Il faut revoir le style de l'intro.
Le reste est mieux écrit au niveau de l'anglais.
L'intro est trop longue et répète 20x la même chose

1

Introduction

⟨chapt:intro⟩?

Contents

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

1.1 Context

Utilities are introducing more and more Information and Communication Technology (ICT)* in the grid to face the new challenges of electricity supply [Far10; IA09; FMX+12]. The literature and the industry refer to these nowadays power grids as smart grid*.

In this document, we focus on the **self-healing*** capacity of such grids. A self-healing system* can automatically repair any incident, software or hardware, at runtime [KC03]. For example, a smart grid can optimise the power flow to deal with failures of transformers[1] [FMX+12]. In this way, the incident will impact as few users as possibles, ideally none.

This healing mechanism can be performed only if the smart grid* has a deep understanding of itself (its struc-ture* and its behaviour*) and its environment* (where it is executed). This understanding can be extracted from an, or a set of, **abstraction**(s) of these elements. Abstractions provide an illuminating description of systems, their behaviours*, or their environments*. For example, Hartmann *et al.*, [HFK+14b] provide a class diagram that describes the smart grid topology, when it uses power lines communications[2].

More generally, a self-healing system* is a **self-adaptive system***. Self-adaptive systems* optimize their behaviours* or configurations at runtime in response to a modification of their environments* or their behaviours* [CdLG+09]. Kephart and Chess [KC03] laid the groundwork for this approach, based on an IBM white paper [Com+06]. Since then, practitioners have applied it to different domains [GVD19] such as cloud infrastructure [JG17; Tea15; BKF+17] or Cyber-Physical System (CPS)* [LGC17; FMF+12; HFK+14a].

Model-Driven Engineering (MDE)* uses the abstraction mechanism to facilitate the development of nowadays software [Sch06; Ken02; BCW17]. This methodology can be applied to different stages of software development. In this thesis, we focus on one of its paradigms: **models-@run.-time*** [BBF09; MBJ+09]. The state

[1] Transformers change the voltage in the cables.
[2] Data are sent through cables that also distribute electricity.

of the system, its environment*, or its behaviour* is reflected in a model*, used for analysis. Developers can use this paradigm to implement adaptive systems* [MBJ+09; HFK+14a].

In this thesis, we focus on the use of MDE* techniques, and more specifically, the models-@run.-time* paradigm, for the implementation of self-adaptive systems*. Adaptation processes use models* to have a deep understanding of the system, its structure and its behaviour, and its environment. Using the vocabulary of the research community, the model* represents the knowledge*. That is, **we studied the representation of the knowledge* of adaptive systems*, using the models-@run.-time* paradigm.**

1.2 Problem statement

During our study, we have identified several 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 pace. Five, structure and behaviour of systems have a time dimension.

1.2.1 Data are uncertain

Most fuses are manually open and close 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 always work with estimates** [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 hardware) always estimate the value and have a precision value due to the method of measurement [Met08; BMM14]. Humans are error-prone. And 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. 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 Global Positioning System (GPS)* may impact the understanding by, for instance, showing the user on the wrong road (compared to the real one). **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.

1.2.2 Actions have long-term effects

Reconfiguring a smart grid implied to change the power flow. It is done by connecting or disconnecting specific cables. That is, opening or closing fuses. As said before, technicians need to drive physically to fuse locations to modify their states. Besides, in the case of the Luxembourg smart grid, meters send energy measurement every 15 min, non-synchronously. 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 paces, it can cause repeated decisions.

Actions are never immediate, take time to be executed, and have long-term effects. Through this thesis, we refer to such actions as long-term actions*. In computer science, the definition of "immediate" is specific to a domain. For example, for graphical user interfaces, a response time of less than 200 ms is considered as immediate. However, while working at the processor level, the execution time of one instruction is measured at the nanoseconds scale.

Not considering this delay may lead to sub-optimal decisions. For example, not

considering the delay for the system to handle the migration of a virtual machine may lead to repeat decisions. We argue that **developers should take into account this delay if the frequency of the monitoring stage is lower than the time of action effects to be measurable.**

1.2.3 Systems may have emergent behaviours

Smart grid behaviour is affected by several factors that cannot be controlled by the grid manager. One example is the weather conditions. Smart grids rely on an energy production that is distributed over several actors. For instance, users, who were mainly consumers before, now produce energy by adding solar panels on the roof of their houses. The production of such energy depends on the weather, and even on the season[3]. Another example is the increasing adoption of electric vehicles, which de facto drastically increase the consumption of electricity during the night. Ignoring this characteristic of adaptive systems* may result in suboptimal situations that can be understood with difficulties.

System behaviour may be emergent. [ZA11] Different factors can explain this phenomenon. As for smart grids*, systems may evolve in a stochastic and uncertain environment. Or, some system, like those name ultra-large systems, are so complex that a few engineers can tame it. But, groups of engineers will have an understanding of some part of the system.

Despite the complexity, engineers still need to understand how the system is behaving or behaved. It will help them to optimise the global behaviour or to understand and repair errors. **Engineers need tooling support to trace back previous behaviours and or replay them.**

1.2.4 Different part of a system evolve at different paces

Every meter sends consumption and production data every 15 min. However, this collection is not synchronous. That is, all meters do not send their data at the same timestamp. The global system, which receives all data, has not thus a global vision

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

with the same freshness for all the part of the grid. Electricity data are very 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. When analysing the data, the process should consider this difference of paces, and estimate the evolution of data. Otherwise, they will reason over outdated data, which do not reflect the real situation. It may lead to suboptimal decisions.

Different parts of the same system may evolve at different paces. Some systems are heterogeneous in terms of hardware and software. This diversity results in different evolution or reaction paces. 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 paces, 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. In the worst case, some data would be outdated and cannot be used. **Solutions to seamlessly predict or estimate what should be the current state of these outdated elements are thus required.**

1.2.5 Evolution of systems is linked with time

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

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

Decisions are not made upon the last state of the system but how it evolves. The analysis process should thus navigate both in the structure of the system and its

behaviour over time. **Engineers need efficient tooling to structure, represent, query, and store temporal data on a large scale.**

1.3 Challenge

In this section, we present five challenges for the research community in MDE* and adaptive systems*. The last one has been published in our vision paper regarding time awareness in MDE* in [BHM+17].

1.3.1 Data are uncertain

enges:u-data⟩?

Data become a cornerstone piece to autonomously derive decisions from them, or at least to support decision-making processes. We argue that their uncertainty will impact all the development stages of software, from the design to the execution. Design techniques should provide mechanisms to help developers abstract and manipulating uncertain data. Control flows use data, for example, in the branching conditions. This branching should be redesigned to consider the uncertainty in the 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. 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 global 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)?

1.3.2 Actions have long-term effects

ong-term-act⟩?

Decision-making processes follow the growing complexity of software. They are more and more able to make not only decisions on the current state of the system but also its past and future ones. And this decision may also have long-term effects.

Due to this complexity, developers and users may misunderstand the decisions taken by a system. Plus, designers may neglect or underestimate the impact of a

decision. Moreover, as highlighted by Bencomo *et al.*, [BWS+12], systems should be self-explained. They should be able to explain the decisions made.

To achieve this vision and to help designers and users understanding the impact of a decision, we argue that the software engineering community should address the following question:

How to represent, query, store, and understand the impacts of long-term actions?

1.3.3 Systems may have emergent behaviours

allenges:ermger-bhv⟩? The growing complexity of systems also has another impact: they have emergent behaviour. This behaviour may be suboptimal and hard to understand by designers, who generally have a local vision of the system.

However, when this behaviour leads to failure, engineers still need to understand why and how to avoid a novel occurrence of the problem. Plus, as the behaviour might be suboptimal, they need to optimise it.

To reach this goal, engineers need tooling support to help them in their investigation process. In other words, the research community should answer the following global challenge:

How to understand, predict, and optimise emergent behaviours?

1.3.4 Different part of a system evolve at different paces

allenges:diff-paces⟩? Systems may be heterogeneous and diverse, in terms of hardware but also software. Due to this diversity, the different part of a system may evolve at different paces.

However, engineers may need to reason on a global view. They will thus need to deal with data that have different freshness but also with components that can have their behaviour changed at different spaces. While decisions are executed to optimise the global behaviour, the system may be in an inconsistent state or a less good state than the initial one.

When designing the adaptation process, engineers need thus to consider this difference in freshness. For example, they need to implement estimation functions to estimate the value of an outdated data. Plus, they need to consider the fact that the

system can be in an inconsistent state while being updated. To sum up, the software engineering community should answer the following global challenge:

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

1.3.5 Evolution of systems are linked with time

es:evol-syst)?

System behaviours are temporal: the execution of the different actions are made over time. Plus, the structure of a system evolves over time. Not only may the last state be important but also how it evolves. Engineers need thus to analyse this temporal evolution.

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

However, we notice that modelling, persistence, and processing of data evolution remain understudied. Thomas Hartmann started addressing these challenging in his PhD thesis [Har16]. The final global challenge, not fully addressed, is thus:

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

1.4 Scope of the thesis

:intro:scope)?

Among all the challenges described in the previous section, this thesis focus on three of them: data uncertainty* (Section 1.3.1), long-term actions* (Section 1.3.2), and emergent behaviours (Section 1.3.3). More precisely, we address three sub-problems of these challenges.

Managing uncertainty requires significant expertise in probability and statistics theory. The literature provides different solutions to manage uncertainty [Zad96; 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?

Adaptation processes may rely on long-term action* like resource migration in cloud infrastructure. The lack of information about unfinished actions and their expected effects on the system, the reasoning component may take repeated of 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 diagnose the self-adaptation process?

1.5 Contribution & Validation

?⟨sec:intro:contrib⟩?

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 4. Second, we define a meta-model, and we formalise it, of the knowledge of adaptive systems*. We present this contribution in Chapter 5.

Ain'tea: Managing Data Uncertainty at the Language Level This contribution addresses the challenge of the manipulation of uncertain data (cf. Sub-Challenge #1). We propose Ain'tea, a language able to represent uncertain data as built-in language types along with their supported operations. It contains a sampling of distributions (Gaussian, Bernoulli, binomial, Dirac delta function, and Rayleigh) that

covers the different data types (booleans, numbers, and references). We implement a prototype of the language, publicly available on GitHub[4]. We use a real-world case study based on smart grid*, built with our partner Creos S.A.. It shows first that our approach does not impact the conciseness of the language. Second, it highlights the feasibility and the advantages of uncertainty-aware type checking systems on the language level.

This contribution is under submission at the JOT Journal[5]:

- "Ain'tea: Managing Data Uncertainty at the Language Level", Mouline, Benelallam, Hartmann, Bourcier, Barais, and Cordy

A temporal knowledge meta-model This contribution addresses the challenge of reasoning over unfinished actions and of the 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 decisions impact 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 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 Computing[6] (ICAC) and at the ACM/SIGAPP Symposium On Applied Computing[7] (SAC):

- "Enabling temporal-aware contexts for adaptative distributed systems", Mouline, Benelallam, Hartmann, Fouquet, Bourcier, Morin, and Barais

[4] `https://github.com/lmouline/aintea/`
[5] `http://www.jot.fm/`
[6] `http://icac2018.informatik.uni-wuerzburg.de/`
[7] `http://www.sigapp.org/sac/sac2018/`

- "A Temporal Model for Interactive Diagnosis of Adaptive Systems", Mouline, Benelallam, Fouquet, Bourcier, and Barais

1.6 Structure of the document

We split the remaining part of this document into eight chapters. First, Chapter 2 describes the necessary background of the thesis. 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 3. Then, in ??, we detail the vision defended in this thesis. We present the arguments in support of it and those that are in opposition. Chapter 4 and Chapter 5 described our two contributions, which go towards the defended vision. The former details our language, Ain'tea, that integrates uncertainty as a first-class citizen. The latter explains our temporal metamodel that can represent past and ongoing decisions with their circumstances and effects. Finally, we conclude in Chapter 6, and we present a set of perspectives of our work.

2

Background

t:background\?

Contents

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

2.1 et 2.2 trop long !

Manque de théorie.

2.1 Adaptive systems*

sec:back:adapt-syst)?

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

The complexity of nowadays software systems comes with laborious, error-prone, and redundant tasks (installation, configuration, maintenance, *etc.*) for developers. Moreover, software systems can be executed in uncertain and evolving environments* [EM10]. Following the autonomic computing vision pushed by IBM engineers [Com+06], Kephart and Chess have established the basis for adaptive systems* [KC03]. Adaptive 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 self-adaptive system* [BSG+09].

Danny Weyns identified two principles for adaptive systems* [Wey19]: the internal and the external principle. 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].

Furthermore, each adaptive system* is composed of four elements: the environment*, the managed system, adaptation goals, and the managing systems [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 of the environment* or the system. To express such goals, different approaches have been defined, such as probabilistic temporal logic [CGK+11] or fuzzy goals [BPS10]. Finally, the managing system will use these goals to drive the adaptation of the managed system in response to

changes in the environment*. It thus continuously monitor the environment* and the managing system. Researchers use different names to refer to this element: autonomic manager [KC03], architecture layer [GCH+04], adaptation engine [ST09], reflective subsystem [WMA12], controller [FHM14].

In the literature, we can find different approaches to engineer adaptive systems* [GCH+04]. Among them, the most used one took its inspiration from control theory [BSG+09]: the feedback control loop. The common implementation is the Monitor, Analyse, Plan, and Execute over knowledge (MAPE-k)* loop [KC03; Com+06]. This loop is split into four phases: monitoring, analyse, planning, and executing. During the monitoring phase, all information of the managed system and the environment* are put into the knowledge. Based on the updated knowledge, the analyse phase detects any need for adaptation using the adaptation goals. If any, the planning phase computes the set of actions to adjust the managing system structure or behaviour. Finally, the executing phase completes the plan.

Danny Weyns have identified six waves of engineering adaptive systems* [Wey19]. First, *automating tasks* that focus on the automation of repetitive and error-prone management tasks of complex systems [KC03]. Second, *architecture-based adaptation* that defines architecture as the basis of supporting engineering of adaptive systems*. Third, *models-@run.-time** [BBF09; MBJ+09] that establishes a technique to link the adaptive system* with a model with a "causal connection". Four, *goal-driven adaptation* that emphasised on the specification of the system requirements, exposed to uncertainties, and those for the solutions. Five, *guarantees under uncertainties* that consider uncertainty as a cornerstone concern for adaptive systems* and that define techniques to ensure the adaptation goals and the functional correctness of the adaptation components. Six, *control-based adaptation* that stresses the use of the control theory to benefit from a robust mathematical formalism. Among these six waves, this thesis is part of the third one, models-@run.-time*, that we detail in the next section.

2.1.2 Models-@run.-time*

The adaptation process needs to have a deep understanding of the system and its environment*. Following the MDE* methodology[1], research efforts have led to the models-@run.-time* paradigm [MBJ+09; BBF09]. The paradigm defines a model* that is "causally connected" to the system. The model* abstracts the structure and the behaviour of the system, the environment*, and the adaptation goals. 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. Moreover, Morin *et al.,* [MBJ+09] define models-@run.-time* allow stakeholders to represent a set of configuration points that can be selected at runtime to implement the adaptation mechanism.

Using this approach, we can say that engineers use a model-based architecture. Morin *et al.,* described a possible solution that is characterised by three layers, four types of runtime models, and five elements [MBJ+09]. The three layers are: online model space, causal connection, and business application. The business application layer contains the logic of the system. It has sensors for monitoring and actuators (here called factories). This layer is connected to the online model space, which is platform-independent, through the causal connection layer. The online model space contains the four different runtime models: feature, context, reasoning, and architecture model. The feature model represents the different configuration point of the system. The context model abstracts the relevant part of the system and the environment*. The reasoning model links between the first two models by associating the features with a particular context. Finally, the architecture model specifies the different entities of the system. These models are exchanged by the five elements that implement the adaptation mechanism. First, the *event processor*, which implements the monitoring stage of the MAPE-k* loop, updates the context model with information received

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

through the sensors. Second, the *goal-based reasoner*, which implements the analyse phase, uses the feature and reasoning model to select the new features for adjusting the system and achieving the goals. Third, the *model weaver*, which implements a part of the planning stage, transforms the selected features into a new architecture model, checked by the *configuration checker*. Finally, the configuration manager, which implements a part of the planning stage and the executing one, computes and executes the sequence of actions to reach the proposed architecture model.

More generally, this solution stresses the use of a model layer, causally connected to the system, and used by the adaptation process. 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 5). 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.

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 meta-model (cf. Section 5.3.2).

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.

Temporality In dynamic data, sometimes 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 a noise that makes it deviate from its original 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 According to the literature, data sources are grouped into two main categories, either sensed (measured) data or computed (derived) data [PZC+14]. We refine this with an additional category called profiled. Pro-

filed data may be set either by a user (**profiled by a human**) or by an external system (**profiled by an external**).

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

:adapt:knowledge:req⟩ 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]: "(1) Statement that translates or expresses a need and its associated **constraints** and **conditions**, (2) **Condition or capability that must be met or possessed** by a system [...] to satisfy an agreement, standard, specification, or other formally imposed documents".

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

Action

In the IEEE Standards [III17], an action is defined as: "**process of transformation** that **operates upon data** or other types of inputs to create data, produce outputs, or **change the state** or condition of the subject software".

Back to adaptive systems, we can define an action as a process that, given the