. URL: <https://www.voelter.de/data/books/GenericToolsSpecificLanguages-1.0-web.pdf> (cited on page 29).

[W3C05] World Wide Web Consortium (W3C). Document object model (DOM). <https://www.w3.org/DOM/>, 2005. [Accessed July 2019] (cited on page 29).

[WA13] Danny Weyns and Tanvir Ahmad. Claims and evidence for architecture-based self-adaptation: a systematic literature review. In *Software Architecture ECSA*, 2013. URL: https://doi.org/10.1007/978-3-642-39031-9%5C_22 (cited on page 17).

[Wal96] Christian Walck. Hand-book on statistical distributions for experimentalists. Technical report, University of Stockholm, 1996 (cited on pages 88, 95).

[WBR11] Richard Wang, Dana Butnariu, and Jennifer Rexford. Openflow-based server load balancing gone wild. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE*, 2011. URL: <https://www.usenix.org/conference/hot-ice11/openflow-based-server-load-balancing-gone-wild> (cited on page 45).

[Wey19] Danny Weyns. Software engineering of self-adaptive systems. In *Handbook of Software Engineering*. 2019. URL: https://doi.org/10.1007/978-3-030-00262-6%5C_11 (cited on pages 16, 17).

[WH04] Tom De Wolf and Tom Holvoet. Emergence versus self-organisation: different concepts but promising when combined. In *Engineering Self-Organising Systems, Methodologies and Applications*, 2004. URL: https://doi.org/10.1007/11494676_1 (cited on page 17).

- 5330 [WHH10] Danny Weyns, Robrecht Haesevoets, and Alexander Helleboogh.
5331 The MACODO organization model for context-driven dynamic agent
5332 organizations. *TAAS*, 2010. URL: [https://doi.org/10.1145/](https://doi.org/10.1145/1867713.1867717)
5333 [1867713.1867717](https://doi.org/10.1145/1867713.1867717) (cited on pages 55, 56, 59, 62, 64, 65).
- 5334 [WHR] Jon Whittle, John Edward Hutchinson, and Mark Rouncefield. The
5335 state of practice in model-driven engineering. *IEEE Software*. URL:
5336 <https://doi.org/10.1109/MS.2013.65> (cited on page 23).
- 5337 [WHR⁺13] Jon Whittle, John Edward Hutchinson, Mark Rouncefield, Håkan
5338 Burden, and Rogardt Haldal. Industrial adoption of model-driven
5339 engineering: are the tools really the problem? In *Model-Driven En-*
5340 *gineering Languages and Systems, MODELS*, 2013. URL: [https:](https://doi.org/10.1007/978-3-642-41533-3%5C_1)
5341 [//doi.org/10.1007/978-3-642-41533-3%5C_1](https://doi.org/10.1007/978-3-642-41533-3%5C_1) (cited on page 24).
- 5342 [WKE08] Gerd Wübbeler, Michael Krystek, and Clemens Elster. Evaluation
5343 of measurement uncertainty and its numerical calculation by a
5344 monte carlo method. *Measurement science and technology*, 2008.
5345 URL: <https://doi.org/10.1088/0957-0233/19/8/084009> (cited
5346 on page 153).
- 5347 [WMA12] Danny Weyns, Sam Malek, and Jesper Andersson. FORMS: unifying
5348 reference model for formal specification of distributed self-adaptive
5349 systems. *TAAS*, 2012. URL: [https://doi.org/10.1145/2168260.](https://doi.org/10.1145/2168260.2168268)
5350 [2168268](https://doi.org/10.1145/2168260.2168268) (cited on pages 17, 18, 55, 56, 64, 65).
- 5351 [Woh14] Claes Wohlin. Guidelines for snowballing in systematic literature stud-
5352 ies and a replication in software engineering. In *18th International*
5353 *Conference on Evaluation and Assessment in Software Engineering,*
5354 *EASE*, 2014. URL: <https://doi.org/10.1145/2601248.2601268>
5355 (cited on pages 53, 75).
- 5356 [WSB⁺09] Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng,
5357 and Jean-Michel Bruel. RELAX: incorporating uncertainty into the
5358 specification of self-adaptive systems. In *International Requirements*
5359 *Engineering Conference*, 2009. URL: [https://doi.org/10.1109/RE.](https://doi.org/10.1109/RE.2009.36)
5360 [2009.36](https://doi.org/10.1109/RE.2009.36) (cited on pages 68, 70, 72).

- 5361 [WSB⁺10] Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng, and
5362 Jean-Michel Bruel. RELAX: a language to address uncertainty in
5363 self-adaptive systems requirement. *Requirements Engineering*, 2010.
5364 URL: <https://doi.org/10.1007/s00766-010-0101-0> (cited on
5365 pages 68, 70, 72).
- 5366 [Yu11] Eric Yu. Modelling strategic relationships for process reengineering.
5367 *Social Modeling for Requirements Engineering*, 2011. URL: [http://ftp.cs.utoronto.ca/public_html/dist/eric/DKBS-TR-94-
5368 6.pdf](http://ftp.cs.utoronto.ca/public_html/dist/eric/DKBS-TR-94-6.pdf) (cited on page 131).
- 5370 [ZA11] Enrico Zio and Terje Aven. Uncertainties in smart grids behavior
5371 and modeling: what are the risks and vulnerabilities? how to analyze
5372 them? *Energy Policy*, 2011. URL: [https://doi.org/10.1016/j.
5373 enpol.2011.07.030](https://doi.org/10.1016/j.enpol.2011.07.030) (cited on pages 5, xv).
- 5374 [Zad] Lotfi A Zadeh. Fuzzy sets. In *Fuzzy Sets, Fuzzy Logic, And Fuzzy
5375 Systems: Selected Papers by Lotfi A Zadeh*. URL: [https://doi.org/
5376 10.1142/9789814261302_0001](https://doi.org/10.1142/9789814261302_0001) (cited on pages 11, xxii).
- 5377 [ZAY⁺19] Man Zhang, Shaukat Ali, Tao Yue, Roland Norgren, and Oscar
5378 Okariz. Uncertainty-wise cyber-physical system test modeling. *Soft-
5379 ware and Systems Modeling*, 2019. URL: [https://doi.org/10.1007/
5380 s10270-017-0609-6](https://doi.org/10.1007/s10270-017-0609-6) (cited on pages 67–70, 72, 76).
- 5381 [ZGC09] Ji Zhang, Heather Goldsby, and Betty H. C. Cheng. Modular veri-
5382 fication of dynamically adaptive systems. In *Proceedings of the 8th
5383 International Conference on Aspect-Oriented Software Development,
5384 AOSD*, 2009. URL: <https://doi.org/10.1145/1509239.1509262>
5385 (cited on pages 55, 57, 59, 60, 64, 66, 76).
- 5386 [ZSA⁺16] Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and
5387 Roland Norgren. Understanding uncertainty in cyber-physical sys-
5388 tems: a conceptual model. In *Modelling Foundations and Applica-
5389 tions, ECMFA*, 2016. URL: [https://doi.org/10.1007/978-3-319-
5390 42061-5_16](https://doi.org/10.1007/978-3-319-42061-5_16) (cited on pages 67–70, 72, 76).

Titre : Vers un *framework* de modélisation avec des données temporelles et incertaines pour les systèmes adaptatifs

Mots clés : systèmes dynamiquement adaptatifs, représentation des connaissances, ingénierie dirigée par les modèles, modélisation de l'incertitude, modélisation temporelle

Résumé : Les systèmes auto-adaptatifs optimisent leurs comportements ou configurations au moment de l'exécution en réponse à une modification de leur environnement ou de leurs comportements. Ces systèmes nécessitent donc une connaissance approfondie de la situation en cours qui permet de raisonner en considérant les opérations d'adaptation. En utilisant la méthodologie de l'Ingénierie Dirigée par les Modèles, il est possible d'abstraire cette situation. Cependant, les informations concernant le système ne sont pas toujours connues avec une confiance absolue. De plus, dans de tels systèmes, la fréquence de surveillance peut différer du délai nécessaire pour que les mesures de reconfiguration aient des effets mesurables. Ces caractéristiques s'accompagnent d'un défi global pour les ingénieurs logiciels : comment représenter les connaissances incertaines tout en permettant de les interroger efficacement et

de représenter les actions en cours afin d'améliorer les processus d'adaptation ?

Pour relever ce défi, cette thèse défend la nécessité d'un *framework* de modélisation qui inclut, en plus de tous les éléments traditionnels, l'incertitude et le temps en tant que concepts de première classe. Par conséquent, un développeur sera en mesure d'extraire des informations relatives au processus d'adaptation, à l'environnement ainsi qu'au système lui-même.

Dans cette optique, nous présentons deux contributions évaluées : un modèle de contexte temporel et un langage pour les données incertaines. Le modèle de contexte temporel permet d'abstraire les actions passées, en cours et futures avec leurs impacts et leur contexte. Le langage, appelé Ain'tea, intègre l'incertitude des données en tant que concept de première classe.

Title : Towards a modelling framework with temporal and uncertain data for adaptive systems

Keywords : dynamically adaptive systems, knowledge representation, model-driven engineering, uncertainty modelling, time modelling

Abstract : Self-Adaptive Systems optimise their behaviours or configurations at runtime in response to a modification of their environments or their behaviours. These systems therefore need a deep understanding of the ongoing situation which enables reasoning tasks for adaptation operations. Using the model-driven engineering (MDE) methodology, one can abstract this situation. However, information concerning the system is not always known with absolute confidence. Moreover, in such systems, the monitoring frequency may differ from the delay for reconfiguration actions to have measurable effects.

These characteristics come with a challenge for software engineers: how to represent uncertain knowledge that can be efficiently que-

ried and to represent ongoing actions in order to improve adaptation processes? To tackle this challenge, this thesis defends the need for a unified modelling framework which includes, besides all traditional elements, temporal and uncertainty as first-class concepts. Therefore, a developer will be able to abstract information related to the adaptation process, the environment as well as the system itself.

Towards this vision, we present two evaluated contributions: a temporal context model and a language for uncertain data. The temporal context model allows abstracting past, ongoing and future actions with their impacts and context. The language, named Ain'tea, integrates data uncertainty as a first-class citizen.