INSTITUT POLYTECHNIQUE DE PARIS

NNT : 20XXIPPAXXXX

Thèse de doctorat

A generic and adaptive approach to Explainable AI in autonomic systems: the case of the smart home

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Instituat polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 24 juin 2022, par

ETIENNE HOUZÉ

Composition du Jury :

Nicolas Maudet LIP6, Paris Sorbonne Université	Rapporteur, Président
Christian Becker Université de Stuttgart	Rapporteur
Philippe Lalanda Université Grenoble-Alpes	Examinateur
Mohamed Masmoudi Institut de Mathématiques de Toulouse	Examinateur
Jean-Louis Dessalles Télécom Paris	Directeur de thèse
Ada Diaconescu Télécom Paris	Directrice de thèse
David Menga Ingénieur de recherche, EDF R&D	Encadrant, invité
Mathieu Schumann Ingénieur de recherche, EDF R&D	Encadrant, invité

A generic and adaptive approach to Explainable AI in autonomic systems: the case of the smart home.

Étienne Houzé

2022

Acknowledgements

Over the course of a 3-year work, there are many moments when things get complicated and the horizon darkens. I had the chance to be surrounded by extraordinary people who helped me through these years. This page is dedicated to them, even though thanking to their value them would require many more lines.

To my advisors Ada, Jean-Louis, David and Mathieu. You have been able to motivate me, to overcome my laziness and guide me towards improving myself. Ada and Jean-Louis, you knew how to let me discover and understand by myself the different concepts, difficulties and stakes of my topic. Your high expectations regarding writing quality, proof-of-concept implementations and theoretical advances in the question of explaining fueled my desire to get better and push further in the right direction. David, you made me discover many new technologies, applications, theories and challenges that I had no idea existed, opening the real of possible developments of my research. Mathieu, your dedication to this project and many others were motivations to continue and work towards larger scale applications and discoveries.

To Wassim. You helped me implementing the demonstrator, handling all low-level management and devices. Without your help, the prototype would not have looked the same, and the output would have been different. Most importantly, for three years, you have been my friend at EDF, sharing many interesting conversations about work, technologies and life.

To Philippe, Laurent, Nathalie, Sophie, Joëlle, Loys and all of the people from EDF. To Ned, Julien, Maroua, Marc, Thomas and all of the people from the DIG team at Télécom. You have been incredible colleagues and friends over the last three years. I cannot count how many entertaining and insightful discussions I shared with you during this time. You were the reason I enjoyed going to Palaiseau despite the three hour commute.

To my family, my parents, my sister, my brother-in-law, my aunt, my grandmothers. You helped me through the COVID lockdowns and always comforted me when I needed it most. To my dad who proof-read the entire manuscript: you highlighted more typos and stupid mistakes than I could count!

To Marie, my wife-to-be. There are some encounters that change the rest of one's life. Having the chance to meet you definitely changed mine.Everyday, your kindness, your caring and your love inspire me and make me a better man.

To all of you, and to all the others with whom I discussed, exchanged and worked over the last three years, I express my infinite gratitude. Without you, this experience would have been drastically different and less enjoyable.

Thank you.

Contents

1 Current outcomes, stakes and goals **11**
- 1.1 Introduction . . . 12
  - 1.1.1 Smart homes . . . 12
  - 1.1.2 Use cases . . . 13
  - 1.1.3 Presentation of the problem . . . 15
- 1.2 Explanations . . . 16
  - 1.2.1 Defining explanations . . . 16
  - 1.2.2 Explanations, causality and counterfactuals . . . 17
  - 1.2.3 Triggering Explanation . . . 19
- 1.3 Architecture and features of smart homes . . . 19
  - 1.3.1 Principles of self-adaptive systems . . . 19
  - 1.3.2 Realization of Autonomic Systems . . . 21
- 1.4 Proposed solution and outline of this thesis . . . 23

2 XAI: a short review **27**
- 2.1 General Context . . . 28
  - 2.1.1 Artificial Intelligence . . . 28
  - 2.1.2 Explainable Artificial Intelligence . . . 32
- 2.2 An overview of current XAI approaches . . . 34
  - 2.2.1 Interpretable by design . . . 35
  - 2.2.2 Post-Hoc Explanations . . . 37
- 2.3 Evaluation of XAI methods . . . 42
- 2.4 XAI and complex cyber-physical systems . . . 44

3 An approach to explanation **47**
- 3.1 The goals of an explanatory system for smart homes . . . 48
  - 3.1.1 User-wise perception . . . 49
  - 3.1.2 Systemic properties . . . 50
- 3.2 A generative model for argumentative dialog . . . 51
  - 3.2.1 Explanation and argumentative dialog . . . 51
  - 3.2.2 The Conflict-Abduction-Negation process . . . 52
  - 3.2.3 Observations on CAN . . . 56
- 3.3 Adapting CAN to the smart home: the D-CAS algorithm . . . 58
  - 3.3.1 Difficulties of adapting CAN to smart homes . . . 58
  - 3.3.2 D-CAS: Decentralized Conflict-Abduction-Simulation . . . 60

3.3.3 An example of D-CAS . . . 67
3.3.4 Analysis of the algorithm . . . 68
3.3.5 Handling multiple causes . . . 70

4 Architecture Description **73**
4.1 General Organization . . . 74
4.2 The Local Explanatory Component (LEC) . . . 78
4.2.1 An interface between SHC variables and D-CAS propositions . . . 79
4.2.2 Reasoning on the LEC . . . 82
4.2.3 A generic and adaptive platform . . . 85
4.2.4 Preserving knowledge locality and privacy . . . 87
4.3 The Spotlight . . . 88
4.4 Simulators . . . 90
4.5 Self-* capabilities . . . 91

5 Implementation **97**
5.1 Implementation choices . . . 98
5.1.1 Previous versions of the demonstrator . . . 98
5.1.2 Implementation choices for the third version . . . 100
5.2 Description of the demonstrator . . . 103
5.3 D-CAS as a tree algorithm . . . 108
5.4 Illustrative examples . . . 110

6 Abductive Inference **117**
6.1 Definition of the problem . . . 118
6.1.1 "Naive" abductive inference . . . 118
6.1.2 The difficulty of defining memorability . . . 119
6.2 Formalizing memorability . . . 119
6.2.1 Notions of Algorithmic Information Theory . . . 120
6.2.2 Memorability as a complexity gap . . . 122
6.2.3 Defining relative memorability . . . 125
6.3 Related Works . . . 127
6.4 Experiments and results . . . 128
6.4.1 Illustration of the approach . . . 128
6.4.2 Illustration in smart home setups . . . 129
6.4.3 A subjective metrics . . . 132
6.5 Integration into the explanatory system . . . 133

7 Conclusion **137**
7.1 Contributions . . . 138
7.2 Limits and possible future development . . . 139
7.2.1 Limitation of the underlying system . . . 140
7.2.2 User interaction . . . 140
7.2.3 Module implementation . . . 141
7.3 Final words . . . 142

List of Figures

1.1 Examples of smart devices . . . 12
1.2 Generic organization of an Autonomic System . . . 22
1.3 Different organizations of Autonomic Systems . . . 23
1.4 Principle of the architecture . . . 25

2.1 Principle of an expert system . . . 29
2.2 Principles of deep learning . . . 31
2.3 Publications related to XAI . . . 33
2.4 General principle of XAI . . . 34
2.5 A taxonomy of XAI approaches. . . . 35
2.6 Interpretability/performance trade-off . . . 36
2.7 Principle of LIME . . . 38
2.8 Output of feature relevance XAI . . . 39
2.9 Prototype, criticism and counterfactual . . . 41

3.1 Outline of an explainable smart home system . . . 48
3.2 The CAN process . . . 52
3.3 Different knowledge distributions . . . 59
3.4 Knowledge distribution in D-CAS . . . 61
3.5 Principle of the D-CAS algorithm . . . 62
3.6 Evolution of internal D-CAS states . . . 69

4.1 Layer Organization of the Explanatory System . . . 75
4.2 The interfaces required by D-CAS for the Spotlight and the LEC. . . . 76
4.3 Example of an explanatory system . . . 77
4.4 Twin-memory architecture of a LEC . . . 80
4.5 Event and predicates classes . . . 82
4.6 Sequence diagram of the processing of a request from the Spotlight by a LEC. . . . 83
4.7 The modular architecture of the LEC . . . 86
4.8 Organization of the Spotlight . . . 89
4.9 LEC Addition . . . 92
4.10 Typical use case of a LEC . . . 93
4.11 Deletion of a LEC . . . 94
4.12 Description of a SHC enriched with LEC information . . . 95

5.1 First version of the demonstrator . . . 99
5.2 Interface of the second version of the demonstrator . . . 100
5.3 A RPI and a NUC . . . 101
5.4 Smart home model used for the third version of the demonstrator . . . 104
5.5 General implementation principles . . . 106
5.6 GUI application of the demonstrator . . . 107
5.7 ExplanationTree class . . . 109
5.8 Simple D-CAS output . . . 112
5.9 A higher intensity triggers the simulation . . . 112
5.10 Handling of erroneous abduction . . . 113
5.11 Knowledge revision in a rationale . . . 114
5.12 D-CAS identifying a failure . . . 115

6.1 Principle of event retrieval . . . 123
6.2 Memorability of days: the wedding example. . . . 129
6.3 Time series data from the 4-room iCasa simulation . . . 130
6.4 Complexity and memorability of the 4-room house . . . 131
6.5 Memorability-based classifier ROC . . . 132
6.6 Memorability of events in the TV scenario . . . 133
6.7 Effect of vocabulary on complexity . . . 134
6.8 Memorability-based abduction . . . 135

List of Tables

1.1 The three different types of inference identified by C.S. Peirce. 17
1.2 Different examples of self-adaptation management in smart homes. . . . 21

3.1 CAN reasoning example 56
3.2 An overview of the different sub-methods of D-CAS 64
3.3 D-CAS rationale example 67

4.1 Communication scheme 78

5.1 Comparison between demonstrator's versions 103
5.2 Implemented devices 105

6.1 Relative memorability 133

7.1 List of contributions 139

Chapter 1

Smart homes explainability: current outcomes, stakes and goals

Summary

Smart homes are cyber-physical systems that aim to fulfill different goals (comfort, security, health-related, etc.) in an individual house. As the number of connected devices continues to increase, so does the system complexity and the number of potential problematic situations. Therefore, one should expect future smart home solutions to integrate self-explainability as a fundamental feature. However, no current solution proposes to generate explanations at runtime in the context of smart home systems. The purpose of this PhD thesis is to solve this issue by designing an explanatory system compatible with smart homes.

Two main issues can be isolated from this main goal. First, the definition of explanation can be hard to grasp, as it is a subjective notion that has long been studied in philosophy, epistemology and cognitive science. It is therefore necessary to precisely define the scope and aim of the explanatory system before going forth. Second, existing smart homes are autonomic systems that present self-adaptation capabilities. For instance, smart devices can be "plug-and-play": they are automatically integrated into an existing environment. Self-adaption is prone to surprising or unexpected behaviors that an explanaible system should be able to handle.

In this introductory chapter, we briefly present the main problem of this thesis by exposing the context and some examples illustrating the need for explanations in smart homes. Then, we further define the two main notions underlying in the issue of explanations in smart homes: the nature of explanation and a description of autonomic systems. This brief analysis of the problem allows to present a sketch of our solution that we will detail in the following chapters.

1.1 Introduction

1.1.1 Smart homes

In Jacques Tati's "Mon Oncle", a famous French movie from 1958, two of the main characters, Mr. and Mrs. Arpel, live in a then-futuristic house equipped with all kinds of controlled devices. This equipment allows them to live effortlessly, all the house-keeping tasks and maintenance work being performed by machines. This movie embodies the long-existing vision of "smart homes", where machines would allow for greater comfort and control over the house. However, in this same movie, the ideal smart home turns into a catastrophe when all devices start failing, showing the potential threats and issues of this vision.

More than 60 years later, the vision of this movie is close to completion. Technological advances have led to the development of a variety of "smart devices": smart heaters, alarms, lights, presence sensors, thermometers, etc. While terminology may vary between authors, most of these devices present connectivity, context-awareness and autonomous features (Silverio-Fernández, Renukappa, and Suresh, 2018).Figure 1.1 shows a few examples of currently available devices. Most of them can be remotely controlled via a central hub, often a smartphone or a tablet, offering control and monitoring to the house occupant.

Figure 1.1: Different smart devices available on the market: smart lights, a voice assistant speaker, smart switches, a presence sensor. Picture taken at the Smart Home Experience Lab (SHEL) at EDF R&D.

A smart home can be defined as a home equipped with various connected components, such as the ones displayed in Figure 1.1, and a network enabling communication

between these components to achieve predefined home-wide objectives (Ricquebourg et al., 2006; Park et al., 2017). This definition leads to diverse implementations, ranging from small sets of connected devices in a single room to fully-equipped buildings where all mundane operations like heater control, light and door management are autonomous. Smart homes can be seen as the declination of Internet-of-Things (IoT) for the individual house.

The benefits of smart homes can be various for their users. Precisely controlling power usage in the entire house can help optimize the overall consumption (Ricquebourg et al., 2006; B. Zhou et al., 2016). Extending smart homes to create ensembles of systems allows for the creation of "smart grids", which can help electricity providers to monitor and optimize energy distribution and integrate privately produced electricity into the network (i.e. electricity produced by solar panels or wind turbines)(Bose, 2017; Frey, Diaconescu, Menga, et al., 2015). Benefits also include added security with connected alarms and health-related technologies, where smart homes can offer added autonomy to their users (Austin et al., 2016). Also, many smart home providers advertise ease-of-use and effortless control of the house as one of the main benefits of their systems.

However, despite these promised benefits, the adoption of smart homes remains lower than expected (Yang, W. Lee, and H. Lee, 2018). A German survey from 2017 shows that only one in three respondents are interested in smart home equipment (Zimmermann, Ableitner, and Strobbe, 2017). This low figure can be interpreted as the results of several drawbacks. (Zimmermann, Ableitner, and Strobbe, 2017) find that most cited reasons are vulnerability issues, privacy-related question and lack of understanding. In a survey conducted in South Korea, (Yang, W. Lee, and H. Lee, 2018) finds that the perceived automation level plays an important role in the perception and desirability of smart home technology. Aside from the topic of smart homes, these topics are identified as major concerns for the future of AI in general and its acceptability (Marcus, 2020).

1.1.2 Use cases

To illustrate this perceived lack of understanding and controls in smart homes from their user's perspective, we present a few illustrative examples. These examples come from field knowledge of various EDF collaborators, as well as from internal discussions and reflections. They will be re-used as typical use cases throughout the rest of this thesis.

Example 1.1

A room in a smart home comprises a temperature regulation system, air quality control, presence sensors, motorized windows and window shades and adaptive lighting. It is the middle of the day. To the occupant's surprise, the room temperature suddenly drops. Therefore, he/she wonders: *why is it cold here?*

This case illustrate the difficulty of anwsering to the seemingly simple "why-question" asked by the user. Due to the number of connected devices in the room, the unusual temperature drop may come from different reasons: i) the temperature is low because a heater is faulty and is unable to heat as expected; ii) a thermometer measures an erroneous temperature and fools the control system into thinking the temperature of the

room is correct; iii) another occupant of the house, for instance the visiting stepmother, has decided to change the temperature target, causing discomfort to this first occupant; iv) outdoor temperature and the house thermal performance do not allow the system to produce enough heat to maintain an acceptable temperature; v) heating was reduced by the system to avoid consuming too much power. If the system is unable to provide an explanation, or at least to "point the finger" at the component responsible for the discomfort, the occupant can potentially take wrong decisions, thus hindering the benefits of the smart home system (Nomura and Kawakami, 2011; N. Li et al., 2020).

Example 1.2

As the user is working in his/her office, in the middle of a sunny spring day, the window blinds roll down. As he/she is surprised by this event, he/she decides to override the system's decision and rolls the blinds back up.

This second example further illustrates the potential harm a misinformed decision may cause on the system. Here, the system may have decided to lower the blinds to protect the room from the direct sunlight and avoid using the A/C. Unknowing of this reason, the user decides to override the command. Thus, the benefit of lower power consumption is lost. Had the user had access to this information, he/she may have acted differently and let the blinds down. We may also observe that in this example, the system's decision to lower the blinds could be due to misreadings from sensors or be the result of a motor failure. Without further knowledge, it is impossible for the user to appreciate the motives and make an informed decision.

Example 1.3

As the user comes back from work, he/she finds his/her home to be colder than expected. He/she inquires the system as to why this happens, and the system is unable to find an answer. In fact, the thermometer of the room reads a temperature that is within the predefined range of tolerance.

The discrepancy between the system's measures and definition of cold and the user's perception creates a conflicting situation which is at the root of the problem presented in this example. Here, an expected "correct" answer from the system to the user's request for explanation would be to identify the discrepancy and propose it as the cause of the problem, which supposes that the system is able to i) identify its notion of cold and confront it with the user's preferences; ii) consider this observation as a possible cause for the problem. In this example,a satisfactory explainable system would be able to handle to integrate the user's perception of cold, compare it with its own goal and use the difference in an explanatiory reasoning. This ability is close to the notion of *self-awareness*, that defines the capacity of a system to observe, model and reason about itself (Kounev et al., 2017).

Example 1.4

A room in a smart home is equipped with many sensors, including multiple temperature sensors. In case of a failure from one of these sensors, how will the system react? Should the user be warned if a single sensor failure does not impede on the system's ability to regulate the room's temperature?

Example 1.4 illustrates one of the many issues coming with the multiplication of devices. As an increasing number of connected devices are integrated into systems (Jain and Murugesan, 2021), the possibilities of failure follow the same trend. In this case, the failure of a single temperature sensor can either provoke an observable effect on the temperature control system, or it can be mitigated if the system handles such case. In both possibilities, what should the system expose to the user? Should he/she be noticed of the failure, even if it has no noticeable impact, risking overwhelming her under low-priority alarms (Maxwell et al., 2020a)? Would an explanatory system be able to analyze a defect and expose it as a possible explanation for a system's behavior?

1.1.3 Presentation of the problem

Currently, no available solution offers to generate explanations that would help the user overcome the interrogations that could arise in the presented example scenarios. As illustrated through the various examples above, smart homes offer challenging environments to explain. They are confronted with unique situations influenced by the configuration of the home, external factors, user preferences and the required coordination between various devices. The major companies of the field, like Google or Amazon, which already provide an AI "assistant" for smart homes, do not offer explanation capabilities. Their assistant usually treats explanation requests similarly to other questions from the user: parsing the question and providing the best answer found on a remote knowledge base.

How can one develop an *explanatory system* that can *generate* explanations to the user of a smart home? This question will be the central problem that this thesis aims to solve. By using the term "generate", we emphasize that the explanation provided by the system should be customized for the current context and constructed at runtime, rather than selected from a predefined knowledge-base. Doing so supposes finding means to preserve the different characteristics of smart home systems such as modularity and self-adaptation capabilities and integrating them into an operational explanatory system. At first glance, this starting problem raises two underlying related questions.

The first problem is to define the goal of the explanatory system. This step is necessary, in order to understand what the expected answer of the system should be and how it can be generated. What is meant by explanation, especially in the context of smart homes? Can we specify core characteristics of explanations that can be used to design an automatic approach to their generation?

Second, the architectural realization of the system itself. How would an explanatory system be organized? Should it be monolithic, centralized or decentralized? What should be the architectural features and goals of the explanatory system? To what extent should it follow the organization of the existing smart home system? Here, the main stakes are

the integration of new devices and the potential upscaling from smart homes to smart buildings with hundreds or thousands of rooms and devices.

1.2 Explanations

Before going further into the resolution of the problem stated by this thesis, it is necessary to better understand the main term of the subject – explanations.

In a standard modern dictionary, one can find *explanation* defined as *"the details or reasons someone gives to make something clear or easy to understand"* (Cambridge Dictionary, 2020). While adequate for everyday use, this definition is based on subjective notions that would require further clarification: "reasons", "clear", "understand". In this section, we aim to go beyond this definition by using insights from philosophy, epistemology and cognitive science. Our goal is to define and identify distinctive elements of explanations from which we can start building an explanatory engine: a system that can generate explanations for a given situation.

Since research in these fields is active and we do not have the required expertise to settle ongoing debates, we present only established theories that are relatively consensual. We will therefore rely on this vision of explanation to guide and motivate the choices of implementation throughout this thesis.

1.2.1 Defining explanations

While used everyday, explanations have also been a topic of research in philosophy for a long time: Aristotle presented in his works a vision of explanations centered around the rhetoric and the discourse one had to produce in order to convince his partner (Trout, 2002; Hoffman and Klein, 2017; Maxwell et al., 2020b). As such, explanation serves as a tool to expand one's knowledge on a topic. (Lewis, 1987) defines explanation as *"someone who is in possession of some information about the causal history of some event [. . .] [trying] to convey it to someone else"*.

To formalize explanations, two main constituents are commonly distinguished: the *explanandum* is the object of the explanation, the description of the phenomenon that is to be explained; the *explanans* is the set of sentences which constitutes the explanations, and from which the explanandum can be logically derived. This basic vision of explanation drives the vision of scientific methodology usually found in the 20^{th} century (Hempel et al., 1965; Popper, 1963; Woodward, 2019).

Explanation and inference

Explanations are closely related to inference, i.e. the processes allowing one to acquire new knowledge from known premises. Philosopher C.S. Peirce famously identifies three main types of inference (Peirce, 1931): *Abduction*, *Induction* and *Deduction*. Examples for each of these inference processes are shown in Table 1.1.

> **Deduction** is the process of going from sound premises to their logical consequences. Deduction is a *valid* inference process, as the soundness of the premises

Type	Example	Validity
Deduction	All men are mortal; Socrates is a man; Therefore Socrates is mortal	Yes
Induction	All ravens I've seen are black; Therefore all ravens are black	No
Abduction	The grass is wet; When it rains, the grass gets wet; Therefore it must have rained	No

Table 1.1: The three different types of inference identified by C.S. Peirce.

entails the soundness of the consequence. While it is the pillar of logical and mathematical reasoning, it lacks the ability to expand knowledge, as the consequences are formally subsets of the premises, hence less strong.

Induction is also known as generalization, going from the experience to infer a general rule. Note that this inference is not valid. In the example from Table 1.1, if there exists a single white raven on Earth that I have never encountered, then the consequence is wrong while the premise remains true. In the classical view of scientific methodology, induction allows to formulate hypotheses by generalizing a sample of observations (Hawthorne, 2021).

Abduction can be defined as "inference to the best hypothesis"[1] (Magnani, 2011): it consists of inferring the hypothesis of an observed consequence. Similar to induction, this inference process is not valid: the inferred knowledge can be wrong. In the example from Table 1.1, the wet grass may be the cause of a sprinkler rather than the rain. Contrary to induction and deduction, abduction was formalized as a proper inference method lately, in the 20^{th} century. Since then, it has gained increasing recognition, becoming an important element of the hypothetico-deductive scientific method developed by Popper (Popper, 1963): scientific knowledge is gained through consecutive operations of abduction, testing and deduction.

Overall, abductive inference is closely related to explanation, as the latter induces the acquisition of knowledge by the explainee. (Hoffman and Klein, 2017) identifies three main kinds of explanations: abductive explanations, retrospection and prospect on. While abduction focuses on proposing an actual cause as an explanation, retrospection uses an alternate world in which neither the cause nor the consequence occurred as an explanation. Prospection, on the other hand, looks into the future and explains a situation by unveiling its potential consequences.

1.2.2 Explanations, causality and counterfactuals

The notion of causality appears to be central in the question of explanation. More often than not, the two notions are confounded in everyday speech (T. Miller, 2018; Hoffman and Klein, 2017). However, one can view explanations and causality as two distinct yet complementary notions: causality is a relation between a cause and an effect, and the

[1] Other definitions consider abduction as being limited to the proposal of hypotheses, separating it from the selection of the best. However we consider here both operations as part of abductive inference.

nature of this cause; while explanation is a process answering to a request (Hoffman and Klein, 2017).

The notion of a causal link between a cause C and its effect E is difficult to define at first glance: the wide array of possible interactions make it hard to determine a formal and universal canvas for causality. In the 20^{th} century, the progress of the scientific method led to the study of causality as a mechanism. The first theories from Hume were refined over the course of the century, notably by Pearl (Pearl and Mackenzie, 2018).

This led to the formalism of Structural Causal Models (SCM) (Peters, Janzing, and Schölkopf, 2017), in which variables can be either *exogenous* or *endogenous*. Exogenous variables are outside of the model, while the value of endogenous variables can be obtained by applying predefined state function f to a restricted set of other variables (exogenous or endogenous). Notably, the functions f are *independent* from the values of the different variables of the models. This theory allows for the formalization of causal interventionism (Pearl and Mackenzie, 2018). The main take of interventionism is the notion that simple correlations and joint observations of C and E are not sufficient to define a causal relation from C to E. Rather, it requires the consideration of the *counterfactual*. The counterfactual from causality is related to the *retrospection* described by (Hoffman and Klein, 2017), where a counterfactual is given as explanation.

To illustrate the notion of counterfactual, we can use the sentences from Example 1.5: if I observe that there is no roof over my lawn, that it did not rain last night and that the grass is dry ($\neg r, \neg p, \neg q$). In this situation, a counterfactual can be: had it rained last night, the grass would have been wet. To formalize this intuition, a counterfactual $p > q$ is defined as follows. In the current world, p is false; in all possible closest worlds where p holds, q also holds (Lewis, 2013). The notion of the *"closest"* possible worlds is important: in our example, the hypothetical world where it rained and there is a roof over my lawn can be considered further away, and therefore does not contradict the counterfactual $p > q$. (Ginsberg, 1986) proposes a formalism to understand this notion of closeness of possible worlds in terms of logical extensions.

Example 1.5

I consider the following logical sentences:

p It rained last night

q The grass is wet

r There is a roof over my lawn

Another possibility to define counterfactuals is to rely on probabilities (Pearl and Mackenzie, 2018; Menzies and Price, 1993): in this case, $p > q$ if $P = p$ is not observed and that the occurence of $P = p$ increases the probability of $Q = q$ to be observed. This definition is closer to the *do-operation* defined by Pearl Pearl, 2009, which proposes to compare $P(Q)$ to $P(Q \mid do(Q = q))$.

1.2.3 Triggering Explanation

Why do people need explanations? We have seen different motivations in Examples 1.2 and 1.1: in the first case, the user inquires about a unusual situation, while in the second she wants to know the reason behind an apparent failure of the system. According to (T. Miller, 2018; Hoffman and Klein, 2017; Lipton, 1990), both these explanations are *contrastive*: the request tries to make up for a perceived discrepancy between the *fact*, i.e. the observed situation, and the *foil*, the situation that the person expected, desired, wished or takes as a reference.

Contrastive explanations are found to make up most of everyday's explanations: in most *why-questions*, (Lipton, 1990; Lombrozo, 2012) the *foil* is explicit but can be formalized. For instance, in Example 1.1, in the request *"Why is it cold here"*, there is a contrast between the perceived cold and the expected regulated temperature that the control system was supposed to achieve. In Example 1.2, the explanation provided by the system should answer why the system decided to lower the blinds, as the user instead expected the blinds to stay open.

However, as contrastive explanations can account for most mundane situations, the identification of the fact and the foil it contrasts against is not always trivial. Depending on the situation, contrastive dimensions and definitions can vary: (Van Bouwel and E. Weber, 2002) identify three distinct kinds of possible contrasts: i) Property-contrast ("why does a have property P instead of Q?"); ii) Object-contrast ("why does a have property P while b has Q?"); iii) Time-contrast ("why does a have property P at time t, rather than Q at time t'?'). Which one of these contrasts is actually relevant depends on the context and the user.

1.3 Architecture and features of smart homes

The second key element of the main problem addressed in this thesis is the architecture of an explanatory system for smart homes. As stated in section 1.1.3, our goal is to design a system that would preserve the main characteristics of smart homes such as self-adaptation, scalability and runtime integration. In this section, we overview these different properties and the architectural features enabling them in order to apprehend them and design the explanatory system accordingly.

1.3.1 Principles of self-adaptive systems

Smart homes are cyber-physical systems in which several components communicate and operate together towards the completion of high-level goals (Ricquebourg et al., 2006; Mekuria et al., 2019). Similar organizations can be found in all application domains: web infrastructures, database management systems, industry operations, etc. Their predominance comes from the rapidly growing number of connected devices and offers high levels of performance, scalability and adaptation (Weyns, 2019). However, complexity sometimes comes from the scale of the system, up to the point where human intervention becomes difficult and costly. For this reason, the paradigm of Autonomic Computing was introduced. It proposes to design systems that are "able to manage themselves

given high-level goals" (Kephart and Chess, 2003), the objective being to minimize the need for human intervention at runtime (Philippe Lalanda, McCann, and Diaconescu, 2013). The vision of Autonomic Computing led to the realization of Goal-Oriented Computing, where high-level system-wide objectives "cascade" along the system to low-level specialized objectives (Diaconescu, Frey, et al., 2016).

Autonomic Systems present four main properties that have been defined in (Kephart and Chess, 2003; Philippe Lalanda, McCann, and Diaconescu, 2013). First, Self-configuration is the capacity of the system to automatically find the appropriate settings following a change to ensure a continuity in service. Second, self-optimization represents the capacity of the system to identify the possible means of improving its performance within its context and making relevant changes towards this aim. Third, self-healing is key in maintainability and robustness of complex systems: it denotes the ability of the system to react to internal or external damage, such as a component's failure. Fourth, self-protection is the ability to preemptively make required changes in order to prevent damage to the system of hindrance to the system's goals. These four characteristics are often grouped together under the denomination of *self-management*, or sometimes *self-adaption*, which denotes the ability of a system to "modify itself in response to changes in its operative environment" (Weyns, 2019; Kephart and Chess, 2003; Krupitzer et al., 2015).

The notion of Self-Adaptive System (SAS) is often found in literature and can sometimes be confounded with Autonomic Computing (Weyns, 2019). However, we can make a distinction between the two approaches. On the one side, Autonomic Systems focus on minimizing the need for user intervention by designing a system that can manage itself, letting the user focus on higher-level goals (Philippe Lalanda, McCann, and Diaconescu, 2013). On the other hand, the focus of SAS is to design software and systems that can handle changes of various kinds (configuration, environment, goals, etc.) without hindering their overall performance and without requiring user intervention (Krupitzer et al., 2015).

Smart homes benefit implement the paradigm of Autonomic Computing. As they are in direct relation with a physical environment, the house, they form a Cyber-Physical System (CPS) (Y. Liu et al., 2017). The control system therefore has to deal with the possible variations coming from external factors of the physical environment: changes in weather conditions, day/night cycle; as well as changes coming from interactions with the user. As such, they benefit from all four previously exposed SAS properties. Table 1.2 illustrates these properties in smart-home context. These properties are visible in modern smart homes devices, for instance where Amazon or Google Assistant are able to detect and integrate compatible devices from other manufacturers, following the "plug-and-play" principle (B. Miller et al., 2001). This capacity contributes to the adoption of smart homes and other smart devices as they can be integrated seamlessly into existing systems, from the end user's perspective.

The different self-adaptation properties can be further classified, based on several criteria (Salehie and Tahvildari, 2009). This taxonomy relies on the "5W + 1H" questions: *When* to adapt, *Why* do we adapt, *Where* do we implement change, *What* kind of change is required, *Who* should adapt and *How* is the adaptation performed? This leads to a

Property	Example
Self-configuration	When adding a new device into a Smart Home System, the home assistant is able to detect it and configure the device's location, communication protocols, goals and variables according to the existing configuration of the entire system and the data exposed by the device.
Self-healing	If a thermometer fails and reports erroneous temperatures, either the thermometer itself or the smart home assistant will identify a potential issue and alert the user, asking for replacement of the device.
Self-optimization	If the system observes that one heater is far less efficient, energy-wise, than the others, it will reduce its priority to prefer the use of other heaters in order to reach the temperature objectives set for the room.
Self-protection	If an attack from the exterior or a damage on a device jeopardizes the integrity of the system or affects its whole performance, the system is able to adapt accordingly, e.g. by temporarily revoking network access or turning off the compromised components

Table 1.2: Different examples of self-adaptation management in smart homes.

variety of different self-adaptation implementations which are reviewed in (Weyns, 2019; Krupitzer et al., 2015).

At the core of self-adaptation are the capabilities of *self-observation* and *context-awareness* (Salehie and Tahvildari, 2009). Self-observation is the capacity of a system to monitor its own state and behavior (Hinchey and Sterritt, 2006; Kounev et al., 2017), while context-awareness denotes the ability of the system to monitor and know about its operative environment (Hinchey and Sterritt, 2006). These two functions rely on observation capabilities from the system, which can be implemented in various ways.

1.3.2 Realization of Autonomic Systems

Autonomic System components are typically classified into two categories. Managed Resources (MR) cover the elements, sub-systems and components that are tracked and monitored. Autonomic Manager (AM) is the part of the system that is responsible for observing the MR and monitoring their changes. Thus, the AM can make the necessary modifications to the MR to adapt it to external or internal factors (Weyns, 2019; Kephart and Chess, 2003; Kramer and Magee, 2009). The goal of Autonomic Computing is that the entire adaptation process requires minimal user intervention, which allows to focus user intervention on high-level specifications of system objectives or management strategies. Figure 1.2 illustrates this generic architecture design. In some implementations, AM and MR are merged into single autonomic components. This *internal* observation approach, however, is often regarded as inferior and more fragile (Krupitzer et al., 2015) and will not be further discussed here.

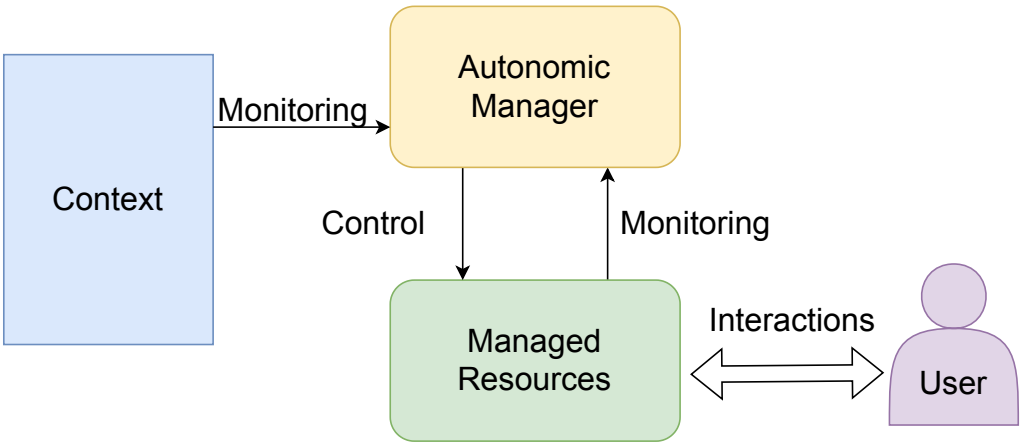


Figure 1.2: The generic organization of an Autonomic System: the Autonomic Manager monitors the Managed Resources and the Context, to provide seamless self-adaptation from the user's point of view.

In centralized architectures, all Managed Resources in the system are monitored and controlled by the same Autonomic Manager: see Figure 1.3a. While simple and easy to set up, this architecture suffers from maintainability and robustness: upgrading part of the system requires to update the central processing unit, which can affect all other parts of the system; a failure on this central component can make the whole system inoperative. Also, scalability is an issue if for large systems the Autonomic Manager is unable to provide the necessary computational power (Weyns, Malek, and Andersson, 2010).

To circumvent these issues, one may adopt a fully decentralized approach, such as depicted in Figure 1.3b. Every component in the system has its own autonomic management logic which observes the component and its environment and makes required changes and communications to the other components. While vastly improving the flexibility and robustness that the centralized approach lacks, full decentralization of adaptation is more difficult to set up and requires coordination among AM components to ensure coherent global behavior (Kephart and Chess, 2003).

For these reasons, a hybrid approach is often preferred, where several layers of AM components are used, the higher-level components monitoring and controlling several lower-level components (see Figure 1.3c). Inter-level communication ensures the required coordination between the various components of the system. Hybridization allows various possible configurations, (Frey, Diaconescu, Menga, et al., 2015; Weyns, 2019; Krupitzer et al., 2015) depending on the nature of the Managed Resources.

Smart homes often adopt either centralized or hybrid adaptation logic. An example of a centralized system would be an assistant controlling various connected lights. The assistant would discover and communicate with each newly installed light in the system, and update the other lights accordingly. For larger scale systems, such as an entire smart building, sub-systems are relatively autonomous in their self-adaptation capabilities, while communicating with a higher-level central interface for building-wide adaptation, such as policy changes or software updates.

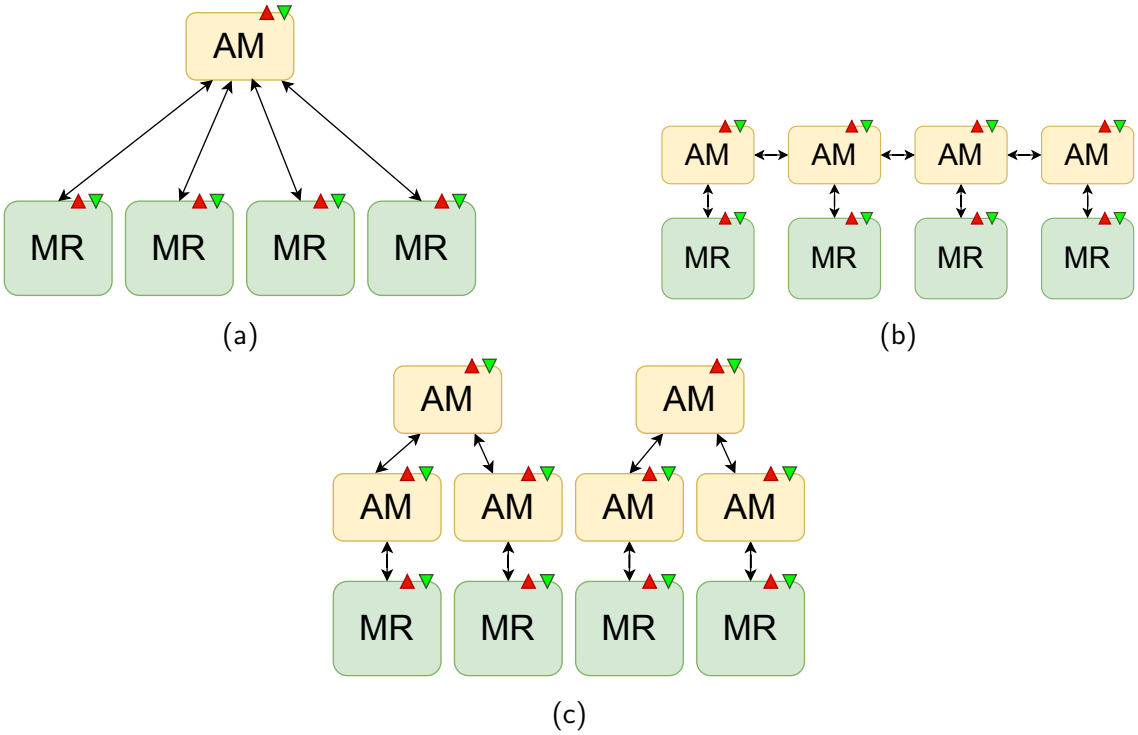


Figure 1.3: Different organizations of Autonomic Systems, as described by (Frey, Diaconescu, and Demeure, 2012) (a) shows a fully centralized autonomic system, while (b) shows an entirely decentralized logic and (c) shows a hybrid approach. Each component is able to receive goals at its level (green arrow), while exposing its state (red arrow).

Autonomic Managers typically follow the MAPE paradigm, which stands for *Monitor – Analyze – Plan – Execute* (Kephart and Chess, 2003). It splits the self-adaptation process into four distinct sub-operations. In hybrid approaches, it is possible that these four operations are not accomplished by the same entities. For instance, in the system depicted in Figure 1.3c, the top component operates the *Analyze* and *Plan* functions, while the lower level implements the *Monitor* and *Execute* functions. Previous work in Autonomic Computing also defines a variation of MAPE, named MAPE-K, where the final K stands for *Knowledge*. This formalism embodies the fact that the Adaptation Logic of an Autonomic System is able to represent its past observations, plans and actions into a knowledge base, which can be used subsequently to improve performance or response time (Weyns, 2019; Kephart and Chess, 2003).

1.4 Proposed solution and outline of this thesis

A first look at the nature of explanations and insights from System Architecture led to some refinements to the main problem posed in this thesis. After having defined the concepts and stakes of explanations, it appears that the goal of this thesis is to design a system that is coherent with the main characteristics of explanations. In Chapter 2, we provide a brief review of existing literature in Explainable AI. This overview highlights

the variety of available techniques and the current lack for approaches able to handle smart home systems.

In response, we identify and further define in Chapter 3 six main goals for a smart home explainable system: i) it must provide *contrastive* explanations for surprising situations; ii) these explanations must be shallow, in the sense that they are composed of few elements that are causally closely related to each other; iii) the explanatory reasoning must be transparent, clearly identifying which components are involved in the process; iv) the explanatory system must provide self-awareness capabilities to handle situations originating from self-adaptation of the autonomic system; v) the proposed architecture must be generic to handle different devices, organizations, contexts and situations; and vi) data locality and privacy must be protected. We propose a generic explanation generation algorithm, that we name D-CAS for "Decentralized Conflict-Abduction-Simulation". D-CAS identifies three main steps in an explanation process: conflict detection, abduction-based propagation and simulation of counterfactuals.

We present the architecture of an explanatory system that can enable our D-CAS algorithm in Chapter 4. The principle of this architecture is show in Figure 1.4. The explanatory system is made of two layers. A low-level layer observes the autonomic system components via their provided observation interfaces. This layer manages the diversity of knowledge and nature of the underlying autonomic system by providing standardized interfaces and knowledge representation. It manages self-adaptation at the local level, hence its qualification as "local" Explainable AI. This allows a higher-level layer, which is named the "Spotlight" to requests on-demand investigation on conflicting states when required. The Spotlight follows the D-CAS process to determine which state should be investigated. The trace of the process is then exposed via a user interface. In this organization, the Spotlight has no precise knowledge regarding the system's state and logic: it simply routes requests and organizes the reasoning.

In Chapter 5, we detail an implementation of the smart home explanatory system. The proof-of-concept demonstrator comprises a physical smart home model equipped with various sensors and controllers hosted on dedicated embedded devices. A different computer hosts the Spotlight and the user interface. This prototype allows to replicate some of the examples presented in this introductory chapter to illustrate how D-CAS can generate explanations meeting our criteria in typical use cases.

In Chapter 6, we tackle an issue that is specific to our target application of contrastive explanations in surprising or unusual situations. Given that few previous similar occurrences exist, most statistical or knowledge-based methods may be unable to provide relevant hypotheses for abductive inference. To allow our explanatory system to handle these situations, we propose a novel method for abduction, based on event memorability. We define a metrics of memorability based on Algorithmic Information Theory, and use events that are identified as remarkable as causal hypotheses.

Chapter 7 concludes the thesis by reviewing our different contributions and proposing perspective to continue the work towards autonomic and explainable smart home systems.

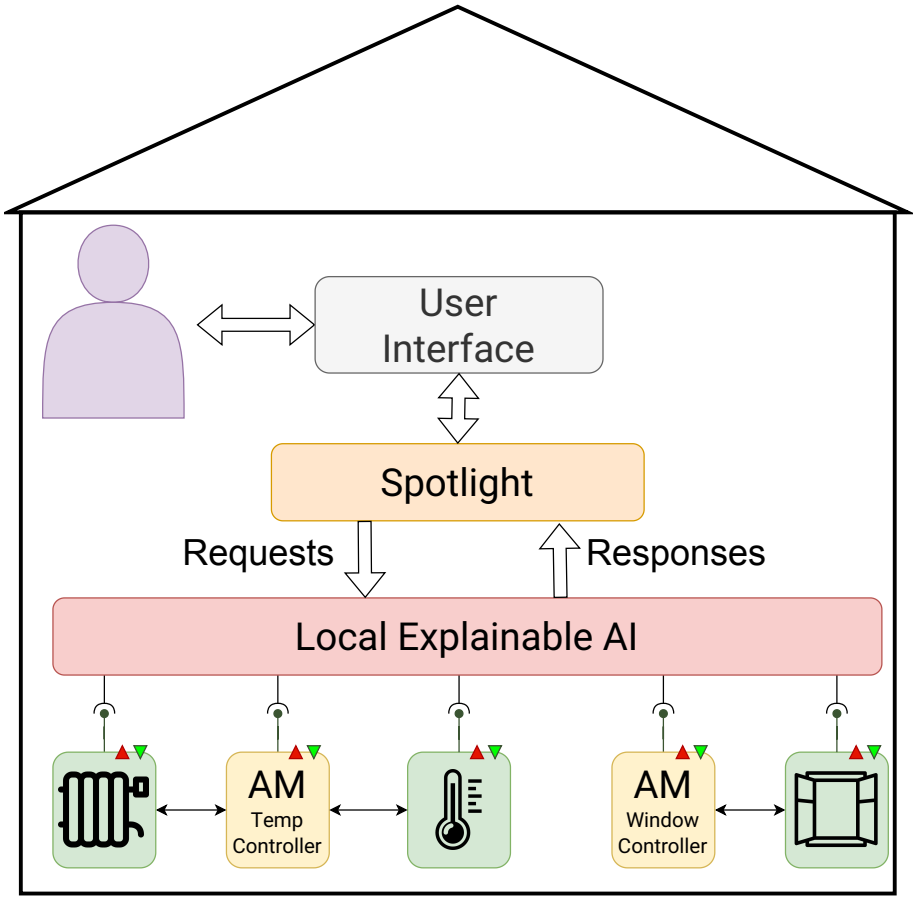


Figure 1.4: Principle of the architecture: a lower layer handles the diversity of resources and handles adaptation to changes. It presents a standardized knowledge representation to a higher-level layer, which coordinates the explanation reasoning by making requests.

Chapter 2

How to explain AI? A short review of modern techniques

Summary

The concept of Artificial Intelligence (AI) formally appeared in the 1950s. According to McCarthy, "intelligence is the computational part of the ability to achieve goals in the world". Since its inception, AI has enabled increasingly difficult tasks to be automated: planning, language analysis, speech generation, image recognition... As computers and AI models became more powerful, they started to outperform human in some tasks. However, with the increasing complexity, interpretability of AI models plummets, raising concerns about the understanding of critical machine-made decisions in areas such as autonomous vehicles, medical assistance or crime prediction. To answer these concerns, recent laws evoke a "right to explanation" for AI users, stating explanation as a critical aspect for the future development of AI.

Faced with this new challenge, the field of Explainable AI, or XAI, appeared in the past decade. Driven by project calls from the Defense Advanced Research Projects Agency (DARPA) as well as legal and economic incentives, many different tools were developed to "open the black box" of complex AI models. Due to the diversity of AI techniques, application domains, constraints and possible objectives implied by the term "Explainable", many XAI approaches exist, most of them designed for a specific task and setup.

In this chapter, we briefly review the history of AI to understand the stakes of explainability in the field. Then, we propose a short review of the current state of XAI, based on widely accepted taxonomies of approaches. Notably, we will focus on some of the most popular approaches to illustrate each of the main categories in the taxonomy. We also address the problem of explanation evaluation and the state of XAI in complex cyber-physical systems such as smart homes. This overview highlights the existing lack for system-wide explainability in smart homes.

2.1 General Context

2.1.1 Artificial Intelligence

Origins

The idea of machines being able to compete with humans on tasks considered to require intelligence is older that the first computers. In the 18^{th} century, a dulcimer-playing automaton impressed royal courts throughout Europe by replicating the precise movements of a human player (Cave and Dihal, 2018). While this automaton was merely a mechanical prowess and far from being called "intelligent", it showed the interest of building intelligent-like machines. Later on, with the beginning of the industrial era, the predominant use of machines influenced novels and the first movies: in Fritz Lang's Metropolis, humanoid robots feel and think as humans.

The development of computers in the wake of World War 2 opened up the possibility of automated tasks. By using computational power, machines became able to compute and solve equations, better and faster than any human. Faced with the supremacy of machines in specialized task, but still being far from intelligence, a need for clearer definition emerged. One of Computer Science's founding fathers, Alan Turing, designed a test which aims to discriminate intelligence. Stating that the ultimate test for a machine's intelligence was to fool a another intelligent being into being mistaken about its nature, he designed what is now known as the Turing Test, or the "imitation game" (Turing, 1950). In this test, a human, the examiner, is conversing with the test subject using a text terminal. The goal of the examiner is to discriminate whether he/she interacting with a human or a machine, while the purpose of the subject is to make the examiner thinks he/she is conversing with another human. According to Turing, an machine would be considered intelligent if and only if it was capable of pretending to be a human to another human, even for a few minutes.

In 1956, at the Dartmouth Conference, the term "Artificial Intelligence" was coined for the first time (McCarthy et al., 1955; Moor, 2006; Norvig and Russel, 2010; Nils J Nilsson and Nils Johan Nilsson, 1998). While this term is yet somehow unclear and its definition varies from authors, a common consensual definition would be that AI is an array of tools and computational methods by which machines perform tasks usually attributed to intelligence. This comprises, but is not limited, to understanding and generating Natural Language sentences, playing games, solving logical problems, recognizing images, visualizing spatial objects.

"Good old-fashioned AI"

The first attempts of creating Artificial Intelligence focused on a logical, symbolical approach (Norvig and Russel, 2010). Based on mathematical logic, computers are able to build a formal reasoning, going from premises to conclusion, planning and reacting to sentences. For instance, natural language processing can be done using a grammatical approach, by matching the various elements of the sentence with known syntactic elements. Based on Chomsky's research on formal grammars (Chomsky, 1956), natural language parsing methods have been developed since the 1960s. The Cocke-Younger-

Kasami algorithm, that was discovered almost 20 years later, illustrates this paradigm: it is capable, in polynomial time, of analyzing the correctness of a sentence within a context-free grammar and give the corresponding syntax (Harrison, 1978).

A notable example of symbolic AI is SHRDLU (Norvig and Russel, 2010; Winograd, 1972). Written by Winograd in the early 70s, this program consists of a small world populated with cubes, spheres, cylinders and pyramids of various colors. The program also comprised a text interface with which a user could interact, asking SHRDLU to describe the scene (*"the red cube is on top of the red cylinder, left of the green sphere"*) or to execute actions (*"Q: put the red cylinder on top of the sphere. A: This is not possible, as the red cube is already on top of the cylinder."*). At the time, SHRDLU was stunning in the sense that it seemed able to fully appreciate the logic of its geometric world and understand its user's requests. However it remained limited in its comprehension by the simplicity of the interactions, and was never scaled to a practical application.

As the performance of computers increased, real-life applications of AI became possible. New AI systems emerged, which were able to plan and make decisions in a specific but complex area of knowledge. These systems are named "expert systems" as they rely on three main distinct elements: a pre-established knowledge base, an inference engine and a user interface (Gill, 1995): see Figure 2.1. The knowledge base is directly imported from human expertise. Then, the machine can take advantage of its raw computational power to quickly infer consequences and answers in predefined contexts. In the 80s, expert systems proved useful in technical applications such as engineering (Maher, 1987; Rahman and Hazim, 1996) and health (Abu-Nasser, 2017).

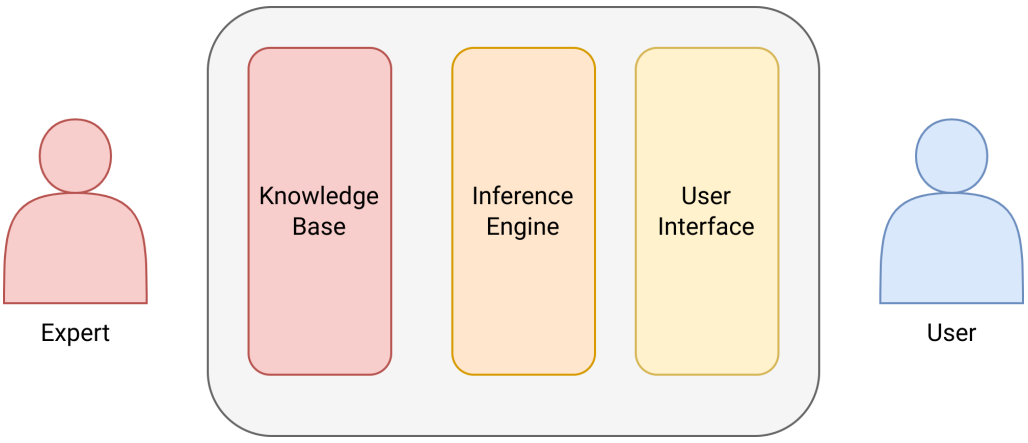


Figure 2.1: Principle of an expert system: the domain-specific expertise is located in the knowledge base, while the inference engine and the user interface are kept independent.

In hindsight, symbolic AI is often referred to as "Good old-fashioned AI" (Norvig and Russel, 2010), emphasizing its early time of development. While sometimes considered as old-fashioned, it still remains in use today, often used for meta-levels of more complex AI systems and coordination. This hybridization is particularly relevant in complex systems such as autonomous driving (Ning et al., 2021). In our approach, we adopt this hybrid view (see Chapter 3): our high-level explanatory engine is based on symbolic AI, resulting in a transparent approach, while the lower level tasks of analyzing data and detecting

conflicts, causes and consequences are left unspecified, allowing the implementation of data-driven approaches.

Machine Learning

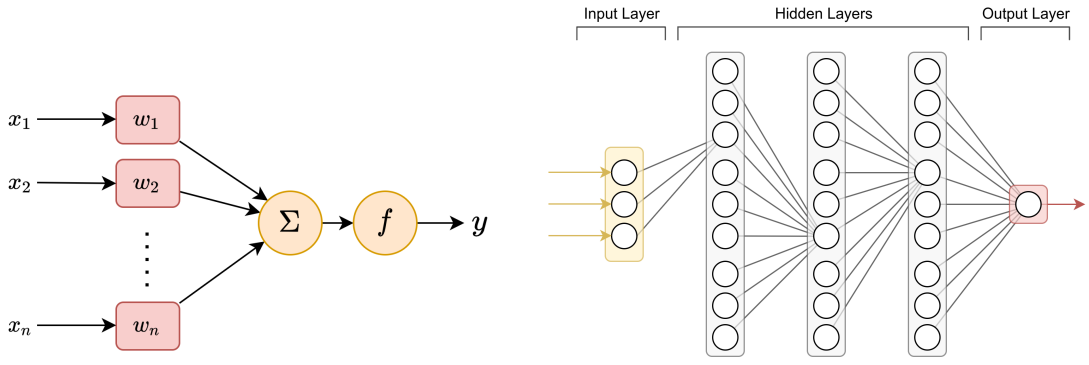
As the availability of data became higher and the processing performance more consequent, the philosophy of AI switched from expert systems to machine learning. Instead of relying on prerecorded human expertise, machines are initialized with a generic model which parameters can be tweaked based on acquired data. The main appeal of this approach is that automated adjustments can be made at runtime as new data is collected and used to update the model's parameters (Bishop, 2006).

Current approaches to Machine Learning (ML) are based on statistics and the estimation of parameters. They are divided in two main categories: *supervised* and *unsupervised*. In supervised learning, the system considers observations as samples (X, Y) of an unknown random phenomenon and tries to best estimate future outputs by searching the best approximation $Y = f(X)$ given currently available data. Example 2.1 illustrates the general idea of ML in a simple *regression* problem, meaning that the target Y is continuous. *Classification*, the other major kind of ML problem, considers situations where Y is discrete. Contrary to supervised learning, where the target y is known at learning time, unsupervised learning models problems where only X is available to the system. An example of an unsupervised learning task is clustering: the machine learns clustering criteria that can be later used for class predictions.

Example 2.1

Without knowledge of physics or thermodynamics, we develop a program to predict the mean temperature of a city based on its location (latitude and longitude) and its altitude. We create a training set consisting of measures from many known cities, resulting in the input set $X_{train} = \{(lat_{Paris}, long_{Paris}, Alt_{Paris}), \dots\}$ and the corresponding target values $Y_{target} = \{T_{Paris}, T_{London}, \dots\}$. Machine Learning proposes to find the function f which minimizes the prediction error for these known examples, then use f on other cities. Different functions and parameters can be used, resulting in many various models. For instance, simplistic model will only consider the altitude of the city and find the linear or exponential relation which best approximates gathered values, $T = f(alt)$. More complex models will integrate the position of the city as well, to improve the prediction performance, at the cost of the simplicity of the model, $T = f(lat, long, alt)$.

Often, Machine Learning is about finding a compromise between using over-complex models that best approximate the available data but show poor generalization capabilities (also known as *overfitting*) (Goodfellow, Bengio, and Courville, 2016b) and simpler but less accurate models. Since the set of possible functions f and parameters one could use is enormous, many different approaches exist which focus on families of functions, for which computable estimators exist: Linear Regression (Maulud and Abdulazeez, 2020), Support Vector Machines (Suykens and Vandewalle, 1999) (SVM), K-Mean Clustering (Hartigan and Wong, 1979), ...



(a) (b)

Figure 2.2: A neural network. A neuron (2.2a) takes a n-dimensional input, has a n-dimensional parameter $W = (w_1, \ldots, w_n)$ and returns $y = f(X \cdot W) = f\left(\sum_{i=1}^{n} x_i w_i\right)$, where f is a non-linear *activation* function. In a network, many of these neurons are organized in layers. Each layer's inputs are the outputs of the preceding layer (2.2b). This kind of configuration is said to be *fully-connected*, as each neuron is connected to all neurons in the previous and following layers. Various other neurons and networks topologies exist (Goodfellow, Bengio, and Courville, 2016a).

More recently, a branch of Machine Learning was particularly scrutinized due to its performance: Deep Learning (Goodfellow, Bengio, and Courville, 2016a). This approach emerged with the popularization of Neural Networks, which are shown in Figure 2.2. Networks are said to be "deep" when they comprise many hidden layers, which can result in gigantic models with billions of parameters. While the perceptron neuron model is old and can be traced back to the early ages of Computer Science, the popularity of Deep Learning came in the early 2010s only (Krizhevsky, Sutskever, and G. E. Hinton, 2012).

This recent popularity is mainly due to three factors. First, Deep Neural Networks are very flexible: their generic principle allows for all continuous functions to be approximated using neural networks (Goodfellow, Bengio, and Courville, 2016a). Second, despite the enormous number of possible parameters (the GPT-3 network from OpenAI requires around 175 billion parameters (Floridi and Chiriatti, 2020)), recent research found efficient algorithms to find optimal parameters based on Gradient Descent (Kingma and Ba, 2014). In addition, techniques have emerged to circumvent frequent problems encountered during learning (vanishing gradient). These heavy computations are made possible by the development and use of Graphical Processing Units (GPU) which are able to handle highly-parallelized operations and drastically reduce learning time, allowing from more complex and more accurate networks to be developed. Third, Deep Neural Networks, given their complexity, require an enormous amount of data to be properly trained. They benefited from the emergence of connected devices, improvements to data collection and processing techniques, storage and sharing devices, that made possible the construction of adequate datasets (Goodfellow, Bengio, and Courville, 2016a).

Some of the most impressive recent achievements of AI were done using neural networks. Among notable examples, we can cite AlphaGo, which was the first machine to beat a world champion at the game of Go, a game that has for a long time been thought

out-of-reach of AI given its complexity (Silver et al., 2017). Many applications of image analysis and manipulation also benefit from Deep Learning, for instance to improve image quality in video games using real-time de-noising and upscaling (Burgess, 2020). Popular Deep Learning tasks also include face recognition and image classification (Patil, Pandey, and Visrani, 2021; Goodfellow, Bengio, and Courville, 2016a) or Natural Language Processing (NLP). Here, the recent development of the Transformer layer () led to models such as OpenAI's GPT-3 or Google's BERT that demonstrate impressive text generation and analysis capability (Floridi and Chiriatti, 2020). The potency of Neural Networks opens future developments such as autonomous driving, since they offer real-time analysis of sensor and camera data decision-making, mostly using recurrent Neural Networks (Grigorescu et al., 2020).

2.1.2 Explainable Artificial Intelligence

Origins and goals

Over the decades of AI developments, the question of explainability has been an issue. The development of expert systems and their applications led to questioning the ability of these systems to be understood by their users. (Swartout, 1983) proposes in 1982 the XPLAIN system to create explanations by exposing some of the rules and knowledge used by the inference engine (Gill, 1995). However applications remained limited, due to the difficulty of maintaining the expert knowledge base and the limited domains these systems were used for, which reduced the need for explanations as most of their users already had proficient knowledge in the area. In 1992, the SWALE project (*SWALE Project* 1992; Leake, 2014) used case-based reasoning to provide explanations for mysterious events such as the sudden death of a racehorse named Swale. Despite its success, no major research followed, and explainability in AI remained a confidential topic.

However, the unprecedented growth in complexity, capability and usage of AI in the 21^{st} century brought forth new questions of explainability, fueled by concerns regarding the legal accountability of AI incidents (Doshi-Velez, Kortz, et al., 2017), data privacy (Manikonda, Deotale, and Kambhampati, 2018), security (Hossain et al., 2019) or performance (N. Li et al., 2020). In 2016, a major call for project from the DARPA (DARPA, 2016) resulted in a major resurgence in research interest for Explainable AI, or XAI. Figure 2.3 shows this effect in the number of XAI-related publications. The DARPA project call notably introduces model interpretability, privacy, legal responsibility among the goals of XAI. It presents a general principle, depicted in Figure 2.4 to highlight the distinction between standard AI and XAI: an explainable model is the result of the learning process, which can then be scrutinized by the end user.

XAI appears to be an important element for the development of future AI methods, be it "Trustworthy AI" (Wing, 2020) or "Robust AI" (Marcus, 2020): both visions integrate explainability as a key feature to enable trust between a system and its user. Trust was also cited as a motivation for the pioneering developments of XAI (Ribeiro, Singh, and Guestrin, 2016; DARPA, 2016). However, the relation between the explainability of an AI system and an increase of trust is not yet confirmed. While some studies show that

[1] `https://ieeexplore.ieee.org`, accessed October, 21st 2021

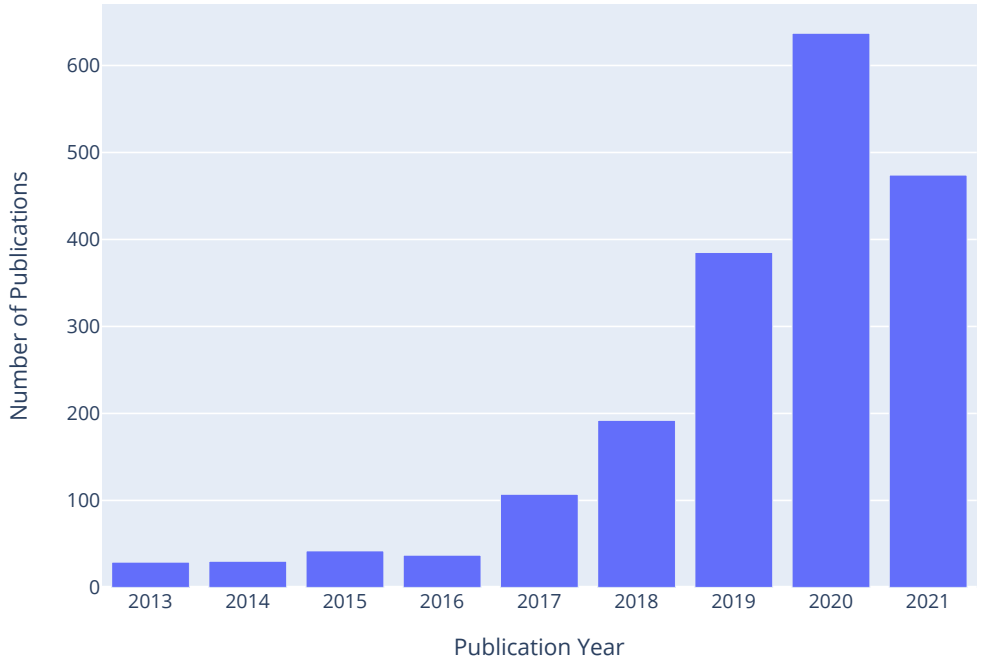


Figure 2.3: Number of publications found on the online library IEEEXplore[1] related to keywords "XAI", "Explainable AI" and "Interpretable AI". As all publications were not yet registered for year 2021, the corresponding number is temporary.

providing explanations to a user lowers the cost of mistakes (Glass, McGuinness, and Wolverton, 2008), others find that the correlation is unclear, varying depending on the situation and the user (Dhanorkar et al., 2021). In (Nothdurft, Heinroth, and Minker, 2013), the authors find that in some applications, the efficiency of the AI agent is the prime factor of trust. While we keep in mind these objections, we will, for the rest of the thesis, consider explanation as a desirable feature for AI systems, as this appears to be the general trend of the industry.

Terminology of XAI

With the important number of publications related to XAI, as shown by Figure 2.3, a variety of terms emerged to designate closely related yet different concepts, such as "explainable", "interpretable" or "transparent". These terms are often defined in literature reviews with some minor differences between authors (Arrieta et al., 2020; Adadi and Berrada, 2018; Doran, Schulz, and Besold, 2017; Rojat et al., 2021; Mohseni, Zarei, and Ragan, 2021). We will settle on using the same notations as (Arrieta et al., 2020). *Understandable* AI is regarded as the target of XAI: an understandable system is intelligible to its human user as a whole, without further need to detail its inner algorithmic or data processing units. By contrast, an *interpretable* AI model is a model that is, by

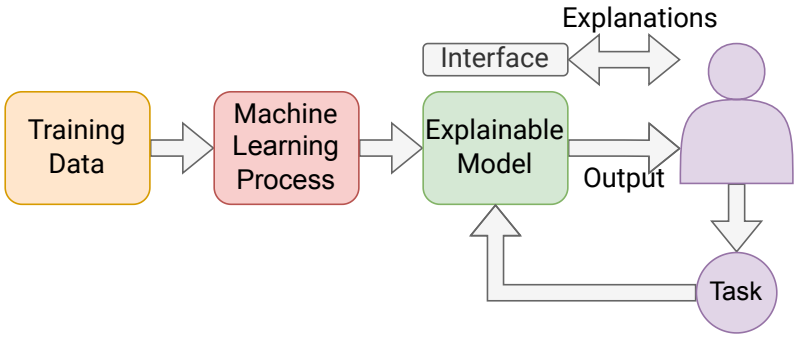


Figure 2.4: General principle of XAI: an explainable model is learned from training data. Then, as tasks are given by the user, this model is able not only to produce an output, but also to provide explanations for its decisions via an interface. Figure inspired from (DARPA, 2016)

design, observable and whose inner workings are understandable to a human user. On the other hand, an *explainable* model is an AI model which can generate explanations, i.e. reasons and motivations for a given decision. Note that with these definitions, an explainable system offers *post-hoc* understandably.

2.2 An overview of current XAI approaches

Given the variety of challenges and AI methods exposed in the previous sections, many different XAI solutions have been designed to tackle different aspects, challenges of the question. Figure 2.5 gives an incomplete overview of the different kinds of approaches one might use to explain an AI agent (Arrieta et al., 2020; Biran and Cotton, 2017; Guidotti, Monreale, Ruggieri, Turini, et al., 2018; Papastratis, 2021; Mohseni, Zarei, and Ragan, 2021). This classification's first distinction is based on the time of occurrence of the explanation: *post-hoc* approaches provide explanations after the decision is made, i.e. have access to the questioned point input and output to create their explanation, while *explainable by design* approaches consider explainability of their model before making decisions.

While the taxonomy presented in Figure. 2.5 is directly inspired from (Arrieta et al., 2020), we want to emphasize that many methods do not fall clearly into one category, and distinctions may be blurry. For this reason, (Adadi and Berrada, 2018) proposes an alternate classification, for instance differentiating the scope of the XAI method from its approach (intrinsic or post-hoc) and its specificity (model-specific vs model-agnostic), thus providing a 3-dimensional taxonomy of methods. The scope of an XAI method can be either: i) *local*, the model targets the explanation of single data points, e.g. justifying why a specific image was classified as a cat; or ii) *global*, the method tries to make the entire model interpretable, e.g. proposes a visual representation of the reasoning and of the criteria used by the AI agent to categorize images.

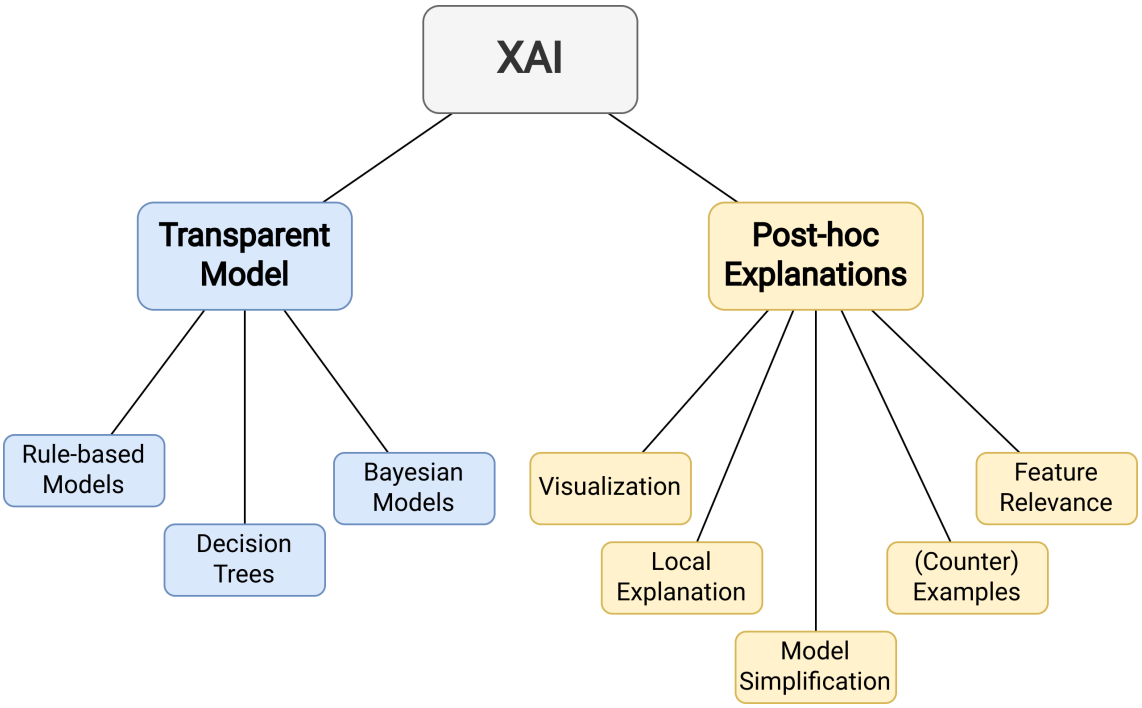


Figure 2.5: A taxonomy of XAI approaches.

2.2.1 Interpretable by design

Aside from proposing tools to generate post-hoc explanations for black-box AI models, XAI also advocates for the development and use of so-called transparent models (i.e. models which parameters are visible to the user) and interpretable models (i.e. whose inner working and decision is understandable to the user) (Arrieta et al., 2020; Adadi and Berrada, 2018). The original call from DARPA (DARPA, 2016) was not specific about the tools: it only emphasized the need for having access to understandable AI models.

Examples of interpretable models

With regards to the recent history of AI and Machine-Learning, many attention is currently drawn towards Deep Neural Networks, which are inherently complex and non-interpretable. However, this is not representative of the entirety of ML models: some of them can be apprehended, even by a relatively inexperienced human user (Arrieta et al., 2020). Notably, this is the case for rule-based models, which differ from the old-school expert systems in how they acquire knowledge: instead of being recorded by a human expert, their knowledge base is built using data extraction. While the extraction process can be complex to understand, the resulting rule base consists of logical rules that can be written in human-understandable fashion (Swartout, 1983).

Linear regression are among the simplest regression models in Machine Learning, and are often used as typical examples for novices (Bishop, 2006). As they consist in simple parameter optimization of linear combinations of variables, their predictions can be

easily understood by users versed in mathematics. Decision Trees are another popular ML model that is considered interpretable (Safavian and Landgrebe, 1991; Guidotti, Monreale, Ruggieri, Turini, et al., 2018). They consist of applying successive conditions on variables, such as threshold comparisons, thus resulting in a tree-like branching of tests where leaves represent the final class attribution. As these models apply successive tests, they are interpretable by human users.

k-Nearest Neighbors (or k-NN) models are a very popular classification method whose principle is intuitive enough to be considered interpretable (Bishop, 2006; Zhang, 2016). To assess which class a point belongs to, the algorithm observes the k nearest neighbors of that point. The majority class among these neighbors is then attributed to the point. While the metrics used to define the neighborhood can be hard to grasp, the results of k-NN clustering are often well understood (Arrieta et al., 2020).

A trade-off between interpretability and performance?

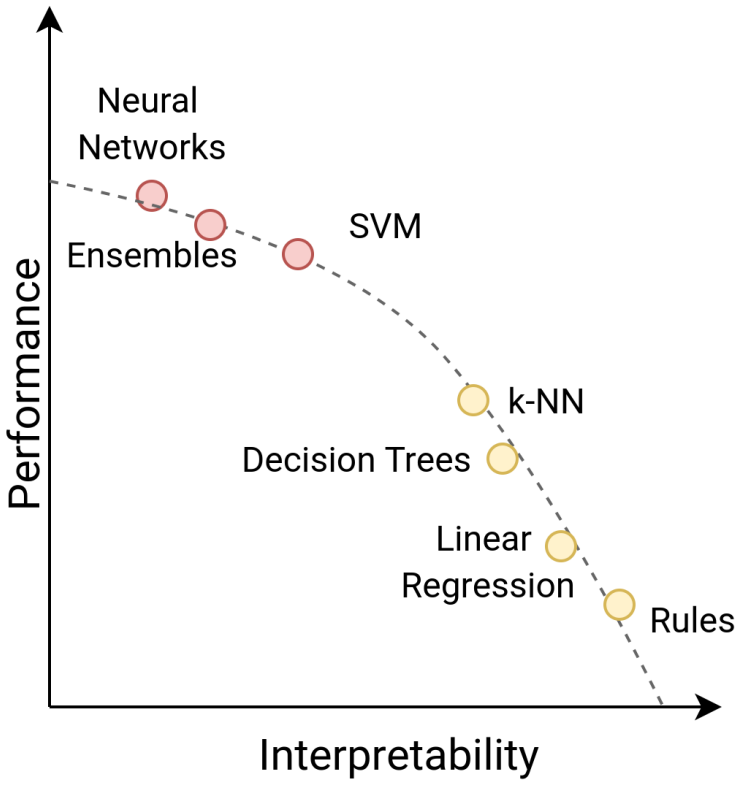


Figure 2.6: A general trade-off may appear between interpretability and performance of a model: the higher the performance, the more numerous the parameters, the less interpretable the model. Models considered as black-box are shown in red, while interpretable models are shown in yellow.

A common intake on interpretable models is that there exists some kind of trade-off between interpretability and performance (DARPA, 2016; Nesvijevskaia et al., 2021). Figure 2.6 depicts illustrates this trade-off, according to (Arrieta et al., 2020). This

phenomenon comes from the correlation between the simplicity of a model and its interpretability: the less parameters, the easier it is to understand a model but the less accurate it is. This observation is coherent with the general trend of AI where Deep Neural Networks outperform other methods in popular tasks such as image recognition or NLP, resulting in opaque models relying on millions of parameters.

However, as noted by (Rudin, 2019), this trade-off is not always true and can lead to the use of overly complex models, favored with the help of explainable techniques to improve their acceptance. For some cases where structures are present in the data, simpler models can yield equivalently good, if not better results. Think for instance of a linear regression opposed to a Depp Neural Network: for problems where the underlying structure is linear, the linear regression model can outperform the neural network while being far more interpretable.

2.2.2 Post-Hoc Explanations

Post-hoc explanations occur once the decisions is made by the AI agent. As such, they serve as a diagnostic tool and aim to identify the different criteria which were of prime importance for the decision. Historically, this kind of XAI has been the first to boom in 2016, with major contributions such as LIME (Ribeiro, Singh, and Guestrin, 2016) or SHAP (Lundberg and S.-I. Lee, 2017). In this category, different strategies exist, depending on the task to be explained, and the model used for the decision. Some approaches are *model-agnostic*(Arrieta et al., 2020; Guidotti, Monreale, Ruggieri, Turini, et al., 2018), meaning that they do not depend on the internal architecture of the model, but rather rely on observations of inputs and outputs to generate explanations. The opposite category is *model-dependent* XAI, which uses knowledge about the model to build their explanation. Model-agnostic methods are often regarded as being superior, given they are designed to be generalized to different models and/or applications (Adadi and Berrada, 2018). One can also argue that local explanations, as they are taking into account the context of the specified questioned decision, are closer to the notions of explanations presented in Chapter 1.

Feature Relevance

When a black box model classifies a data point in a given category, what were the most important features in that decision? Even if this is not a complete explanation, having access to this kind of information provides helping tools to apprehend and understand how the model works. To this extent, numerous approaches propose to highlight the feature that appear as the most relevant a model, either locally or globally. These techniques are referred to as *feature relevance* approaches in reviews (Arrieta et al., 2020; Adadi and Berrada, 2018).

A notable feature relevance approach is LIME (Ribeiro, Singh, and Guestrin, 2016), which stands for Local Interpretable Model-agnostic Explanations. LIME proposes to study the importance of features by generating points in the neighborhood of the questioned input and see the outcome of the model on these new points. The results are used to build a simple local approximation of the model, which can then be used to

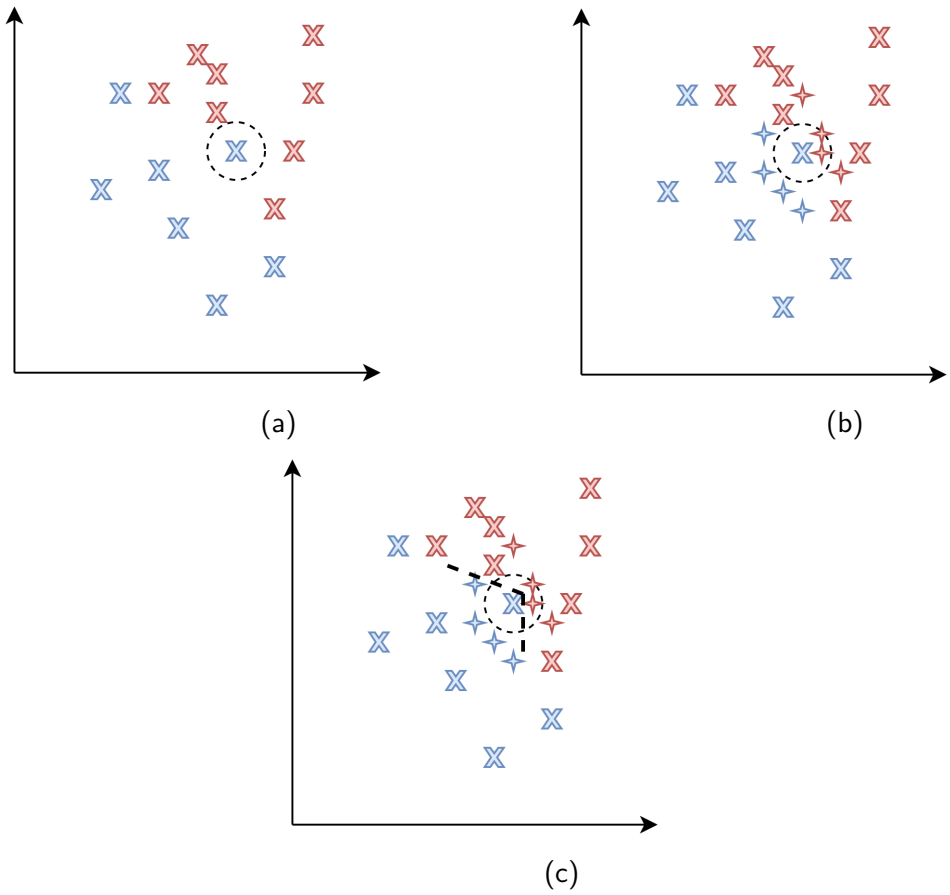


Figure 2.7: Principle of LIME. To investigate why the circled data point was categorized as blue (2.7a), LIME samples points around it and classifies them using the same model (4-pointed stars in (2.7b)). This results in a local approximation of the decision border, represented as a dashed line in (2.7c). This approximated border can then be used to identify the most relevant variables for the classifier in this area.

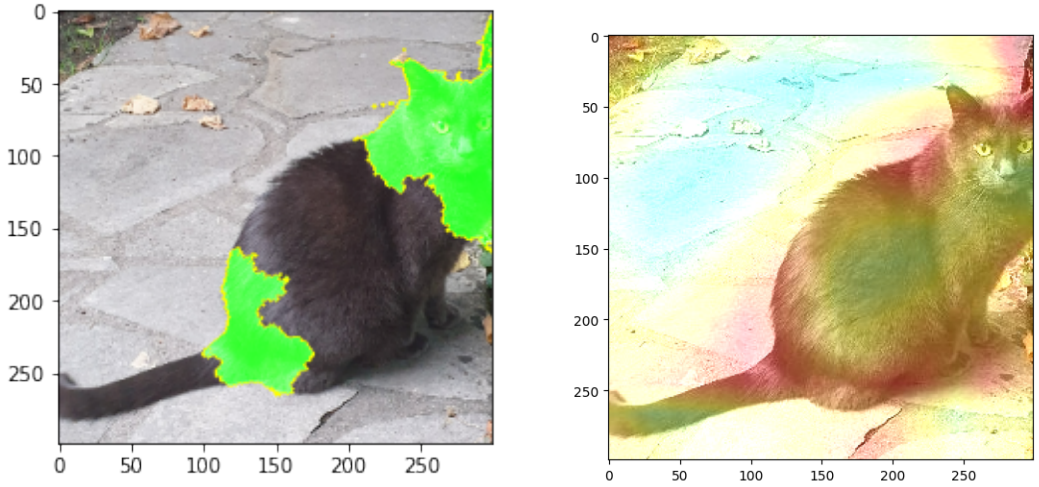
understand the impact of the different features. Figure 2.7 illustrates the principle of this approach.

Another notable feature relevance approach is SHAP (Lundberg and S.-I. Lee, 2017). This technique proposes to study the Shapley values of the different features used by the model. Shapley values are a concept from game theory (Shapley, 1953) where they are used to quantify the relative importance of the contribution of one player in a team for cooperative games. SHAP computes a similar metrics for variables in AI models and rank features according to this weight. The application of SHAP can result in either a local or a global explanation, as different domains can be used for computations (Lundberg and S.-I. Lee, 2017).

The output of feature relevance methods can be displayed visually, especially when the input is an image for tasks such as image classification. In this case, feature importance can be encoded as a heatmap over the original input image highlighting which areas or structures where prominent in the decision of the classifier. Both SHAP and LIME

propose such visual representations. Figure 2.8a illustrates an example output of LIME visualized for an image classification task[1]: the highlighter area corresponds to the most important features.

Other feature relevance approaches fully embrace the visual representation: it is the case of GradCam (Selvaraju et al., 2017), which, contrary to the aforementioned methods, is limited to neural networks. Grad Cam proposes to visualize the gradient of the parameters of a neural network for image classification and use it as a tool to highlight areas of interest in the picture. Figure 2.8b shows the output of GradCam on an image classification task[2]. Note that, like LIME for image classifiers, GradCam can be considered as visualization-based, depending on the taxonomy (Adadi and Berrada, 2018).



(a) (b)

Figure 2.8: Visual outputs of feature relevance approaches. (a) Visualization of the zone detected by LIME as the most relevant for the classification of the image as a cat. (b) Exposition of the gradient value using GradCam: areas with higher gradient value (in red) are supposedly more important for the model.

Visualization

Visualization of the data and the model can be key in understanding how an AI agent makes a decision. Data visualization techniques have been popular in Machine Learning before the development of XAI in 2016, as they provide an appealing and useful tool for ML practitioners to represent their model and illustrate their proposition in articles and textbooks (Bishop, 2006). Notably, with Neural Networks, visualization of layers, similar to what is presented in Figure 2.2b can quickly convey information regarding the architecture of the network. For model-specific, visualization of the different activated neurons in each layer, i.e. neurons whose output value is high, is used as explanation

[1]LIME code from `https://github.com/marcotcr/lime`

[2]GradCam code from `https://github.com/samson6460/tf_keras_gradcamplusplus`

for the model's output (Yosinski et al., 2015). Overall, most visualization techniques are model-specific, or data-specific: tools designed for neural networks are not portable to SVMs.

However, in some cases, visualization of the data can be used to explain the decision made by a model. For instance with classification tasks, a visual representation of tool like t-SNE or its derivatives can provide low-dimensional visualization of complex data (Maaten and G. Hinton, 2008). This can be sufficient to explain the classification of points as belonging to a given class, as the cluster organization appears in the representation (Chatzimparmpas, Martins, and Kerren, 2020).

A notable visualization-based XAI technique is Individual Conditional Expectation, or ICE (Goldstein et al., 2015). ICE proposes model-agnostic visualization of the relations between features of the model. It improves the existing Partial Dependence Plots (PDP) introduced in (Friedman, 2001): while PDP only shows the evolution of the average outcome with regards to the parameters, ICE offers a superposition of plots for different points, showing relations that would remain unnoticed in the aggregated average.

Model Simplification

Aside from feature relevance and visualization-based methods, a third approach to post-hoc explanation generation is to rely on model simplification. Here, the general principle is to approximate the black-box AI model with an interpretable secondary model (Arrieta et al., 2020; Ras, Gerven, and Haselager, 2018). The variety of possible interpretable models (see Section 2.2.1) offers numerous possibilities of approximation.

As neural networks are often considered the most complex models (Adadi and Berrada, 2018; Castelvecchi, 2016), many methods propose to approximate them with simpler models. (Confalonieri et al., 2019) uses a rule-based ontology to approximate classification results by a deep network. Diverse rule-extraction techniques exist depending on the target model (Ras, Gerven, and Haselager, 2018). Others rely on building decisions trees from neural networks (Augasta and Kathirvalavakumar, 2012) as they are considered potent and generic yet interpretable models (Safavian and Landgrebe, 1991), especially for classification tasks.

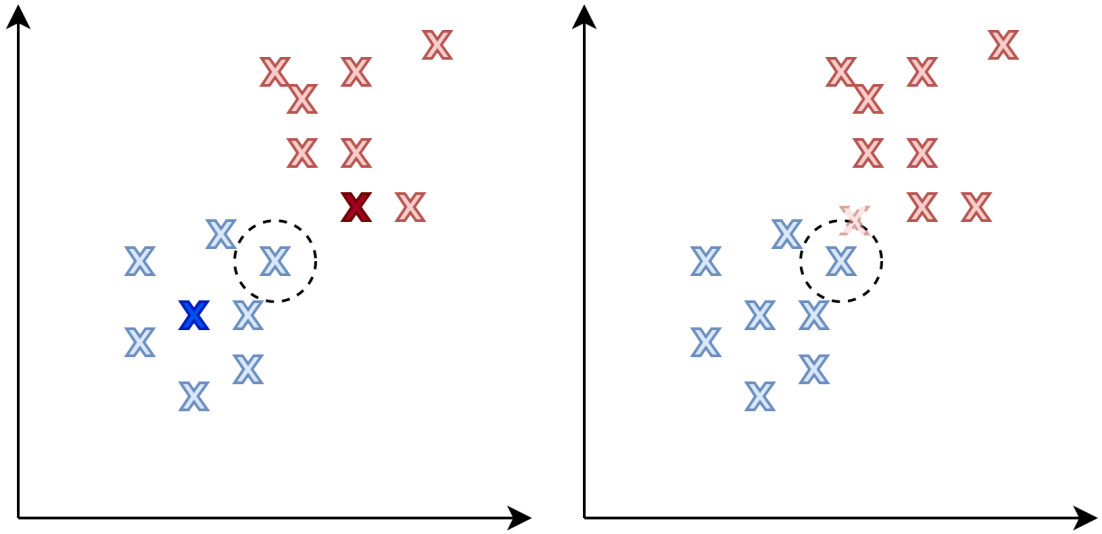
Other model simplification approaches are model-agnostic. For instance, (S. Tan et al., 2018) relies on distillation to train a secondary interpretable model with results from the black-box AI agent. LIME can also be considered as a model-simplification approach: in its process, it first uses a local linear approximation of the model to identify its most notable feature (see Figure 2.7c where the decision border for the linear model are displayed). A variation of LIME, named aLIME for anchor-LIME (Ribeiro, Singh, and Guestrin, 2018) produces a local rule-based decision model approximation based on the same principles as LIME.

Examples, Counterexamples and Counterfactuals

Producing examples or counter-examples for a decision is a technique that is often used by humans when informally asked to explain a decision or a phenomenon (T. Miller, 2018; Hoffman, Mueller, and Klein, 2017). Some XAI methods offer to use this ap-

proache to explanations: instead of focusing on finding causes, relevant parameters or equivalent interpretable models, they identify meaningful examples, counterexamples or counterfactuals that they present to the user, letting him/her understand the logic of the classification based on this information.

Prototypes and criticisms approaches (Kim, Khanna, and Koyejo, 2016) offer to yield, for each class of an AI model, an example illustrating best the class (i.e. the prototype of that class) and a criticism, that is an instance of a different class closest to the point. For a point x classified as belonging to the class c, the method will give as an explanation the point considered most representative of the class c, as well as a point from a different class c' that is the closest to x.



(a) (b)

Figure 2.9: The difference between prototype, criticism and counterfactual. (2.9a) for a given point (circled), we provide a prototype (dark blue) and a criticism (dark red), respectively a representative instance of the selected class and a most-similar-looking instance of another class. (2.9b) for a given point, the counterfactual (light dashed) is a hypothetical point which would have been classified differently while only slightly different from the original.

Counterfactual XAI approaches differs from the exhibition of examples and counterexamples (Verma, J. P. Dickerson, and Hines, 2021). Here, rather than finding a previously seen datapoint that was classified the same (or differently in the case of counterexample), the goal is to find a hypothetical point that has not been observed, that would be classified differently and that is as close as possible from the original point. This notion relates to the already mentioned logical understanding of counterfactuals (see Section. 1.2.2 and (Lewis, 2013)).

Formally, consider a model f an input point x from a manifold $\mathcal{X}$, such that $f(x) = y$. For this point, an counterfactual $x_{counter}$ for a class $y' \neq y$ is defined as:

$$x_{counter} = \text{Arg}\min_{x'} \max_{\lambda} \lambda(y' - f(x')) + D(x, x') \ . \tag{2.1}$$

The first term of the minimized expression accounts for the desire that the classification of x_{conter} to be as close as possible to y', the second term accounts for the minimization of the distance to the original point x (Wachter, Mittelstadt, and Russell, 2017). However, the definition of this distance D is not easy, similar to the issue with defining the notion of "closest" possible worlds in (Lewis, 2013). (Verma, J. Dickerson, and Hines, 2020) accounts for this difficulty by describing a more complex version of Equation 2.1:

$$x_{counter} = \text{Arg}\min_{x'}\max_{\lambda}\lambda(y' - f(x')) + d(x, x') + \gamma(x, x') + l(x', \mathcal{X})\,, \tag{2.2}$$

where d is a standard distance (Euclidean, ...), γ accounts for the sparsity of the difference (how many features differ between x and x') and l for the distance from $x_{counter}$ to the original data manifold $\mathcal{X}$ (even though $x_{counter}$ is not an actual example, it should be plausible). Additional terms may be included, to account for other possible features of the counterfactual, such as feature mutability.

(Verma, J. Dickerson, and Hines, 2020) provides an overview of different possible counterfactual generation approaches. Among them, (Wachter, Mittelstadt, and Russell, 2017) is a model-agnostic method, (Barredo-Arrieta and Del Ser, 2020) focuses on plausible adversarial examples to assess the robustness of Neural Networks, (Rathi, 2019) uses SHAP input to generate counterfactuals.

Counterfactual approaches are slightly different: instead of giving an existing point belonging to a different class, they output the differences that would have made the classifier change its decision regarding the given input point. (Wachter, Mittelstadt, and Russell, 2017) proposes to generate such counterfactual examples to understand the behavior of models in a model-agnostic fashion. (Barredo-Arrieta and Del Ser, 2020) uses counterfactuals to generate plausible adversarial examples in order to assess the robustness of Neural Networks.

2.3 Evaluation of XAI methods

The main motivations for XAI are presented in (Mueller et al., 2019) and span over many fields. This and the fact that, as we have seen in the previous chapter, explanations are complex matter, makes precise evaluation and comparison of XAI methods a hard topic (Adadi and Berrada, 2018). For instance, consider the examples from Figure 2.8: are they good explanations for the classifier's decision? Which level of explanation are they targeting (Maxwell et al., 2020b)? After a short survey, (T. Miller, Howe, and Sonenberg, 2017) finds out that most XAI papers fail to integrate the social science aspect of explanations, while this is considered a major part of the topic (T. Miller, 2018). Similarly, (Adadi and Berrada, 2018) estimates that only around 5% of XAI literature address the issue of evaluation and identifies this as on of the major axes of development in the future of the field. In fact, evaluating explanations is an inherently difficult task, as the objective and the perception can vary depending on the audience (T. Miller, 2018; Leake, 2014; Maxwell et al., 2020b).

To this matter, (Doshi-Velez and Kim, 2017) introduces three different evaluation level to estimate the performance of an XAI method with regards to different aspects,

which we present in descending cost order. First, "application grounded evaluation" requires humans evaluating the interpretability of the model on real world application. This level is regarded as costly (Antunes et al., 2008) as it requires human expert evaluation on often complex end applications, but it arguably provides the best evaluation framework. Second, "human-grounded" evaluation relies on humans judging the performance of the XAI system in simple tasks and toy setups. Third, "functionally-grounded" evaluation uses both automatic metrics and proxy tasks to assess the XAI's performance.

Since "functionally-grounded" evaluation is the cheapest, it is mostly used in XAI literature to compare the performance of different approaches using objective metrics from Machine Learning and Statistics: precision, accuracy, stability (Ribeiro, Singh, and Guestrin, 2016; Schlegel et al., 2019). While not always representative of the human understanding of explanation, these evaluations are useful to provide insight on some approaches: for instance, (Schlegel et al., 2019) shows that LIME is unstable, being sensitive to small perturbation in the data, and therefore cannot be suited for some applications.

Examples for "human-grounded" evaluation frameworks can also be found. Often, human subjects are asked to choose between two or more explanation rationales in specific toy scenarios to evaluate the relative performance of the explaining system. (Nothdurft, Heinroth, and Minker, 2013) finds that explanation dialogues can improve user trust towards a machine. (Larasati, De Liddo, and Motta, 2020) surveys 48 novice people to evaluate a cancer diagnosis posed by a machine expert, depending on the kind of justification provided: thorough, contrastive, descriptive, etc.(Cai, Jongejan, and Holbrook, 2019) tests the impact of visual examples in a simple task: users where asked to draw an object, and a model had to recognize it. The experiment compares "normative" examples, i.e. prototypes of the selected class, and "comparative" examples, i.e. prototypes of the closest match classes. The study finds that in cases of misclassification, the normative explanation provides visible improvement in the user's understanding of the misclassification.

Human-based evaluation of XAI methods requires guidelines and common tools to provide objective comparison. (Gentile, 2021) introduces two metrics to evaluate human perception of generated explanations: i) proxy tasks, i.e. a scenario in which the test subject is asked to reproduce the result of the AI agent; ii) mental models, i.e. the test subject is asked to build and represent a model of the inner workings of the agent's behavior. (Mohseni, Zarei, and Ragan, 2021; Tintarev and Masthoff, 2011) identify additional criteria: computational metrics (as already presented); explanation usefulness and satisfaction; impact on trust and reliance. Similar criteria are proposed by (Sokol and Flach, 2020). (Mohseni, Zarei, and Ragan, 2021) consider that, as evaluation of explanation by humans is the prime goal of XAI, it is necessary to include it to the design process of an XAI system. They propose general guidelines for designing explainable systems to integrate these considerations. Recently, Human-Centered XAI (HCXAI) emerged as a response for these issues, by integrating human users at the center of the design and explanation process of the system (Chaput, Cordier, and Mille, 2021; Ehsan and Riedl, 2020).

2.4 XAI and complex cyber-physical systems

Most XAI works currently do not address the particular case of complex cyber-physical systems, such as smart homes. Instead, they focus on "opening the black box", i.e. explaining the decision of a single AI agent using a given model (Adadi and Berrada, 2018; Arrieta et al., 2020; Rajani and Mooney, 2018). This follows the initial goals of XAI from the 2016 project call (Mueller et al., 2019; DARPA, 2016) which explicitly introduced XAI as a tool to improve the interpretability of black-box model AI agents.

However, as we have seen in Chapter 1, smart homes are complex cyber-physical systems. They are composed of many interacting agents which collectively achieve high-level user-defined goals (Ricquebourg et al., 2006; Mekuria et al., 2019). In smart homes, the difficulty of explanation does not come primarily from the complexity of the AI models used in components: as (Mekuria et al., 2019) observes, most of reasoning smart home systems rely on rule-based or other interpretable system, at the scale of a component. Rather, the difficulty arises form the multiplicity of sensors, actuators and controllers. We illustrate this in Example 2.2: a problematic situation originates from conflicting goals between a CO_2 regulation system and a temperature controller. In this situation, there is a need for system-wide explanations, a task that is not yet commonly addressed in XAI literature.

Example 2.2

In a smart home, a room is equipped with a smart heater and a connected window, a thermometer and a CO_2 sensor. The smart heater has a built-in model of the room's temperature, learned using past data, that it uses to compute the adequate power to use to maintain a given temperature within the room. At the same time, the connected window, using its own model of the CO_2 concentration in the room, opens up, leading the temperature of the room to unexpectedly decrease. Even though the model of the thermometer is understandable, there is a need for another level of explanatory coordination within the smart home to generate the following explanation to the user: "the room is cold because a window opened to avoid high CO_2 concentration".

Autonomous cars and other self-driving vehicles pose a relatively similar problem to explainability, as they are complex CPS (Zablocki et al., 2021; Grigorescu et al., 2020). Furthermore, given the high cost of eventual mistakes and the possible legal concerns regarding the responsibility of the car occupant (Doshi-Velez, Kortz, et al., 2017), they are considered a challenging environment for XAI. Most of the explainability research, though, focuses on the black box model driving the car, i.e. the single AI agent that uses as input the images and data from cameras and sensors and outputs a decision (throttle, braking, direction) (Zablocki et al., 2021). While fulfilling the goal of identifying reasons for an accident, this approach leaves aside issues that can arise in complex systems: failure of one or more components, conflicting information between sensors, etc. Autonomous vehicle, however, differ from smart homes as their configuration does not undergo changes during its lifespan (unless some supervised maintenance where reconfiguration is handled by a human expert).

Integrating explainability to self-adaptive systems remains a topic that has not yet been entirely investigated, as few existing works address self-adaptive specific issues for explainability. (Blumreiter et al., 2019) introduces the MABE-Ex framework, a derivative from the MAPE model that is commonly found in self-adaptive systems (Kephart and Chess, 2003). MABE-Ex, which stands for Model-Analyze-Build-Explain adds an additional layer of an explanatory model on top of an otherwise classic MAPE cycle. They illustrate their approach on an autonomous vehicle, taking into consideration the complex nature of the system. However, they do not provide any implementation architecture. Another topic for explanation in complex CPS, presented by (Welsh et al., 2014), is to consider self-explanation as an additional self-adaptation property that requires specialized architectural features to be enabled.

In this thesis, our goal is to propose an overall method to bring explanation into self-adaptive cyber-physical systems. Since XAI proposes a multitude of possible approaches to explain local agent decisions, our work will focus on proposing a system-wide architecture and algorithms to aggregate the different capabilities of individual components and integrate them into a full-fledged explanatory system.

Chapter 3

An approach to self-explanatory smart homes

Summary

As current XAI research focuses on developing interpretable models, no suitable solution exists to generate explanations at the scale of an entire complex cyber-physical system such as a smart home. The development of an adapted explanatory system faces multiple difficulties, among which the heterogeneity of components, the scope and goal of the explanation, the adaptation and integration of runtime changes in the house environment or in the control system.

To tackle these issues, we draw a parallel between explanations and argumentative or deliberative reasoning. Both processes are based on an initial contrast between an expected state of affairs and the actual observation. Previous work proposed to generate an argumentative dialogue as the trace of a conflict-solving cognitive process. We build our solution based on such a process. This allows to put user interaction at the center of the system.

The starting point of our method is named CAN. It divides argumentative reasoning into three main steps to which it owes its name: *Conflict*, *Abduction* and *Negation*. It is a centralized iterative process that propagates the initial conflict until it eventually reaches a solution.

In this chapter, we first introduce an array of target goals for a smart home explanatory system. We detail the CAN procedure and show how it achieves some of these goals. Then, we present our extension of CAN, named D-CAS, which stands for *Decentralized Conflict-Abduction-Simulation*. In this process, the knowledge of the system is scattered across the different components and all core steps of CAN are performed locally, while a central component coordinates the interactions. This central components is minimal, in the sense that it does not have understanding of the system, but acts as a mere coordinator.

3.1 The goals of an explanatory system for smart homes

As we have highlighted in our literature review, no generic framework has been proposed to address the specific issues of explanations in the context of smart homes.

We propose to design an explanatory system as presented in Figure 3.1. A physical environment is monitored by an existing control system, which is usually constituted of multiple connected devices and controllers. This applies to currently available smart home systems (Mekuria et al., 2019). The user gives high-level objectives to this control system, such as a target temperature, illumination of the room or a schedule. To make the control system *explainable*, we introduce an *explanatory* system whose task is to observe the state of the smart home's environment and control system, to understand it and to generate explanations. Ultimately, the explanatory system presents its reasoning to the user via interactions. This overall concept is an adaptation proposed by (DARPA, 2016), shown in Figure 2.4, to adapt to the general organization of a smart home.

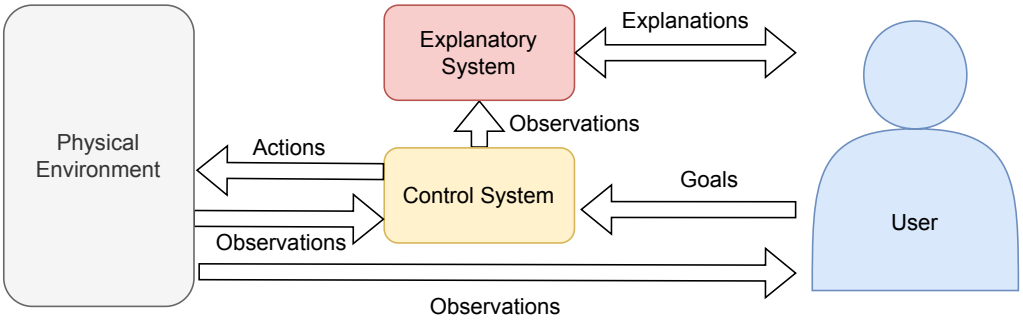


Figure 3.1: The addition of an *explanatory* system to an existing *control system* makes the entire system *explainable*, i.e. capable of generating explanations regarding its behavior.

However, it is still necessary to further define the objectives of the system. The problem of explanation is indeed too general to cover every use case at once. Rather, one needs to make choices regarding the objectives. We base our approach on the guidelines exposed by (Mohseni, Zarei, and Ragan, 2021), integrating the goals of the explanatory system early in the design process.

We have identified six main attributes a satisfactory smart home explanatory system should present. These attributes can be classified into two main categories: *user-wise* attributes denote the characteristics of the explanation rationale that should be presented to the user; *systemic* attributes denote the properties of the explanatory system from an architectural point of view. However this differentiation can be blurry, as some systemic aspects may be apparent in the rationale exposed to the user, and vice-versa. We have identified the following attributes: user-wise *contrast*, *shallowness* and *transparency* of the explanatory rationale; systemic *self-awareness*, *genericity* and *privacy*.

3.1.1 User-wise perception

Contrast Most explanations are contrastive, in the sense that they account for the difference between an expected state of affairs and actual observations (T. Miller, 2018; Lipton, 1990; Pearl and Mackenzie, 2018). We decide to focus on contrastive explanations because most explanations can fall in this category as the comparison is sometimes implicit in a why-question (Lipton, 1990). This limitation helps formalizing the notion of explanation while still covering most use cases. Situations that do not correspond to user expectations are when the need for explanations is most dire in automated systems (N. Li et al., 2020): for instance, in Example 1.1 from the introduction, the user's need for an explanation is triggered by the contrast between the observed cold temperature and her expectations (i.e. the room temperature being regulated by the system). The first criterion for the explanatory system therefore is that it should work in these situations.

Focusing on contrastive explanations is a major topic. Most of the time, user expectations match his/her observations. Situations where expectations contrast with observations are rare. In our opinion a relevant explanatory system must target these rare situations. This can be challenging: given the scarcity of similar situations, data collection is hard, which hinders the performance of statistical approaches. Therefore, the overall explanation process should not rely only on data-based approach, but rather integrate a vast array of possible methods.

Shallowness Explanations and causality are two closely related notions: the former often requires to exhibit causes of the phenomenon to explain to account for it (T. Miller, Howe, and Sonenberg, 2017; Hoffman and Klein, 2017). However, unraveling the causal chain that leads to the explanandum can be a complex process and end up generating an intricate causal graph (Ladkin and Loer, 1998). While such outputs can be correct and particularly useful, notably when targeting expert users for maintenance tasks, they can be perceived as confusing by typical users. For most mundane explanations, the depth of the argumentation is limited by the number of logical steps a human can easily handle: (T. Miller, 2018) estimates this number to be single-digit. This limitation can be posed as a first argument for our goal of a *shallow* explanation: an acceptable explanatory system will not present extensive causal chains, but rather prioritize shallower but more acceptable sequences.

Shallowness, in the context of smart homes, can also have implications regarding the multi-scale aspects of the system. In multi-scale architectures, the user provides high-level goals which cascade through the different levels of the control systems to end up defining low-level targets for the physical actuators in various devices (Diaconescu, Di Felice, and Mellodge, 2019). In this situation, the explanatory logic should follow the same organization: considering a given high-level explanandum, the process should not directly consider a low-level explanans; rather, the different arguments of the rationale process should progressively dig into the system, eventually stopping when reaching a limit adapted to the user's profile. The resulting explanatory process can be qualified as shallow, in the sense that there is a limit in the depth gap between consecutive arguments in the output.

Transparency One of the main objectives of explanatory systems is to build *trust* between the machine and its user (Arrieta et al., 2020). (Wing, 2020) identifies that transparency is a key to enabling trust. This observation is particularly true in smart homes, as they are a part of the user's mundane intimacy. Therefore, we pose transparency of the reasoning process as an objective for smart-homes explanatory systems: each step of the process should be exposed in the output and unambiguously identify which component of the control system is responsible for the situation. This vision follows the definition of explanation as cognitive processes rather than a fixed given list of causes and inference rules (T. Miller, 2018; Horne, Muradoglu, and Cimpian, 2019).

In case of error from the explanatory system, transparency allows to identify and track down which part of the reasoning failed. This argument is already presented by (Rudin, 2019) to advocate for broader use of transparent AI models in critical applications. In this context, transparency enables targeted feedback from either the user or the system. In a later example, in Chapter 5, we show how this process can be used to mitigate the impact of erroneous inferences from the explanatory system.

3.1.2 Systemic properties

Self-awareness Smart homes are examples of self-adaptive systems (Kramer and Magee, 2009): adaptation is required to reconfigure the system, to perform updates, to keep track of objective changes or environment factors. As the control system exposed in Figure 3.1 presents self-adaptation properties, so should an explanatory system. We propose to extend this scope by designing the explanatory system as to present self-awareness capability. This property encompasses self-adaptation and denotes that a system is able to react to changes by learning and modeling its behavior and environment (Kounev et al., 2017).

A typical illustration of self-awareness for the explanatory system is a *plug-and-explain* property, similar to the existing "plug-and-play" which exists in smart homes and greatly favors their adoption by a wide audience (B. Miller et al., 2001). Here, the addition of a new device into the control system is automatically handled by both the control system and the explanatory system, as to integrate this new device into future explanations. Notably, the addition of the new device can be identified as a possible cause for future failures in the system. Another example is self-optimization: improving performance based on perception is facilitated when process transparency enables the identification of the component responsible for each part of the reasoning.

Genericity The configuration of each smart home differs by the multitude of device providers, the uniqueness of the monitored physical environment, the profile of the users and their objectives (Mekuria et al., 2019). Similar to what exists in XAI research, where the general focus for explainable methods is towards model-agnostic solutions (Adadi and Berrada, 2018), smart home explanatory systems should be agnostic about the architecture of the control system. Also, they should provide a standardized output regardless of the properties of the devices present in the control system and its organization. To enable this, the explanatory system relies on standard components which self-specialize to the underlying smart home control system, rather than to design-specific components.

Architectural genericity can also be transposed into the explanatory reasoning: regardless of the situation, the architecture of the system or the explanandum, the principles and algorithm of the explanatory system should remain unchanged. This process genericity improves the understandability of the overall system, as it reduces the number of different processes and algorithms to expose to the user. In this sense, genericity can be understood as implying a certain form of minimalism in both the organization of the explanatory system and its reasoning process.

Privacy Privacy concerns have been fueling the development of XAI in recent years (Marcus, 2020; DARPA, 2016) and remain a major concern among smart home users (Marikyan, Papagiannidis, and Alamanos, 2019).The explanatory engine must not pose additional threat to its user's privacy. Thus, our focus is to create an explanatory engine that is able to work locally, without referring to distant computing or data. This relates to the philosophy of Edge Computing (Shi et al., 2016), which we will later implement in our demonstrator (see Chapter 5).

3.2 A generative model for argumentative dialog

3.2.1 Explanation and argumentative dialog

The distinction between explanation and argumentation is not always well-defined. (Bex and Walton, 2016) finds that they differ by their goal rather than their intrinsic process: an explanation aims to make a person understand a situation, notably by exposing *causes*, hence using abductive inference and counterfactual reasoning; while an argumentation tries to convince a person of a proposition by exposing different *arguments*. In fact, both processes can be triggered by a why-question, showing that the ambiguity is also present in natural language (Bex and Walton, 2016; Lipton, 1990). While a general consensus exists to differentiate explanations from arguments (McKeon, 2013), we propose to rely on the structural proximity between the two forms of reasoning. This allows to use existing argumentative reasoning frameworks as a basis to develop an explanatory system satisfying our different criteria.

A most notable argumentative reasoning framework is the Argumentation Theory formalism developed after Dung's original work (Dung, 1995). This theory considers an Argumentation Framework as a set A over which is defined a binary "attack" relation R. $R(a, b)$ means that argument a attacks argument b, and therefore that b cannot be accepted by an agent considering a. It allows to define the notion of an "acceptable argument" a with regards to a set of arguments $S \subset A$ as an argument for which every attacker is itself attacked by an argument of S: $\forall b \in A, R(b, a) \implies \exists s \in S, R(s, b)$. This definition coincides with the intuitive notion of acceptability, i.e. that an agent accepts an argument external to its original beliefs if it can attack any objection to this new argument.

By offering a mathematical formalism, Dung's theory allows to model complex socio-logical interactions, such as legal cases (Bench-Capon, 2020) or medical case study (Longo, Kane, and Hederman, 2012). Despite these achievements, this formalism

is too rigid to be considered in mundane argumentative dialogues: as (Dessalles, 2016) points out, Dung's formalism is based on the existence of a predefined graph-like representation of the different arguments and attack relations. This condition can be hard to satisfy in an explanatory system for smart homes, especially with regards to our goal of contrastive explanations: as we've pointed out, these explanations focus on unusual situations, for which the existence of an established knowledge graph is hypothetical. The situation of smart home systems is closer to informal argumentation dialogues, where humans are able to form arguments and evaluate the situation at runtime (Ghadakpour, 2003).

3.2.2 The Conflict-Abduction-Negation process

In typical mundane conversations, an essential notion linking consecutive sentences is *relevance*: each proposition is relevant with regards to the previous sentence. To account for such interaction, (Dessalles, 2016) introduces a novel cognitive process named CAN, for *Conflict – Abduction – Negation*. This process is centered around the notion that the motivation for an argumentation is the solution of an initial conflict, each new argument either helping towards its resolution or bringing forth an underlying conflict. The basic principle of CAN is illustrated in Figure 3.2: from the consideration of a conflict, an agent can either i) use abductive inference to propagate it to its causes; ii) negate the currently examined conflict and thus consider an alternate world which allows the discovery of new causes and consequences; iii) find a solution to the examined conflict, either by performing an action effectively ending the conflict, revising its perception of the world or, as a last resort, giving up on the conflict.

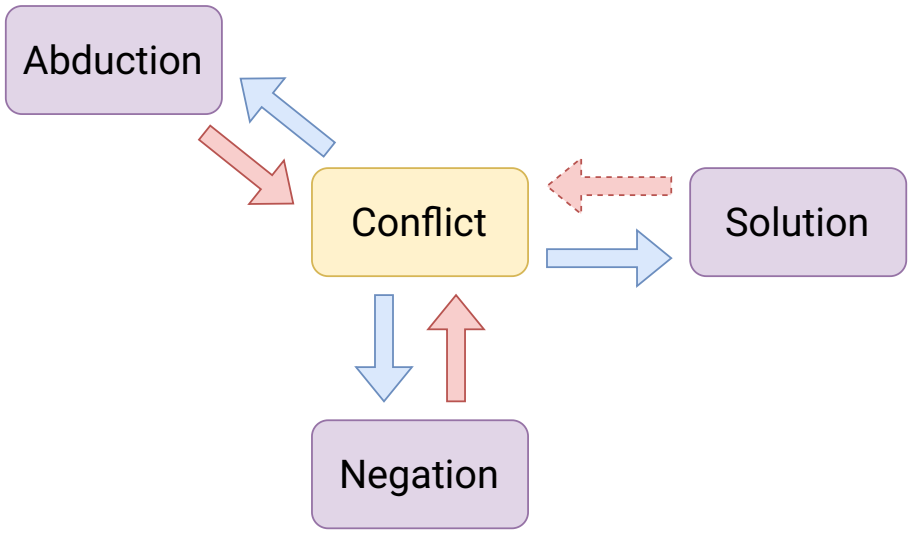


Figure 3.2: The CAN process. When a conflict is detected, three possible follow-ups exist (blue arrows): perform an abductive reasoning, consider the negation or solve the conflict (either by performing an action, revising knowledge or giving up). The two first possibilities propagate to a new conflict (red arrows), while solving the conflict may also raise another conflict somewhere else in the system (red dashed arrow).

Model CAN models the world with a triple $(\mathcal{P}, \mathcal{R}, \nu)$ whose elements are the following.

- $\mathcal{P}$ corresponds to the knowledge of the agent regarding the state of the world. Formally, $\mathcal{P}$ is a set of propositional formulae (instantiated predicates), each one being evaluated to a given Boolean value which describes the corresponding state of the world. For instance, to describe that the room is cold, CAN will use the proposition $\texttt{cold}(\texttt{room}) = True$. The set of existing predicates corresponds to the *vocabulary* of the agent, while the different objects are mapped to objects of the world. As CAN's goal is to approximate a cognitive process, the original contribution (Dessalles, 2016) does not further specify the structure of $\mathcal{P}$: the nature of the objects used in formulae is unspecified, which is the origin of some difficulties when seeking to implement CAN in the realistic setup (see below, Section 3.3.1).

- $\mathcal{R}$ describes the inference knowledge of the CAN agent. In the original paper, the proposed implementation consists of various predicative formulae (Dessalles, 2016). For instance, to express how the user understands the causal relation between the window being open and the room being cold, CAN uses the rule $\texttt{open}(\texttt{X}) \wedge \texttt{in}(\texttt{X}, \texttt{Y}) \Rightarrow \texttt{cold}(\texttt{Y})$. Note, however, that CAN does not require the inference knowledge $\mathcal{R}$ to be of any specific nature. As CAN dissociates the abductive reasoning from the rest of the conflict-solving process, it only requires that the agent is able to perform abductive reasoning, without any assumption regarding the nature of the inference knowledge $\mathcal{R}$.

- ν is called the *necessity*. It is a numerical score corresponding to the agent's *opinion* regarding the propositions. Formally, it is a mapping from Boolean propositions to real numbers $\nu : \mathcal{P} \mapsto \mathbb{R}$. For any given proposition $p \in \mathcal{P}$, $\nu(p) > 0$ indicates that the agent believes or wishes p to be true, while a negative necessity means that the agent believes or wishes p to be false. For instance, if the agent does not want the room to be cold, this preference can be encoded as a necessity $\nu(\lfloor \wr\updownarrow\lceil(\nabla\wr\Updownarrow)) = -30$. The absolute value of the necessity, called its *intensity*, indicates the strength of the opinion. As the necessity accounts for an opinion, it is an odd mapping: $\forall p \in \mathcal{P}, \nu(\neg p) = -\nu(p)$. In the original CAN description, necessities were given as inputs from the user, indicating his/her state of mind, and as settings for the system's preferences.

From this simple model of the agent's knowledge, CAN then breaks down the conflict-solving process into three separate main steps:

Conflict In CAN formalism, a conflict is defined as a discrepancy between an agent's prior opinion and the observed state of affairs. Formally, this occurs when the value of a proposition $p \in \mathcal{P}$ contradicts its associated necessity $\nu(p)$: either p is true and $\nu(p) < 0$ or p is false while $\nu(p) > 0$. As the CAN model is based on the solution of such conflicts to motivate an argumentative reasoning, their detection is the first major step of the process. A conflict is therefore modeled by a couple (P, N), where $P = p$ is the conflicting proposition and $N = \nu(p)$ is the corresponding necessity. The *intensity* of the conflict is defined as the absolute value $|N|$ of the necessity. This notion is essential

in CAN and serves multiple purposes. Notably, as no structured memory (e.g. LIFO or FIFO piles) is used in CAN, intensity is used to determine which conflict should be examined: CAN uses a greedy approach, examining the most intense conflict first.

Example 3.1

As a person comes into her office, she observes that the temperature is lower than expected. In CAN model, this observation is formalized as a true Boolean proposition $p = \texttt{cold}(\texttt{room})$. However, since the person expected the temperature control system of the room to work properly, the person's surprise is modeled by supposing that the necessity is negative: $\nu(p) = -30$. This results in a conflict $(\texttt{cold}(\texttt{room}), -30)$. The CAN process can start the generation of an argumentative reasoning based on this conflict.

Abduction Abductive inference is the process of considering the hypothetical cause of an observed phenomenon. Therefore, it is central to the study of causality (Pearl and Mackenzie, 2018), argumentation and explanation (Hoffman and Klein, 2017; Douven, 2021). Moreover, it is commonly found in everyday reasoning (Douven, 2021). CAN integrates abduction into its general scheme and identifies it as one of the three main steps of argumentative reasoning. In CAN, abduction is used to propagate a conflict, from the original phenomenon to its hypothetical causes. Formally, when a conflict (p, N) is identified and considered, abductive reasoning can be performed by the agent, using its knowledge of rules $\mathcal{R}$ and current state $\mathcal{P}$. If a proposition c is thought to be a cause for the conflicting proposition p, the conflict can be propagated to c: the necessity $\nu(c)$ is set to either $|N|$ or $-|N|$, depending on which sign is necessary to create a conflict. The resulting conflict $(c, \pm|N|)$ is subsequently considered to be the new conflict to solve for the next iteration of the problem. However, this propagation is conditioned by the previous necessity associated with the cause c: if $|\nu(c)| > |N|$, the intensity of the incoming conflict is not enough to change the agent's opinion concerning c. Example 3.2 illustrates how CAN uses abduction to propagate the initial problem of the room's temperature towards a hypothetical cause.

Example 3.2

(continuation of Example 3.1) In her understanding of the physics of the room, the person possesses a rule $\texttt{open}(\texttt{window}) \implies \texttt{cold}(\texttt{room})$ expressing her empirical knowledge. Based on this rule, she infers the proposition $c = \texttt{open}(\texttt{window})$. As she had no prior opinion regarding it, she can propagate the conflict onto it, setting $\nu(c) = -30$. The new conflict to examine is now $(\texttt{open}(\texttt{window}), -30)$.

Similar to conflicts, CAN uses a greedy approach for abduction: in case several parallel hypotheses can be inferred from the original conflict p, CAN will process the first one, for instance c_0. The original CAN model does not specify on which criterion this order can be established. Since this lead does not necessarily succeed, other leads will have to be considered. This occurs effortlessly in the CAN process: since the original conflict (p, N) still stands, it is examined anew. During this second examination, the hypothesis c_0 can

no longer serve as a valid hypothesis: its necessity had previously been set to *at least* $|N|$ during the first abduction and can no longer satisfy the propagation condition. Therefore, the process automatically considers the second possible cause proposed by abduction, c_1, and propagates the conflict onto it. Importantly, the propagation criterion, which states that the conflict can be propagated only if the incoming intensity is not higher than the existing one, induces that CAN cannot loop indefinitely (see Section 3.3.4 for consideration regarding the termination of the process.)

Negation In CAN model, a conflict is described as (P, N), where N stands for the necessity of the proposition P. One might notice that when one negates a conflict, the resulting couple $\texttt{NEG}(P, N) = (\neg P, -N = \nu(\neg P))$ also corresponds to a conflict. This property allows CAN to propagate the conflict to its negation. As illustrated in Example 3.3, instead of wondering why the window is open, with an intensity of 30, one might instead wonder what prevents the window from closing, with the same intensity of 30.

This kind of reasoning is closely related to counterfactual thinking, which we have already identified as being a pilar for the notion of causality (see Section 1.2.2, (Hoffman and Klein, 2017)). By considering the negation of a conflict, CAN considers the closest state of the world where this conflict is negated, which corresponds to the understanding of counterfactual (Lewis, 2013).

Example 3.3

(continuation of Example 3.2) Faced with the open window, to understand why it is open, the person considers the negation of the conflict: $(\neg open(Window), 30)$, which corresponds to wondering why is the window not closed. She finds out that this would make the room uncomfortable, as the CO_2 concentration would become too high. In CAN formalism, this preference is encoded by $\nu(\texttt{high\_co}_2(\texttt{room}) = -20)$. Therefore, considering the negation raises the conflict $(\texttt{high\_co}_2(\texttt{room}), -20)$.

In CAN, negation allows to consider potential causes and consequences of conflicts that would not have been otherwise integrated into the process. In Example 3.3, negation allows to propagate the conflict towards the CO_2 concentration within the room, and hence to expose the following rationale: the room was cold because a window was open by the system to ventilate it and reduce the CO_2 concentration.

Solution CAN identifies three possibilities for an agent to end a conflict (p, N). First, the agent can effectively solve it by performing an *action* that changes the state of the world and therefore the value of the conflicting proposition p. Second, the agent can *revise* its knowledge, i.e. considering changing the value of p without acting on the world. For instance, the person of Example 3.2 can observe that all windows are closed, and thus change the value of $\texttt{open}(\texttt{window})$, thus terminating the conflict affecting this proposition. Third, the agent can *give up* on the conflict, by switching the sign of the necessity of p. By setting $\nu(p) = -N$, this ends the conflict as the necessity becomes adequate with the observed value of p.

In all cases, CAN stores the new necessity of the proposition, so its intensity is used to memorize that this proposition was considered with a certain strength. This is notably used to avoid endlessly considering the same cause, as illustrated by Example 3.4.

Example 3.4

(Continuation of Example 3.2) Following the abductive inference, the examined conflict is $(\texttt{open}(\texttt{window}), -30)$. However, after checking the different windows of the room, the person finds that none is open, and thus revise her knowledge with $\texttt{open}(\texttt{window}) = false$. This revision terminates the conflict affecting the windows. As this conflict no longer exists, CAN examines the remaining one, i.e. $(\texttt{cold}(\texttt{room}), -30)$. Now, the window state can no longer be proposed by abductive inference: the propagation condition is not met, as the necessity of $\texttt{open}(\texttt{window})$ is 30.

Internal state ($\mathcal{P}$ and ν)	Examined conflict	Description
$\mathcal{P} = \{\texttt{cold}(\texttt{room})\}$ $\nu(\texttt{cold}(\texttt{room})) = -30$	$(\texttt{cold}(\texttt{room}), -30)$	Initial State
$\mathcal{P} = \{\texttt{cold}(\texttt{room}), \texttt{open}(\texttt{window})\}$ $\nu(\texttt{cold}(\texttt{room})) = -30$ $\nu(\texttt{open}(\texttt{window})) = -30$	$(\texttt{open}(\texttt{window}), -30)$	Abduction propagation
$\mathcal{P} = \{\texttt{cold}(\texttt{room}), \neg\texttt{open}(\texttt{window})\}$ $\nu(\texttt{cold}(\texttt{room})) = -30$ $\nu(\texttt{open}(\texttt{window})) = -30$	$(\texttt{cold}(\texttt{room}), -30)$	Knowledge revision
$\mathcal{P} = \{\texttt{cold}(\texttt{room}), \neg\texttt{open}(\texttt{window}),$ $\neg\texttt{on}(\texttt{heater})\}$ $\nu(\texttt{cold}(\texttt{room})) = -30$ $\nu(\texttt{open}(\texttt{window})) = -30$ $\nu(\texttt{on}(\texttt{heater})) = 30$	$(\texttt{on}(\texttt{heater}), 30)$	Abduction Propagation
$\mathcal{P} = \{\neg\texttt{cold}(\texttt{room}), \neg\texttt{open}(\texttt{window}),$ $\texttt{on}(\texttt{heater})\}$ $\nu(\texttt{cold}(\texttt{room})) = -30$ $\nu(\texttt{open}(\texttt{window})) = -30$ $\nu(\texttt{on}(\texttt{heater})) = 30$	None	Action

Table 3.1: Successive internal state of the agent during the Examples 3.1, 3.2 and 3.4 and their continuation. The transcript from this process to natural language can be subsequently generated. For instance: "Why is it cold in the room? Using abduction, I infer the cause may be an open window. However, I do not find this to be true. Another cause may be that a heater is turned off. This is true. Were the heater on, the room would no longer be cold, which solves the initial conflict."

3.2.3 Observations on CAN

Overall, the functioning of CAN is represented in Table 3.1. It displays the internal state $\mathcal{P}, \nu$ of the agent during the process. This highlights the separation CAN operates be-

tween the logical reasoning of the agent and the argumentative rationale. Logical reasoning is comprised in abductive inference and during the propagation of the consequences of a new consideration (e.g. in Example 3.3, inferring the high CO_2 concentration from the closed window). On the other hand, the general argumentative rationale is entirely directed by CAN and is formalized by the succession of conflict examinations and chosen steps.

This distinction allows CAN to be a minimalist process, in the sense that it uses a small number of steps that can handle a variety of different situations. It is also minimalist in the information required to keep track of the ongoing process. Rather than storing all calls and previous interactions in a pile-like structure, CAN only relies on its set-structured knowledge representation of Boolean propositions and their associated necessities, using the propagation rule to avoid endlessly considering the same propositions. This minimalist approach complies with the observation from (Ghadakpour, 2003; Gärdenfors, 2004; Dessalles, 2015) that the human mind does not rely on predefined rules or complex reasoning process; rather, it is capable of formalizing concepts and logical rules at runtime. By clearly separating the inference process from the construction of the general reasoning, CAN respects this cognitive ability.

The notion of necessity is the second aspect of CAN's minimalism. By unifying the beliefs and desires of the agent, its introduction lowers the possibilities of the model, when compared to other approaches which keep the two notions distinct. For instance, the Belief-Desires-Intention (BDI) model enables different types of dialogue and reasoning (**todo**). However, since the scope of CAN is limited to conflict-solving situations, unifying beliefs and desires was not found to be detrimental (Dessalles, 2016). Rather, its minimalism is beneficial: by using a unique metrics, it allows to define a simple propagation value: a conflict (P, N) can be propagated onto another proposition Q if and only if $|\nu(Q)| < |N|$. In this understanding, necessity can be associated to the mutability of propositions, CAN being allowed to consider possible causes if the intensity of the request is high enough.

CAN can be directly implemented to generate explanations rather than conflict-solving dialogues. Indeed, when considering the rationale exposed in Table 3.1, one can understand the sequence of conflicts as the user discovers a causal chain and explores hypotheses. Similarities can be observed with some approaches of scientific reasoning, notably the Hypothetico-Deductive method (Popper, 1963). As such, when considering the different criteria we posed for an explanatory system in Section 3.1, we observe that CAN meet the user-wise criteria we defined in Section 3.1. The contrastive aspect of the explanation is apparent as CAN, by nature, focuses on solving a conflict: in Example 3.1, the corresponding why-question would be *"Why is it cold in the room when I expected the opposite?"*. The shallowness of the reasoning is not guaranteed by CAN: rather, it depends on the abduction process. However, the "one-step-at-a-time" approach by CAN allows for a shallow reasoning in the sense that a small gap in the argument's complexity can be taken at a time, if the abduction process is compatible with this approach. This approach is present in the example detailed in Table 3.1 Also, the introduction of the give-up mechanism as part of the process and the conditioning of the propagation on the intensity of the conflict allows to stop the reasoning once a given complexity is reached, for instance once the problem is not worth the effort from the user to verify the situation.

Third, transparency of the reasoning is guaranteed by the small number of different steps and the exposition of the trace of the process as an explanation.

3.3 Adapting CAN to the smart home: the D-CAS algorithm

3.3.1 Difficulties of adapting CAN to smart homes

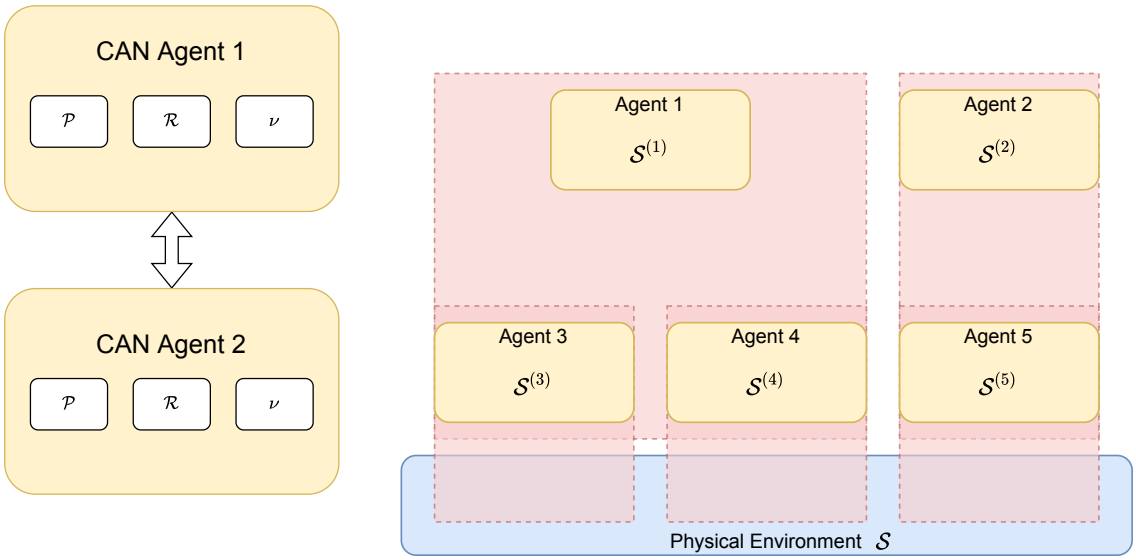
While CAN provides a means to conform to the user-wide goals of an explanatory system for smart homes, the question of systemic properties is out of the scope of the original process. In this section, we identify the current difficulties CAN would struggle with in smart-home applications.

Localization and nature of knowledge

CAN is a centralized process, in the sense of knowledge localization: the same agent is responsible for possessing knowledge of the world and generating the rationale using the CAN process. Knowledge here is modeled by Boolean propositions, necessities and inference rules. In case CAN models a discussion between different agents, knowledge is fully shared between them: each agent will expose its state representation $\mathcal{P}$ and necessities ν to others. This organization is not possible in typical smart home systems: given the diversity and heterogeneity of connected devices (Jain and Murugesan, 2021; Ricquebourg et al., 2006; Mekuria et al., 2019), they often rely on decentralized or hybrid organizations (Krupitzer et al., 2015; Weyns, Schmerl, et al., 2013), where high-level controllers have a partial aggregated knowledge and low-level controllers have precise but specific knowledge acquired from measures of the shared physical environment. For instance, a controller responsible for the state of a window has no access to the temperature measures from the neighboring room.

Some complex systems adopt a centralized knowledge approach, building and maintaining a knowledge base covering the entirety of the system and its environment. (Alirezaie et al., 2017) builds an ontology at the scale of a smart home for device integration, but it is limited to simple cases. (Sukor et al., 2018) proposes an ontology framework for activity detection. However, maintaining a knowledge base spanning the entire system is hard, as the number of fields and interactions increases with the number and types of devices. In addition, as many self-adaptive systems integrate some degree of decentralization, a unique knowledge base would not integrate well in these architectures and potentially hinder their flexibility and adaptation capabilities. Figure 3.3 illustrates the difference in knowledge organization between the original CAN approach and that of a typical smart home system.

The difficulty of maintaining a knowledge base for smart home systems does not only come from the number of components. It also comes from the complexity of the physical environment. In CAN, knowledge is represented by Boolean predicates $\mathcal{P}$. The original paper presented a possible implementation where the inference knowledge $\mathcal{R}$ is composed of predicative formulae. As logical rules are considered easily understandable



(a) (b)

Figure 3.3: Difference of knowledge distribution. In CAN (a), agents have are conscious of the state of the entire world, and can share their opinion via dialogue. On the other hand, in a smart home system (b), devices' knowledge (in red) is limited to their observations and their eventual connected components: the set of all variables describing the system's state $\mathcal{S}$ is not entirely covered by any individual knowledge $\mathcal{S}^{(k)} \subsetneq \mathcal{S}$. Knowledge sharing exists through context awareness (e.g. agent 1 is aware of some variables of agents 3 and 4, hence $S^{(4)} \cap \mathcal{S}^{(3)} \neq \emptyset$).

by humans (Augasta and Kathirvalavakumar, 2012; Guidotti, Monreale, Ruggieri, Pedreschi, et al., 2018), this guarantees that the output and reasoning of CAN is adapted to model realistic dialogues. However, in smart homes, the state of the physical world is complex: it can be modeled using a set of real-valued variables $x_i(t)$ which are measured by devices. The interactions between such variables are more complex than logical rules. As a consequence, while in CAN the rule-based model $\mathcal{R}$ is used for both forward inference (deduction, from causes to consequences) and backwards (abduction, from consequences to causes), the same is not possible in real-life cyber-physical systems, where maintaining a centralized rule-based knowledge spanning over multiple components hinders the architecture of the existing control system.

Negation

Negation plays an important role in the CAN model, but also in reasoning in general. As noted by (Pearl and Mackenzie, 2018; T. Miller, 2018; Menzies and Price, 1993), negation, counterfactual and hypothetical thinking are prime components of the human capacity of abstraction and reasoning. However, while this operation is made easy in the CAN formalism, where it simply consists of negating a conflict, proposition and necessity, and propagates the consequences and causes using the inference knowledge, this is not the case in Cyber-Physical systems. Here, the system is in direct observation of a physical

environment which can not be modified at will and acts as "ground truth". Furthermore, as already stated, perfect inference knowledge is hard to acquire, if not impossible, for real-life Cyber-Physical Systems, especially so when considering the multitude of components and interactions.

Flexibility

As a cognitive process designed to model interactions and argumentative dialogue, CAN offers a great flexibility. The prime example is that the process does not specify an order for executing the different possible steps at each conflict examination. Rather, in the examples provided in (Dessalles, 2016), this flexibility of order plays a role in the variety of tasks handled by the process without adding overweight to the overall complexity of the rationale. Considering an implementation to real-world cyber-physical systems, such questions of decision order must be solved to guarantee the process' reliability. However, the resulting system must not lose the initial capability of CAN to adapt to new situations, which is particularly essential considering the context of explanations for unusual situations.

Another challenge arising with the adaptation of CAN to a realistic setup is the composition of the vocabulary of predicates and their objects to form propositions. While the approach used by CAN allows for the flexibility that is required when modeling a cognitive process, its lack of formalism poses challenges to the implementation. What can be identified as the objects of the predicates? How does one create a predicate, and decide of its truth value to generate the propositions?

Finally, the original CAN proposal does not address the challenge of evaluating necessities: they are intended to reflect the agent's opinion, inducing a ranking between the different propositions (does the agent prioritize the temperature of the room or its ventilation?). This choice is motivated by the requirement of modeling various behaviors with the unique necessity score. However, it poses the problem of the implementation: in a realistic setup, how can the system determine which necessity scores should be associated to various propositions?

3.3.2 D-CAS: Decentralized Conflict-Abduction-Simulation

Compared to CAN, the knowledge model of used to represent the smart home explanatory system differs: here, a physical environment $\mathcal{E}$ is measured and can be described using a set $\mathcal{S}$ of time series variables $x_i(t)$. The evolution of these variables is governed by a set of physical equations $\mathcal{F}$. Each component k of the smart home system is monitoring a defined subset $\mathcal{S}^{(k)} \subset \mathcal{S}$ and has no access to variables external to this domain (see Figure 3.3b). The exact physical laws of F are *a priori* unknown, but approximations can be possessed by components (this corresponds to components modeling all or part of the system). Note that this notation does not differentiate between variables internal to the control system (e.g. a device's voltage or internal state, a switch position) and environment variables (e.g. the temperature of the room, time of day). At any given time, a set of Boolean propositions $\mathcal{P}$ describes the state of the world. Here, a boolean proposition $P \in \mathcal{P}$ is an abstraction that can be associated with a value, either true or false, to rep-

resent the user's perception of the world. A proposition P can therefore be understood as a mapping $P : \mathcal{S} \mapsto \{0, 1\}$ and can be represented by a word or a conjunction of words. For instance, the proposition $\texttt{cold}(\texttt{room}) \wedge \texttt{room\_type}(\texttt{room}, \texttt{living} - \texttt{room})$ indicates that the user feels cold in the living-room.

Considering this situation, and the previous observations on CAN, we designed a variation of CAN which specifically targets smart homes and similar complex Cyber-Physical Systems. This algorithm is named D-CAS, which stands for *Decentralized Conflict – Abduction – Simulation*. Two major differences with CAN are explicit in this name: i) this new approach considers the decentralization of world knowledge and inference processes; ii) Simulation replaces Negation. This latter change is necessary in smart homes, where the state of the world cannot be easily mentally negated. Instead, simulation refines the vision of Negation in the original CAN process: instead of simply "negating" an observed state of affairs $\mathcal{S}_t$, D-CAS runs a simulation to study the evolution of the system given an alternative initial situation. I.E., to negate a given proposition p at time t, it considers an alternate state of affairs $\mathcal{S}'_t$ such that $p(\mathcal{S}'_t) = 0$, then uses known available approximations in components to evaluate the consequences of a state changes $\mathcal{S}'_{>t}$. This implementation is closely related to the notion of "thought experiments" (Brown and Fehige, 2019) which have been identified by (Pearl and Mackenzie, 2018) as prime examples of counterfactual reasoning.

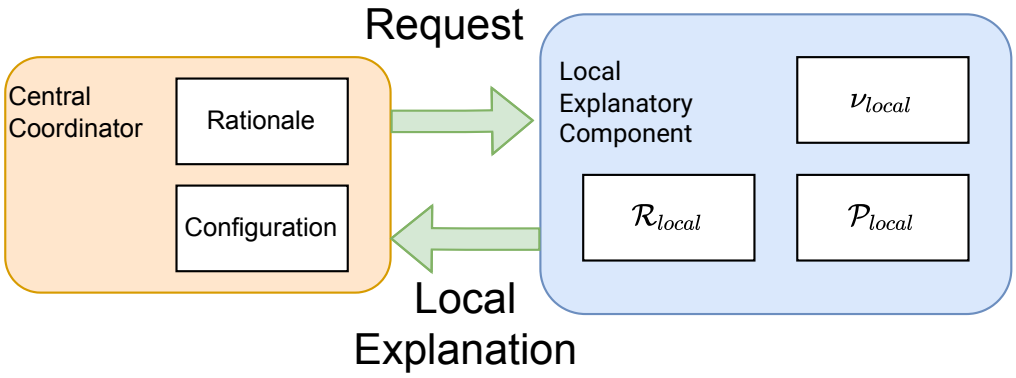


Figure 3.4: Knowledge distribution in D-CAS. The central Coordinator's knowledge is kept minimal, integrating only the current rationale and configuration information that allows it to address the different components. Local Components, on the other hand, contain local knowledge following the original CAN formalism: Boolean propositions $\mathcal{P}$, inference knowledge $\mathcal{R}$ and necessities ν.

The principle of D-CAS is as follows. Various *Local Explanatory Components* (LECs) have local knowledge of the world, both in terms of opinions (necessities ν), inference knowledge ($\mathcal{R}_{local}$) and world knowledge ($\mathcal{P}_{local}$). This knowledge corresponds to the partial states $\mathcal{S}^{(k)}$ accessible to one or several components k of the smart home control system[1]. In addition, a central coordinator has knowledge of the explanatory rationale and the configuration of the system (see Figure 3.4). With this setup, the coordinator

[1]In the architecture described later in Chapter 4, exactly one LEC is attached to each component from the smart home.

executes the main loop of the D-CAS algorithm (see Algorithm 1) which iteratively identifies the LEC responsible for a conflict (i.e. the component whose knowledge domain contains the conflict) and requests it to provide a local handling of this conflict. As the central coordinator merely routes requests and delegates conflict handling to LECs, we call it *Spotlight*, as its role can be illustrated by putting the relevant component into the spotlight.

This local handling can be understood as processing one CAN step using local knowledge. The LEC is able to identify the conflict within its local knowledge and model it. Then, depending on the situation, it can use its knowledge to i)identify a hypothetical cause for the conflict; ii) find an action that can solve the conflict; iii) revise its knowledge comprehension of the physical world to end the conflict. The response from the LEC indicates which one of these options it chose, along with the result of the option (i.e. which hypothesis is designated by abduction, which action is identified, which knowledge is revised).

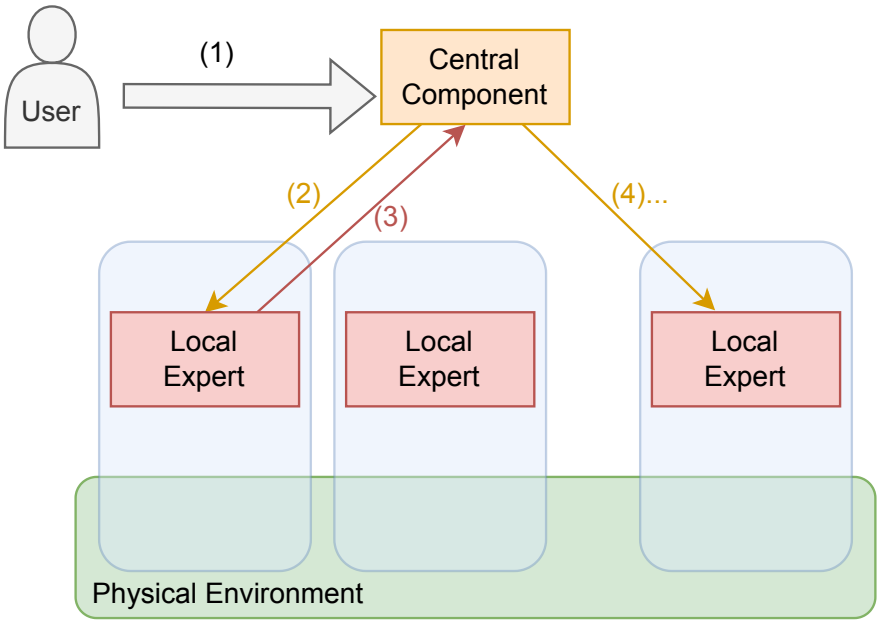


Figure 3.5: Principle of the D-CAS algorithm. When a user requests an explanation (1), a central component named Spotlight identifies which local expert is responsible for the problem. Then, the request is transferred to this expert (2) which, depending on its knowledge on the situation can either propagate it using abduction, ask for a simulation or give-up the conflict (3). Then, the Spotlight considers the eventual other conflicts affecting other components (4), repeating the process.

Algorithm 1 shows the main loop of D-CAS, which runs on the Spotlight central coordinator. This main loop is triggered by a request from the user to explain a situation. This request is converted into a conflict-like object, which contains a proposition and an associated conflicting necessity, following CAN's (P, N) conflict format. Then, the Spotlight locates which one of the LECs is responsible for the conflict's proposition P (line 2). If no such component exists, D-CAS stops the search for this conflict and

backtracks. If a component is found, D-CAS delegates the request to it and retrieves the response. Depending of the response, the Spotlight will either consider the new conflict and responsible designated by abduction, or give up the conflict and backtrack or run a simulation and wait for its potential outcomes.

Algorithm 1: The D-CAS algorithm: the Spotlight successively considers conflicts and routes the request to the most relevant component to locally process it.

```
Input: A request req from the user
Result: A conflict-solving process whose trace can be exposed as an explanation
Data: Pointers to the LECs in the system
C a set of examined conflicts, G a set of considered give-ups
(P, N) ← analyzeRequest(req);
responsible ← locate(P);
while responsible ≠ self do
    if responsible = null then
        Backtrack() ;
    end
    answer = responsible.investigate((P,N)) ;
    switch answer do
        case ABDUCTION do
            (P,N) ← Answer.Hypothesis ;
            responsible ← locate(P) ;
        end
        case GIVE UP do
            Backtrack();
        end
        case ACTION do
            simulator.run(Answer.Action) ;
            Conflict ← waitForProblems() ;
        end
    end
end
```

Algorithm 1 relies on different sub-methods to delegate the different complex operations required in the general rationale. This delegation follows the base principle of CAN to differentiate the explanatory rationale from the logical processes involved within the reasoning. The different methods used in D-CAS are displayed in Table 3.2 with the component hosting them, their signature and eventual comments. D-CAS's use of delegation and identification of relevant components follows the principles from autonomic computing where components expose their abilities and can be called to solve the task they are specialized in (Philippe Lalanda, McCann, and Diaconescu, 2013; Kephart and Chess, 2003). More detail regarding the enabling of self-adaptation features for the explanatory system will be given in the next chapter which focuses on its architectural aspects.

Name	Signature	Actor	Additional Comment
analyzeRequest	any $\rightarrow (P, N)$	Spotlight	
locate	$P \rightarrow$ Pointer	Spotlight	Adds the conflict to C. If the conflict provided has be given-up, or none is provided, uses the highest-intensity conflict in C. If C is empty, returns a pointer to the spotlight.
backtrack	None	Spotlight	Adds the currently examined conflict to G and ends the current loop.
investigate	$(P, N) \rightarrow$ Response	Local Expert	
run	$(P, N) \rightarrow$ None	Simulator Module/ Local Expert	

Table 3.2: An overview of the different sub-methods of D-CAS

AnalyzeRequest The first step of the process is an interface with the user. As the user speaks Natural Language, a mandatory step is to process the expressed sentence into a *Conflict* object, which contains a proposition P and an associated necessity, similar to the original CAN notation (Dessalles, 2016). This method is considered out of the scope of the present thesis: as many approaches exist to handle Natural Language Processing tasks, we estimate that this step can take advantage of existing work. Notably, voice interactions are already a commonly found feature in smart home systems (Kowalski et al., 2019; Jivani, Malvankar, and Shankarmani, 2018), which is especially useful for systems designed to assist elderly people. Therefore, we do not consider this front-end of the explanatory system as part of our contribution, and will left unspecified the exact method **analyzeRequest()** and its signature in Table 3.2.

Locate In order to keep Spotlight's knowledge minimal, no information regarding the LEC's local knowledge is stored in the coordinator. Rather, the Spotlight contains a **locate()** method, which returns a pointer to the LEC that is most relevant to process the given proposition. In addition, **locate** adds the examined conflict to the set C of examined conflicts. This method can be understood as the Spotlight broadcasting to all known LECs a $know$ request containing information about the questioned proposition P and analyzing their response to identify which LECs are competent. In their response, LECs return an estimate of their knowledge of the proposition, which allows the Spotlight to rank them. In case more than one LEC are competent, the Spotlight will process only the first one, eventually exploring the other components if need be (i.e. if the exploration of the first component fails to solve the conflict). The **locate()** method therefore allows the Spotlight to implement the CAN-like reasoning without having access to the local knowledge $\mathcal{P}_{local}$ of LECs across the system, thus enabling the genericity property of the system since the Spotlight is system-agnostic.

Investigate Delegation of specialized tasks is the key element of D-CAS, extending the differentiation principle introduced by CAN between the explanatory rationale and logical operations. The **investigate()** method is the implementation of this principle in D-CAS. With it, the Spotlight calls previously identified relevant LEC to process the investigated conflict. This investigation by the LEC consists of examining local knowledge and performing locally the CAN steps of conflict detection, abduction and negation. As the exact order of these steps is not specified by CAN, we settled on using the order exposed in Algorithm 2: the LEC first checks the state of the conflict using its local knowledge (line 1), then checks if the intensity of the conflict is necessary to trigger an action (line 5). Then, it tries to infer an hypothetical cause using abduction (line 9) and if negating the conflict is possible (line 15). In this outline of the algorithm, local knowledge in functional: it is embedded within the different methods in bold font which the LEC executes to understand its environment.

Algorithm 2: A possible implementation of the **investigate()** method. In this implementation, priority is given to potential actions, then abduction and negation.

Input: A conflict (P, N)
Output: A `Response` object

```
1  fail ← checkConflict((P, N));
2  if fail then
3  |  return (GIVE UP, N) ;
4  end
5  action ← findAction(P, N));
6  if action ≠ null then
7  |  return (ACTION, action) ;
8  end
9  hypothesis ← abduction((P, N)) ;
10 if hypothesis ≠ null then
11 |  return (ABDUCTION, hypothesis);
12 end
13 negation ← findNegation((P, N));
14 if negation ≠ null then
15 |  return (ACTION, negation)
16 end
17 return (GIVE UP, N) ;
```

The **investigate()** method returns a `Response` object, which encapsulates a type indicator among GIVE UP, ACTION and ABDUCTION, and the eventual new conflict to consider. The ACTION response type covers both the situation when an actual action is possible and when merely considering negation. This is due to our previous observation on the state of negation in a cyber-physical system: as physical observations can be interpreted as a ground-truth, negation of a current state is interpreted as a simulation of an alternative state of affairs. A difference between the two exists at the local scale. The action is considered practically feasible by the LEC while no such

requirement exists for the negation; but from the point of view of the Spotlight, both of these decisions result in the same consequence: simulating an alternative state of the world and its consequences. During the **checkConflict()** method, the LEC has the opportunity to revise its knowledge: if the incoming intensity is higher than the confidence of the LEC regarding one of its beliefs, it can revise the latter, which will result in effectively identifying the conflict. This particular ability allows will be further detailed in the following chapters.

Run D-CAS relies on the ability to perform simulations to observe possible outcomes of alternate scenarios and propagate conflicts. Again, the chosen approach is to delegate the operation from the Spotlight to specialized components via the **run()** method. Prior to using this method, the Spotlight identifies which component is most capable of performing a simulation by calling a method similar to the **locate()** method: however here the call can also reach dedicated simulators, given that some simulation operations can require heavy computations that LEC's cannot perform by themselves. Upon starting the simulation, the Spotlight notifies all LEC's in the system, so that they can consider the outcomes of the simulation. Thus, simulation results can raise new conflicts in case one resulting state was associated with a negative necessity in one LEC. In this case, the hook **waitForProblems()** is called, and a new conflict is considered in a new iteration of D-CAS by the Spotlight.

In our model, all boolean propositions correspond to an action consisting of setting the physical environment in a state where this proposition is not verified anymore. Some of these actions correspond to do-able operations, e.g. switching on a button, shutting down a device. Others are not directly feasible, e.g. setting outdoor's temperature to a given value. However, since we consider simulation as the explanatory system's realization of mental model, this does not constitute a problem for the system. In addition, it is possible for LEC's to still make a distinction between possible and impossible actions, by flagging them and making this flag explicit in D-CAS' output.

Backtrack Knowledge revision is mandatory when using abductive inference, as abduction is not a sound method (Magnani, 2011): there is no guarantee that the abductive inference is correct. This "trial-and-error" methodology has been integrated into the general scientific method by (Popper, 1963), where knowledge revision and refutation of previous theories is a key element. Similarly, it is often found in explanatory rationale (T. Miller, 2018). The original CAN process integrates this revision as the possibility to revise predicates' necessities or values during the process. As we've seen, the context of smart homes makes such revision harder, as the physical environment cannot be directly manipulated. However, a first revision is enabled at the level of the local component, which can revise its interpretation of the world to modify, if need be, the meaning and truth value of Boolean propositions.

At the Spotlight level, revision is left to the **backtrack()** method. This method adds the currently considered conflict into the set of given-ups conflicts G in the Spotlight: thus, conflicts that are present in this set can no longer be examined by the algorithm if they have been dismissed. Furthermore, if the Spotlight was previously running a simulation, **backtrack()** stops the simulation and broadcasts a message to the LECs to

stop considering simulation outputs. **Backtrack()** can thus be understood as a branch-cutting method, effectively setting the reasoning back to the root of the branch, while using the give-up set G as a memory of the discarded option, as not to explore the same branch again. In Chapter 5, we will describe an implementation of D-CAS based on this tree representation.

3.3.3 An example of D-CAS

We illustrate the functioning of D-CAS by mentally processing the same situation as presented in Examples 3.1, 3.2 and 3.3. The resulting rationale is presented in Table 3.3, detailing the C and G sets for each iteration of the algorithm, along with the requests and their answers. The process follows the same unraveling as the one exposed with CAN in Table 3.1. We observe the following steps. First, the LEC responsible for the temperature makes the hypothesis that an open window causes the cold temperature. However, asking the window LEC about it reveals that the window was in fact closed, which results in the hypothesis being abandoned and added to G.

C	G	`responsible`	Request	Response
$\{(\texttt{cold(room)}, -30)\}$	$\emptyset$	LEC_Temperature	$(\texttt{cold(room)}, -30)$	Abduction: $(\texttt{open(window)}, -30)$
$\{(\texttt{cold(room)}, -30), (open(window), -30)\}$	$\emptyset$	LEC_Window	$(\texttt{open(window)}, -30)$	Give up: $(\texttt{open(window)}, -30)$
$\{(\texttt{cold(room)}, -30), (open(window), -30)\}$	$\{(\texttt{open(window)}, -30)\}$	LEC_Temperature	$(\texttt{cold(room)}, -30)$	Abduction: $(\texttt{off(heater)}, -30)$
$\{(\texttt{cold(room)}, -30), (open(window), -30), (\texttt{off(heater)}, -30)\}$	$\{(\texttt{open(window)}, -30)\}$	LEC_Heater	$(\texttt{off(heater)}, -30)$	Action: $(\texttt{on(heater)}, 30)$
$\{(\texttt{cold(room)}, -30), (\texttt{open(window)}, -30), (\texttt{off(heater)}, -30)\}$	$\{(\texttt{open(window)}, -30)\}$	LEC_heater	$(\texttt{off(heater)}, -30)$	Give up: $(\texttt{off(heater)}, -30)$
$\{(\texttt{cold(room)}, -30), (open(window), -30), (\texttt{off(heater)}, -30)\}$	$\{(\texttt{open(window)}, -30), (\texttt{off(heater)}, 30)\}$	LEC_temperature	$(\texttt{cold(room)}, -30)$	Give up: $(\texttt{cold(room)}, -30)$
$\{(\texttt{cold(room)}, -30), (open(window), -30), (\texttt{off(heater)}, -30)\}$	$\{(\texttt{open(window)}, -30), (\texttt{off(heater)}, 30), (\texttt{cold(room)}, -30)\}$	Spotlight	None	None

Table 3.3: Application of the D-CAS algorithm to the continuation of Examples 3.1, 3.2, 3.3, following the same course as Table 3.1

Then, the Temperature LEC is once again interrogated and proposes that the heater being turned off can be the cause of the cold. The investigation then goes to the heater, which confirms the hypothesis and proposes the action of turning it back on, which can be simulated (or run, depending on the capability of the system). After some time, the investigation resumes and since the heater is now turned on, the conflict $(\texttt{off(heater)}, -30)$ is no longer relevant, hence discarded. Then, since the simulation showed that the temperature increased, the conflict $(\texttt{cold(room)}, -30)$ is in turn discarded. This concludes the rationale, as the set of given ups covers the set of considered conflicts.

3.3.4 Analysis of the algorithm

The D-CAS algorithm was designed to be the driving mechanism of a house-wide explanatory system for smart homes. As we've identified before, CAN meets our user-wise requirements, by providing a transparent, contrast-focused and shallow process, in the sense that we've defined in Section 3.1. D-CAS builds upon this basis by keeping the main concepts of conflict detection, solution and propagation of CAN. As such, the observations made for CAN remain true for D-CAS regarding the compliance to user-wise explanatory goals. However, the adaptation of D-CAS to the context of smart homes brings other questions: are there cases where D-CAS can endlessly loop and not terminate ? How does it scale up ? Also, we study the question of dealing with situations where a conjunction of causes is identified by abduction.

Termination

To evaluate the termination of D-CAS, we make the assumption that all invoked submethods terminate. Given that we do not specify implementation for these methods for now, we need to rely on such hypothesis to continue our reasoning[2]. We also make the assumption that the knowledge of the system is finite, i.e. it can be entirely expressed using a finite set $\mathcal{P}$ of propositions scattered across the different components. If both of these conditions are met, D-CAS is guaranteed to terminate when given as input a request carrying a conflict (P, N).

The sketch of the proof of this termination is as follows. Consider the set $\mathcal{P} = \bigcup \mathcal{P}_i$ of all the propositions describing the conflict. Consider M the highest preference intensity set for any component in the system: $M = \max_{\mathcal{P}}(|\nu(P)|)$. This maximum is well-defined since the set $\mathcal{P}$ is finite. We consider the state of all possible "internal states" of D-CAS, which can be described as the set of examined conflicts C and the set of given-ups G. An internal state is terminal if all the conflicts in C are covered by G, i.e. $\forall (P, N) \in C, (P, N') \in G, |N'| \geq |N|$, as the propagation of conflict can no longer occur once such a state is reached. Also, following our preliminary assumption of a finite set of propositions $\mathcal{P}$, only a finite number of internal states are possible. The existence of at least one terminal state is also guaranteed, since $(C = any, G = \{(P, M) | \forall P \in \mathcal{P}\})$ is a terminal state (all propositions have been given-up with the maximum intensity).

We observe that, for each iteration of the algorithm, the effect on the internal state is the following: i) if no responsible for the conflict is identified, or the target LEC abandons the conflict, the process adds the currently examined conflict (P, N) to G; ii) if abduction is used, then a new conflict (P_{abd}, N_{abd}) is added to C; iii) if an action or negation is found, potential new conflicts (P_{new}, N_{new}) can be added to C. In case the action raised no other conflicts, the two possibilities are as follow: either the action succeeded in solving the conflict, in which case the next iteration will add it to G as it is no longer found by the LEC; or the action did not succeed, in which case the next iteration will also add the conflict to G as no suitable solution was found. Therefore, for

[2]This assumption is not very restrictive: for real-world application, we can consider fixing a time limit on delegate methods, thus ensuring their termination by return a default value.

each iteration, D-CAS cannot explore the same state twice as a new element is appended either to C or G. This behavior of D-CAS is illustrated in Figure 3.6.

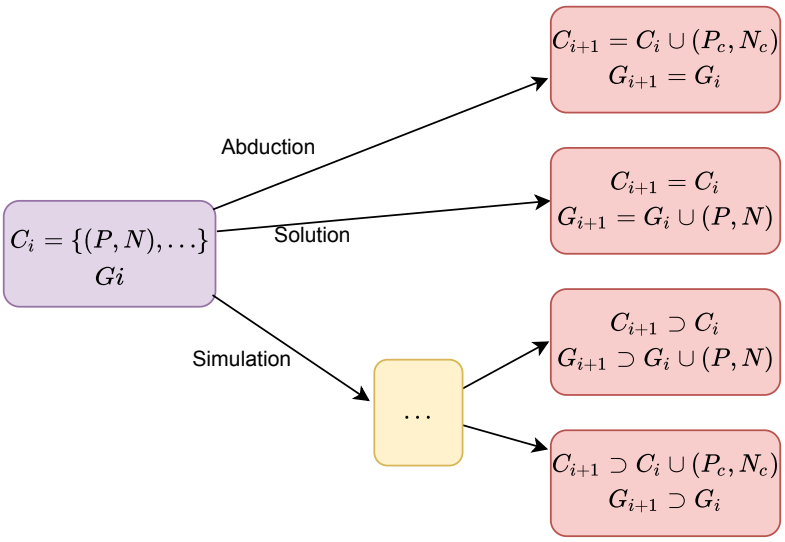


Figure 3.6: Evolution of the internal D-CAS state: with the available mechanisms, no loop can exist in the process, as all operations result by adding an element to either C or G. In the abduction result, P_C and N_C denote the proposed cause proposition and intensity, respectively.

As the algorithm cannot loop, it explores a finite set of possible states, among which exists at least one terminal state (consisting in giving up everything). D-CAS is guaranteed to stop in a finite number of iterations. With our first hypothesis of termination of all delegate methods, this guarantees that D-CAS reaches a terminal state. However, this termination does not guarantee that D-CAS finds an optimal solution. In fact, this optimization is hard to define, given the subjectivity of the user's perception in the definition of optimal explanation (Nothdurft, Heinroth, and Minker, 2013). In our proof-of-concept demonstrator results, in Chapter 5, we will further analyze the output of D-CAS and how it can be ground to explanatory discussions with the user.

Scaling and adaptation

D-CAS does not require the Spotlight to actively maintain system-wide knowledge base. As it requires only pointers to the LECs contained in the system, memory scaling is linear for the Spotlight. As the system grows in complexity, the different LECs need to be more complex, eventually needing better models to take interaction into consideration. However, given that smart homes have a specific organization, local components can integrate this knowledge in order to simplify their model. E.g., in big buildings, rooms can be made aware only of their neighbors to model their temperature evolution. This hierarchical organization is often mentioned in existing smart homes research (Lippi, Mariani, and Zambonelli, 2021; Mekuria et al., 2019).

Following the principle of genericity identified in Section 3.1, the Spotlight keeps no knowledge regarding the state of the physical environment: all specialization is delegated

to the diverse components present in the system. D-CAS' generic approach is illustrated in an example we provided in a communication (E. Houzé, Dessalles, et al., 2021): we studied the possibility of using D-CAS[3] in problematic scenarios where communication between different experts was required to gain knowledge of a complex causal chain of events. The first example is the case of a self-driving car, where accountability can be discussed between the driver, the manufacturer or local authorities; the second example tackled a fictive crime committed under the influence of hate speech and fake news on social media, where knowledge is scattered across the social media company, the author of the fake news and the criminal. In both examples, D-CAS can be used to model possible interactions and help formalizing the necessary interactions between experts.

D-CAS continues the principle of the CAN procedure by keeping communications minimal: different outcomes can be unified by the same message. This is particularly true for the *give-up* operation. From a reasoning standpoint, an unresponsive component and a component that does not find a possibility to continue the reasoning are equally treated as give-ups. This allows for only a handful of required methods which have been described here, allowing to handle a diversity of situations. Also, the genericity of the high-level concepts can enable privacy, by limiting the knowledge of component variables to the sole expert in this component: e.g. the Spotlight has access to the knowledge that there is a problem in a given component, while the exact nature of this problem remains known to this component only. This characteristic, alongside local reasoning, enables the privacy goal we aim to.

We proposed self-awareness as a defining characteristic for an truly explainable system. By using D-CAS, the explanatory system observes the underlying control system and integrates this self-observation knowledge into its exposed reasoning. The question of self-optimization (Krupitzer et al., 2015) is also permitted by D-CAS: the step-by-step approach allows to identify when a part of the reasoning is erroneous. For instance, a bad abductive reasoning is contradicted by a subsequent simulation. Using this insight, the explanatory system can alert components of their mistakes, which can trigger low-level adaptation from components to mitigate their errors.

Other self-adaptation properties are enabled by D-CAS: supposing that all components have self-observation capabilities (which we will describe in the next Chapter), D-CAS allows an adaptive reasoning. If a component observes a local change and proposes it as a possible causal hypothesis via abduction, D-CAS will consider this hypothesis, hence integrating the change into the generated explanation. Similarly, if a component is added into the system at runtime, it can directly acknowledge itself to the Spotlight and subsequently respond when D-CAS investigates knowledge within its expertise range, hence enabling the desired "plug & explain" feature.

3.3.5 Handling multiple causes

Given the complex nature of smart homes, it is *a priori* common that a single phenomenon originates from a conjunction of causes. In this chapter, we have defined CAN as

[3]In this paper, we introduced D-CAS as "D-CAN", using Negation instead of Simulation. This is because these examples were meant to be mental considerations rather than applications to CPS, and therefore did not require to explicitly invoke simulation in negation.

operating over "proposition" which are abstract objects that can be assigned a Boolean value to transcribe a state of the world. In this aspect, it is possible that such a proposition P is in fact a conjunction of several propositions $P = P_1 \wedge P_2 \wedge \dots$, i.e. P is true if and only if the world is in a state $\mathcal{S}$ for which propositions $P_1, P_2, \dots$, are true. The way CAN is designed, a proposition has to be entirely handled by a single component.

However, it is easy to find a situation where this is not the case. Consider, for instance, the situation presented in Example 3.1: the abduction might very well conclude that it is cold indoor because of the proposition $\texttt{cold}(\texttt{outdoor}) \wedge \texttt{open}(\texttt{window})$, proposition that is likely to be handled by two different components of the house. How would D-CAS operate when confronted to such a situation?

A first approach would be to consider both causes by interrogating each component. However, this approach brings several downsides. First, it violates the principle of D-CAS that explanations are sequential processes. This principle facilitates the transparency of the reasoning as it corresponds to the actual process operated by humans when formulating an explanation (T. Miller, 2018). Second, it brings technical difficulties with regards to simulating counterfactuals and pursuing the reasoning: as the formulation of the counterfactual to several propositions covered by different components is harder than identifying an alternate state which would make a single proposition false. Third, it paves the way towards possible breaches to our shallowness goal: the integration of context. In fact, each conjunction can be almost endlessly augmented by including the context. In the previous example, why not consider $\texttt{cold}(\texttt{outdoor}) \wedge \texttt{open}(\texttt{window}) \wedge \texttt{not}(\texttt{fire}) \wedge \texttt{off}(\texttt{heater}) \wedge \texttt{not}(\texttt{sunny}) \wedge \dots$? Where should one draw the line ?

For all these reasons, we deem it better to keep D-CAS in the minimal form presented here, and follow the greedy approach introduced by CAN when considering multiple causes. First, the abductive reasoning favors the hypothesis that is most *mutable*, i.e. that is the easiest to be considered (or simulated). Mutability is measured with intensity, a high intensity denoting that the the proposition can hardly be modified (and therefore requires an important incoming conflict to be considered). In our previous example of the open window, outdoor temperature is less mutable than the window, as one can simply close the latter. Hence, the abductive reasoning will favor this option. In addition, each proposition is handled by the component most relevant to answer it. If other components are available for this, they are at first discarded. They can be interrogated again through a specific request from the user. Such situation will be presented in the implemented demonstrator in Chapter 5

Chapter 4

Architecture Description

Summary

The D-CAS algorithm allows an explanatory system to generate a shallow, contrast-focused and transparent rationale to answer explanation requests from the user. Its knowledge decentralization is designed to comply to the constraints of application to smart-home systems. However, D-CAS alone is not sufficient to fulfill all the goals set for a smart home explanatory system. Doing so requires to consider an adequate component organization that is based on the principles of genericity, self-awareness and privacy protection while allowing D-CAS to operate.

The Local Explanatory Component (LEC) is the basic element required by D-CAS. It must possess local knowledge of the autonomic control system of the house, observe it and map its measures to the propositions used by D-CAS. As the underlying smart home system configuration is a priori unknown, we design LECs so that each LEC is associated to a unique Smart Home Component (SHC), i.e. a controller or a device from the underlying smart home autonomic system. LECs handle local adaptation: as the SHCs can be updated or modified at runtime, the organization of the paired LEC observe these changes and integrate them into the explanatory reasoning. Also, the heterogeneity of possible devices and controllers is handled at the local level in order to enable seamless system-wise integration. For these reasons, we design LECs as generic frameworks where the D-CAS functions, namely conflict identification, abductive inference and counterfactual simulation are handled by modules that can be modified, updated, added or deleted at runtime. This architecture of LECs is currently under review for a patent.

The LECs are then integrated into the explanatory system, by communicating with the Spotlight, which in addition to its coordination role in D-CAS also serves as a naming and directory service for the explanatory system, keeping updated information of LECs present in the system. Formalized communications between explanatory components are also defined.

4.1 General Organization

The D-CAS algorithm presented in the previous chapter relies on knowledge decentralization and central coordination. A generic process hosted in a central component generates the explanatory rationale by delegating requests to various specialized components (see Figure 3.5). These latter posses local knowledge of the state of the physical environment and its evolution, which they use to answer the delegated requests. In addition, high-power components may specialize in simulating complex interactions in the house. The general architecture of the explanatory system must therefore follow this general organization and include the different components. Its two main objectives are: i) to preserve the features and capabilities of the existing smart home control system, i.e. self-organization, self-healing, self-protection, plug-and-play features should not be hindered by the addition of the explanatory system; ii) to enable the realization of the systemic goals defined for the explanatory system, namely self-adaptation, privacy protection and genericity.

The organization of the smart home control system is a priori unknown. In current systems, different architectures exist (Frey, 2013): i) centralized (Banerjee et al., 2018), e.g. a central unit controlling diverse connected heaters; ii) hybrid (IBM Co., 2005; Weyns, Schmerl, et al., 2013), e.g. a central hub connected to a temperature control hub and a home security hub, each managing its own heaters and alarms; iii) fully decentralized, e.g. several connected heaters using the home network to communicate goals and measures in order to coordinate. While the smart home system organization may vary, and rely on different communication technologies (Han and Lim, 2010; Wenbo, Quanyu, and Zhenwei, 2015), the explanatory system aims to be generic, i.e. organization- and implementation-agnostic. In this regard, we only define two requirements for the autonomic system: i) it is composed of one or several *components* (a generic term including both devices and controllers) which should expose some or all of their characteristics and measures to external scrutiny[1]; (ii) it must include some kind of naming and directory service that lists the different components present in the system and their characteristics. In short, the smart home control system is considered to be a complex system, with undetermined organization and presenting an accessible registry and interfaceable components. Existing smart home systems meet these requirements (Ricquebourg et al., 2006; Marikyan, Papagiannidis, and Alamanos, 2019; Kafle et al., 2017), and similar principles are followed in smart home simulations (Philippe Lalanda and Hamon, 2020).

Following the logic exposed in Chapter 3, the explanatory system is added on top of this existing smart home system. Figure 4.1 illustrates the resulting layered organization. The smart home control system is depicted here with a layer of controllers and a layer of devices to emphasize the plurality of roles in this system. To preserve its features, the smart home system is *not* aware of its observation by the explanatory system: therefore, the addition of the latter does not impede on the capabilities of the former. The LECs constitute the lower level of the explanatory system, interfacing directly with the Smart Home Components and their exposed variables. By Smart Home Components (SHCs),

[1] For security reasons, this scrutiny can be limited to a list of selected authorized agents, in our case the relevant LECs.

we designate their roles either devices or controllers from the original smart home system, conforming to our objective of being agnostic of the organization of the smart home control system. On top of this layer, the Spotlight acts as the naming and directory service for the explanatory system. This layered organization also follows the abstraction of the observations: the state of the physical environment is measured using variables, which are then translated into Boolean propositions with which D-CAS operates. The latter are a simple proxy for natural language words that are used by the user. Hierarchy of abstractions is a common tool in Software Engineering to achieve genericity of the upper layers with regards to the lower layers, which is the goal here. In addition, similar architecture can be found in goal-oriented architectures for autonomic systems, where high-level goals are progressively translated into lower-level objectives (Diaconescu, Di Felice, and Mellodge, 2019).

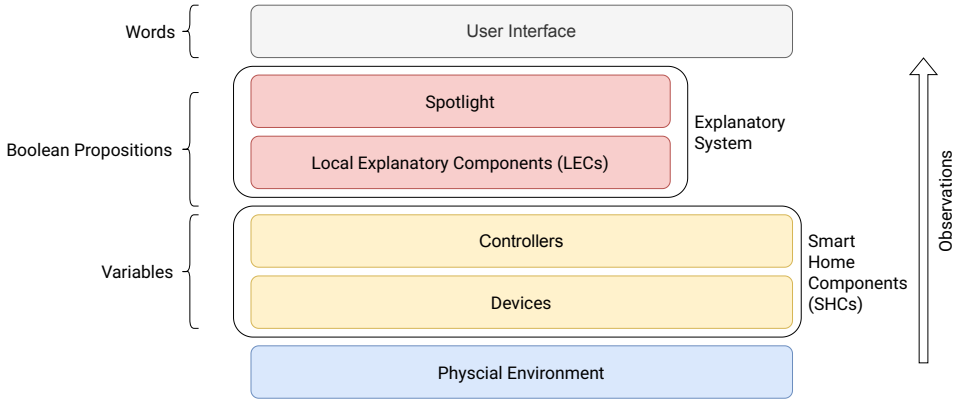


Figure 4.1: The Explanatory System consists of two additional layers to the existing smart home control system. The smart home system is unaware of the explanatory system on top of it, and, as such, entirely independent.

The roles of the Spotlight and LECs are defined by the D-CAS algorithm: LECs possess local knowledge that they use to identify which Boolean propositions correspond to their observations of the control system, to propose causal hypotheses and understand the consequences of actions. Their number is not specified by D-CAS. On the other hand, the Spotlight is defined as a unique and central component, that has *no* knowledge of the system's state, but keeps an updated list of pointers to LECs. These requirements can be formalized by defining the components' blueprints as programming interfaces, as shown in Figure 4.2.

Our proposed solution for LEC organization is to rely on a one-to-one pairing: a LEC is assigned to each Smart Home Component, be it a controller or a device. The associated LEC is responsible for understanding the SHC: for instance the LEC attached to a window controller is able to detect conflicts regarding the window, know its state, potential motor or sensor failures, infer possible causes for its opening/closing. In addition, the window LEC knows that its attached component is able to open or close the window. The one-to-one pairing between LECs and SHCs offers several benefits:

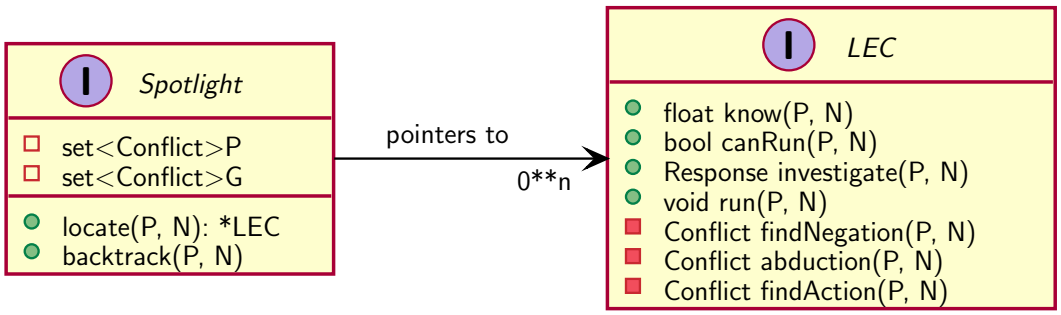


Figure 4.2: The interfaces required by D-CAS for the Spotlight and the LEC.

Knowledge Distibution: It delegates the responsibility of knowledge distribution to the existing autonomic control system of the house: since the competence domain of each LEC corresponds to the one of its associated SHC, its capabilities are limited to those of the SHC. For instance, a "smart" window blind that is aware of the presence of lights in the room will expose this knowledge to its LEC, eventually enabling new abductive inference. As we will later discuss, this feature is interesting with regards to adaptation. Indeed, if we consider that the control system is able to self-organize to best adapt to the layout and goals of the house (Lippi, Mariani, and Zambonelli, 2021; Diaconescu, Mata, and Bellman, 2018), the explanatory system can benefit from this capacity and integrate this self-adaptation capability.

Reasoning Transparency: Directly associating each Smart Home Component to a LEC improves the transparency of the D-CAS algorithm: in the explanation process, each call to a LEC corresponds to a call to the knowledge of a specific SHC. This makes the reasoning more traceable, as it exhibits which components of the house control system are involved.

Edge Comuting: A one-to-one pairing facilitates integration of the *edge computing* paradigm: operations are performed as close as possible to where data originate (i.e. devices and controllers) (Shi et al., 2016). Edge computing typically aims at improving security, reliability and adaptation of cyber-physical systems, as opposed to cloud computing. As such, this design choice is consistent with the goals of privacy defined in Chapter 3.

Self-Adaptation: Pairing SHCs to LECs adds a layer of computing which can handle the observation and integration of local adaptation. This prevents the rest of the explanatory system to reconfigure itself following a local change. For instance, if a configuration change affects the window (e.g. a software update, a motor replacement), handling the subsequent adaptation is the role of the LEC attached to it. The rest of the system will not have to revise its knowledge.

Figure 4.3 depicts an example of the organization of the explanatory system within a smart home. While the smart home organization is hierarchical, with high-level controllers monitoring several lower-level devices, the structure of the explanatory system

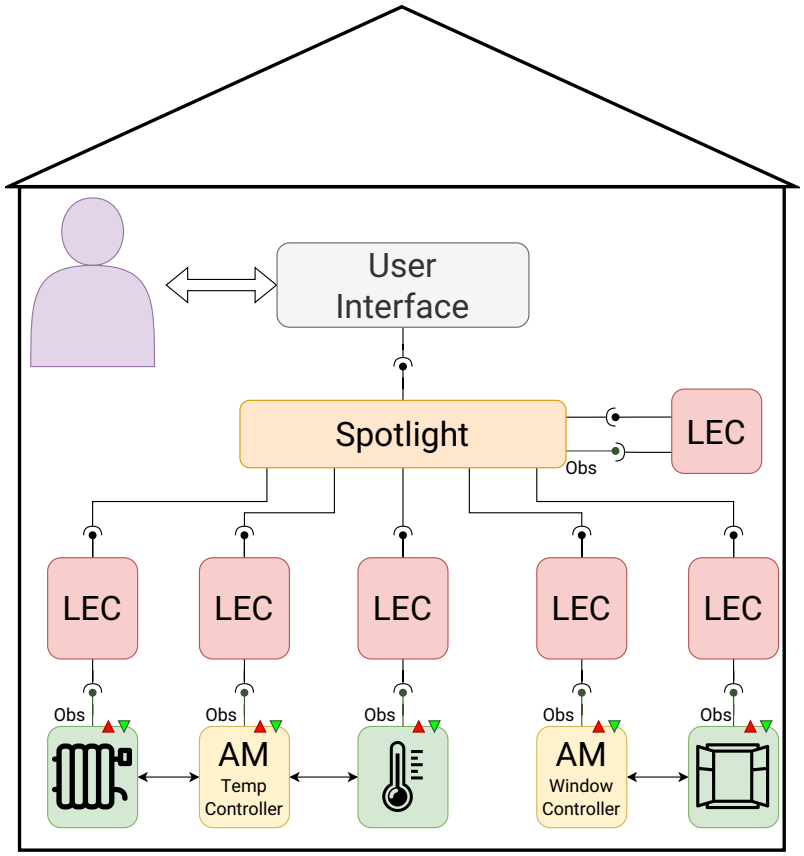


Figure 4.3: A smart home is controlled by an existing autonomic system made of devices (green) and managers (yellow). All of these components are observable, exposing an interface "Obs". Local Expert Components (LECs), in red, read values from their attached component, in a one-to-one pairing; a Spotlight component, in light orange, monitors the entirety of the explanatory system. The Spotlight is itself observable, and a LEC can be attached to it, which allows for self-observation of the explanatory system. Interfaces provided by the LECs and the Spotlight are detailed in Figure 4.2.

does not follow the same organization: each LEC is directly connected to the Spotlight, providing the minimal coordination required between explanatory components. However, note that the control system organization can be available to the LECs via their observation interface. This interface, named "Obs" defines read-only access to some variables of the SHC and is compatible with the Autonomic Computing Paradigm (Philippe Lalanda, McCann, and Diaconescu, 2013). In Figure 4.3, for instance, the temperature controller manages the heater and the thermometer. Hence, it has access to some of their variables, which can therefore be observed by its attached LEC. Similarly, a context-aware SHC makes this context available to its attached LEC. Therefore, the explanatory system can benefit from the knowledge distribution of the underlying smart home system without having to specifically adapt to it: in case of re-configuration of the control system, the explanatory system observes the changes and incorporate them. Figure 4.3 also illustrates the potential issue of conflicting goals, as the window here is shared between the

air quality and the temperature system. Control-wise aspects of this situation have been studied in (Diaconescu, Mata, and Bellman, 2018). These situations can be handled by D-CAS and preset necessities: for instance, D-CAS can propose to close the window as a solution, which would raise a conflict with a necessity transcribing the CO_2 controller's goal (in this situation, necessities are used to prioritize conflicts and avoid endless loops, see Chapter 5).

The necessary communications between components of the explanatory system can be categorized in two main types: D-CAS related communications and communications necessary for the self-observation of the explanatory system. Table 4.1 details the different types of exchanged messages. Generally, a message consists of a header indicating its type and a data field. Note the existence of the wildcard header `UNDEF` which allows for additional features and information communication, enabling possible extensions of the original system.

Type	Sender / Receiver	Data Description	Comments
`EXPLAIN`	Spotlight → LEC	Conflict (P, N)	Calls the **investigate()** method on the LEC
`RESPONSE`	LEC → Spotlight	Response $(T, (P, N))$	Response of the **investigate()** method
`KNOW`	Spotlight → LEC	Conflict (P, N)	
`KNOWN`	LEC → Spotlight	Knowledge score	
`RUN`	Spotlight → LEC & Simulator	(P, N)	Simulation of P
`CAN_RUN`	Spotlight → LEC & Simulator	(P, N)	Similar to KNOW, for simulation
`PING`	All components	None	
`PONG`	All components	`OK` / `ERROR`	
`INFO`	LEC → Spotlight	LEC Description Language	Sent by the LEC upon configuration change
`UNDEF`	LEC ↔ Spotlight	Undefined	Used to ask info and modular functionalities

Table 4.1: The message types exchanged between the components of the Explanatory System to enable D-CAS and self-adaptation.

In the next sections, we detail the nature and behavior of each component type of the explanatory system.

4.2 The Local Explanatory Component (LEC)

As defined by teh D-CAS algorithm, the LECs must present the interface shown in Figure 4.2. In addition, as we've discussed in Chapter 3, the role fo the LEC is to keep an updated local description knowledge $\mathcal{P}_{loc}$, inference knowledge $\mathcal{R}_{loc}$ and necessity knowledge ν_{loc}. As such, tt serves as an intermediate between the high-level reasoning of D-CAS and the heterogeneous Smart Home Components. LECs are associated with

SHCs, following a one-to-one pairing. This design choice implies that LECs should be light-weight components hosted on edge computing devices. In the rest of this section, we detail the internal organization of a LEC and how it works. This section is made of four parts. In Subsection 4.2.1, we define how LECs interpret their observations into higher-level representations. In Subsection 4.2.2, we detail what happens in the LEC when it is called on by the Spotlight, via its *investigate* method, to examine a conflict. Ine Subsection 4.2.3, we detail the modularity of the LEC. Then, in Subsection 4.2.4, we show how the design of the LEC enables privacy-compliant communications.

4.2.1 An interface between SHC variables and D-CAS propositions

The role of interpreting local variables exposed by the SHC towards D-CAS-ready Boolean propositions is central to the implementation of the algorithm. This operation consists of evaluating a set of Boolean propositions to form the local knowledge $\mathcal{P}_{loc}$ based on the observations from the attached component. Several challenges have to be accounted for in this process.

First, this interpretation has to fully comply with the principle of knowledge locality underlying in the one-to-one pairing. LECs should be able to understand and communicate the state of their observed component without having to disclose any of its actually measured variables. This requirement further improves the genericity of the resulting overall system. As a result, the interpretation operation can be seen as a privacy-compliant encoding of low-level, private observations into higher-level disclosable propositions.

Another challenge is the different nature of propositions and variables. Propositions are used by D-CAS to represent the user's description of the world using Boolean word-like propositions (Dessalles, 2016), e.g. `hot(room)` or `open(window)`. Notably, the meaning of these propositions, i.e. the mapping with the measured state $\mathcal{S}$ of the world, is not fixed and can evolve with time or the context. In Example 4.1, the interpretation of the proposition `hot(room)` varies with the context. This conceptualization of runtime predicate definition has been mentioned by (Ghadakpour, 2003) and further analyzed by (Dessalles, 2015) within the framework of "conceptual spaces". By contrast, variables transmitted by the SHC correspond to physical measures, settings or commands that represent a "ground truth" that, once recorded, should not be disputed. In addition, it is important to keep track of past measures, without having to re-interpret all of them once a proposition's interpretation has been modified.

Example 4.1

A smart thermometer is located in a room and associated with its corresponding LEC. The LEC's interpretation role is to translate the different readings from the thermometer into propositions such as "hot(room)" or "cold(room)". However, this translation should adapt to the user: for instance, a room temperature of 19°C can be considered "hot" if the user is away, while it is "normal" if the user is present.

To handle these challenges, we have designed a particular memory architecture for the LEC. Its principle is to rely on *twin-memories* of distinct natures. One is designed to store *events*, i.e. high-level timestamped representations of a subset of measured variables. The other stores *predicates*, i.e. Boolean functions operating on events stating whether an event corresponds to a given proposition.

Formally, an event object is defined as as 3-tuple $e = (t, l, X)$:

$$e = \begin{cases} t \in \mathbb{N} \\ l \in \mathcal{L} \\ X = (x_1, \ldots, x_n) \in \mathbb{R}^n \end{cases}, \tag{4.1}$$

where t designates the timestamp of the event, usually encoded as a long integer (milliseconds since epoch), l is a label identifying the event's type, and X is a vector of characteristics defining the event. This definition of events is similar to generic spatio-temporal events defined by (Y. Tan, Vuran, and Goddard, 2009) in the context of CPS. Once an event is stored in this memory, it cannot be revised: in that sense, the event memory can be considered as a Read-Only Memory for the rest of the LEC. It guarantees the existence of a trace of the past measures from the SHC, materialized in the LEC's design by the existence of a dedicated memory, represented in green in Figure 4.4

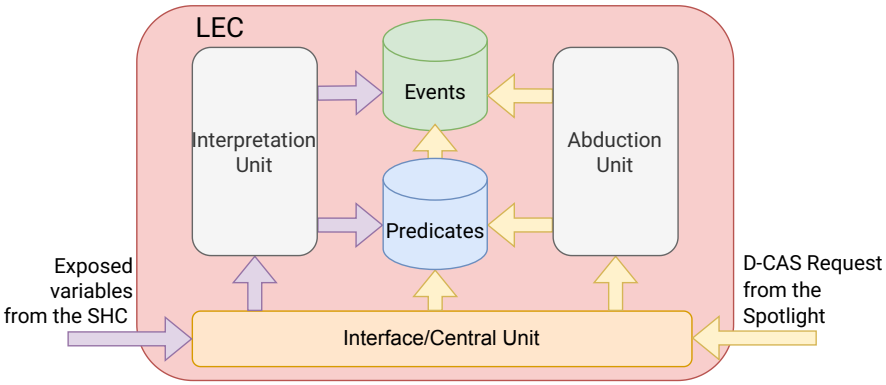


Figure 4.4: The twin-memory architecture of the LEC: events are stored in a designated memory, and cannot be changed afterwards. On the other hand, predicates are designed to be modified at runtime, to express the volatility of the user's notions. Purple arrows indicate the flow of data from the SHC, while yellow arrows show the process of a request from the Spotlight during a D-CAS iteration (see Figure 4.6). The simulation unit is omitted for the sake of clarity, its role being similar to that of the abduction unit.

While being high-level representations of subsets of observed variables, possibly aggregating them over some duration, event objects remain closely related to the variables observed by the LEC and do not fit into the D-CAS model presented in Chapter 3. D-CAS reasons using Boolean propositions. An additional layer of abstraction is therefore necessary. It is implemented in the LEC by *predicates*. Predicates are boolean functions mapping an event recorded by the LEC and additional arguments to a Boolean proposition. In that sense, they are close to the usual definition of predicates in mathematical

logic: symbols representing a property or relation over one or several variables. Formally, a predicate of arity n is defined by:

$$p = \begin{cases} \mathcal{E} \times \mathbb{N}^n \mapsto (\mathcal{P}, \{0,1\}) \\ (e, k_1, \ldots, k_n) \mapsto (rep(p,e), \{0,1\}) \end{cases}, \tag{4.2}$$

where $k_1, \ldots, k_n$ denote the n integer-encoded additional arguments, $\mathcal{P}$ denotes the set of Boolean propositions as used by D-CAS, $\mathcal{E}$ the set of events, and rep a representation function which gives the proposition corresponding to the predicate without supposition regarding its value. To illustrate this, consider the notion of "hot". A naive approach is to consider a room to be hot if the measured temperature of this room is above a predefined threshold, for instance $20°$. The corresponding predicate is p_{hot}, of arity 0, defined as $p_{hot}(e) = (\texttt{hot}(\texttt{e.room}), e.temp > 20°C)$. Similarly, spatial notions can be described: an event e occurring in a room k can be defined as whether the event e, in its dictionary of characteristics X, has a `location` variable corresponding to room k. Thus, p_{in} is defined as a predicate of arity 1, which maps e and k to $(\texttt{in}(\texttt{e}, \texttt{room}_\texttt{k}), e.location = room_k)$.

Contrary to events, that are immutable once stored, predicates can be modified at runtime to account for the possible meaning changes previously evoked. For instance, the threshold of 20 that is used in the p_{hot} predicate can be modified at runtime: this changes which events are considered hot, modifying the perception of recorded events without modifying them.

Following the formal definitions from Equations 4.1 and 4.2, events and predicates can be implemented in Object-Oriented Programming (OOP) languages using the scheme presented in Figure 4.5. Here, the predicate presents two public methods *evaluate()* and *toProposition()*: the former corresponds to the mapping presented in Equation 4.2, the latter to the $rep(p,e)$. In accordance to our definition of predicates as mutable objects, the *evaluate* method can be revised at runtime to adapt to a new interpretation of a notion. For the rest of the implementation description, propositions are defined as String objects, which correspond to the words they describe in natural language.

A specialized unit of the LEC, named "*Interpretation Unit*", is responsible for analyzing monitored variables from the attached SHC and detect events. This unit, visible on Figure 4.4, is also responsible for creating predicates from the same observations and possibly modifying them if need be. The internal organization of this unit will be further described in Section 4.2.3.

The Interpretation Unit is also responsible for evaluating preference necessities, denoted ν_{loc} in Figure 3.4. These necessities encompass the knowledge of the component regarding its different goals. For instance, the temperature controller goal, given a temperature target, will associate negative necessities to propositions corresponding to situations where the target is not reached. This kind of pre-computed necessities (which are computed prior to the explanation reasoning) is used to possibly limit conflict propagation. They can also be used to indicate the confidence of the LEC in its knowledge: for instance, a sensor that is known to be prone to failures can be assigned relatively low necessities, meaning that it would be possible, during the explanation process, to revise its knowledge. This possibility is further explored in the example implemented in Section 5.4.

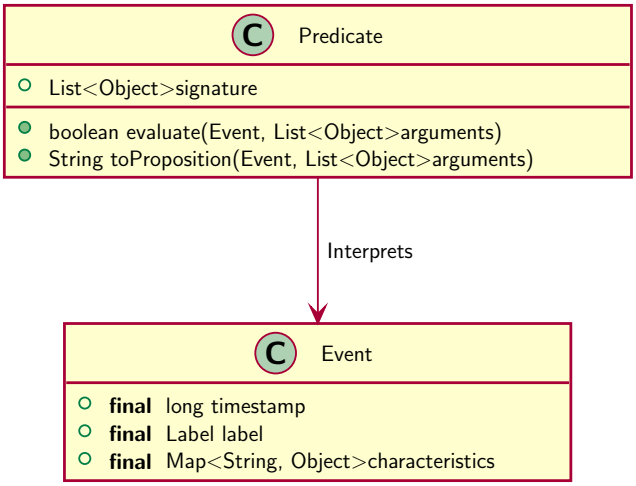


Figure 4.5: Class definitions of events and predicates. Naming conventions are based on Java language. Fields of the Event class are final, following their introduction as immutable high-level representation of past observations.

4.2.2 Reasoning on the LEC

When the Spotlight requests the LEC to provide more information about a conflict, by calling its *investigate* method, the LEC uses its internal knowledge to find the appropriate answer. The overall process, shown in Figure 4.6, is further detailed in this subsection.

(01) The interface of the LEC receives the request, which contains a conflict (P, N) object, as defined in D-CAS formalism (see Chapter 3). (02) The LEC then checks with its memory of predicates if it contains elements that could have generated the proposition P. This confirmation operation is prime, as it bridges the gap between the Boolean proposition used for the D-CAS rationale and the variables exposed by the SHC which have been processed by the LEC and stored as events. For the LEC, this operation consists of identifying whether a recorded event e from the event memory and a predicate p from the predicate memory can evaluate the desired proposition P to be true. That is, whether the following equation is true:

$$\exists \begin{array}{r} e \in \text{Events} \\ p \in \text{Predicates} \\ (a_1, \ldots, a_n) \in \text{Args}(p) \end{array} \left| \; p(e, a_1, \ldots, a_n) = (P, \texttt{True}) \,, \right. \tag{4.3}$$

where Args(p) denotes the set of admissible arguments for predicate p.

(03) In case this search is successful, the process continues within the LEC: the identified event e, the predicate p and the arguments $(a_1, \ldots, a_n)$ are passed to the *abduction unit* (04). This specialized unit possesses inference knowledge which it can use, alongside access to the event (05) and predicate memories (06), to propose possible hypotheses and evaluate their likeliness. Then, the most fitting one, according to the abduction unit, is returned, as exposed in Chapter 3 (07). This result is presented as a new conflict (C, N_C), where C is the proposed hypothetical cause proposition and N_C its associated necessity. In parallel, the Simulation Unit of the LEC (not represented

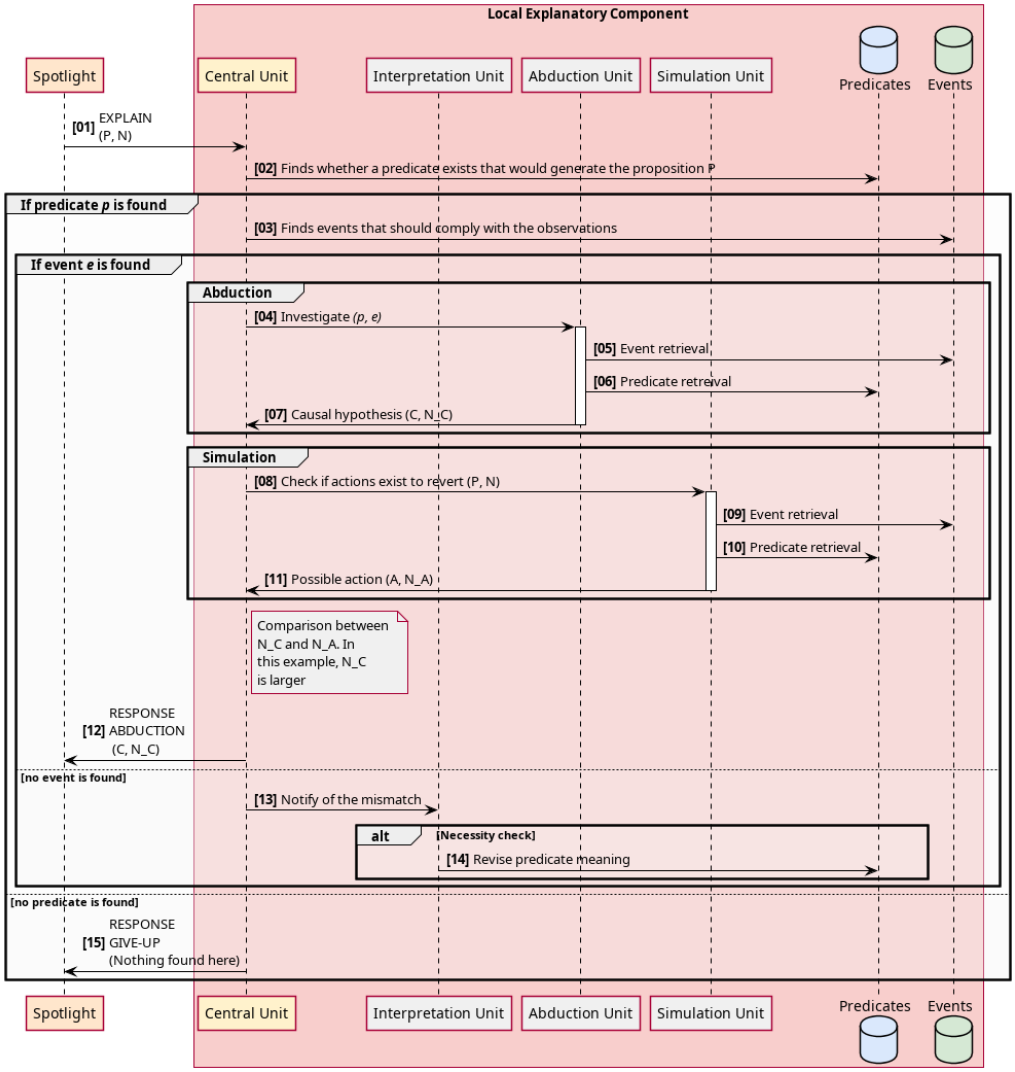


Figure 4.6: Sequence diagram of the processing of a request from the Spotlight by a LEC.

in Figure 4.4) is requested to identify a possible action to revert the examined conflict (08). The Simulation Unit has access to the events (09) and predicates (10) memories to gain additional knowledge of the context. It then returns its best proposal as a couple (A, N_A) (11). In the example depicted here, the LEC choses the abductive hypothesis as the best result, and returns it to the Spotlight (12). result can then be transmitted back to the Spotlight if the propagation criterion is met (i.e. this hypothesis has not yet been considered with a higher intensity). This choice is made by comparing the necessities N_A and N_C: a lower intensity denoting an easily mutable proposition. In case no suitable predicate is found that corresponds to the request of the Spotlight, the returned `RESPONSE` object contains a give-up instruction, with additional information detailing that the LEC was unable to understand the proposition (15).

In D-CAS formalism, necessities model the agent's opinion regarding its observed Boolean propositions. AS they are a unifying metrics for conflict intensities, they are

present in several units of the LEC. Necessities that translate prior opinion regarding beliefs and wishes are handled by interpretation modules, that can assign some Boolean propositions with necessities to translate the Autonomic System's goals into the explanatory reasoning. For instance, `cold(room)` may be associated with a negative necessity, as the system's goal is that this proposition is not realized. Another possible source of necessity can be the evaluation of a probability by the interpretation module. For instance, a proposition `fault(component)` may be associated with a strongly negative necessity, showing that the component considers this proposition as being highly unlikely, considering its observations.

Necessities are used in the process shown in Figure 4.6 to quantify the quality of the causal hypotheses and actions proposed by the Abduction and Simulation Units. The necessity N_C of the returned hypothesis C is a number such that $-|N| \leq N_C \leq |N|$, conforming to the propagation condition of D-CAS that no hypothesis with higher necessity than the incoming conflict can be considered. The exact intensity $|N_C|$ is computed by the Abduction Unit to measure its confidence in the hypothesis: $|N_C| = |N|$ shows a full confidence, while a small value shows that the unit thinks the causal link is unlikely (but has no better suggestion). As this necessity N_C is later transmitted back to the Spotlight (12), an unlikely hypothesis will be treated as a low-intensity conflict by the rest of the system. Thus, it avoids making potentially harmful or complicated decisions following a bold claim. For instance, if an Abduction Unit suggests, with low confidence, that switching off the entire electric system might solve the room being too hot, the low intensity attributed to the hypothesis will not be enough for the system to consider doing so. Similarly, the score N_A given to the action A (11) denotes the confidence of the Simulation Unit that this action might revert the conflict. Thus, a simple comparison between the values N_A and N_C can be used to identify the response to return to the Spotlight.

In case the proposition is understood by the LEC, i.e. there is a predicate p in its knowledge that corresponds to the proposition P, but no event e is found that makes the proposition true, the Central Unit notifies the Interpretation Unit of the mismatch (13). This notification can trigger a knowledge revision, and the Interpretation Unit may revise the meaning of p. (14) The realization of this revision is conditioned on the intensity $|N|$ of the incoming conflict and the confidence of the Interpretation Unit regarding the definition of p. For instance, if $(P, N) = (\texttt{failure(component)}, -50)$, and that the Interpretation Unit knows that this particular component has a history of past failures, it may admit that it is currently failing, thus changing the definition of $p = failure()$ so that it is realized. Conversely, if the Interpretation Unit is confident enough that the component is not undergoing any failure, it may refuse to revise its knowledge, resulting in a GIVE-UP response to the Spotlight (15).

Note that the inference knowledge $\mathcal{R}$ defined in CAN formalism and used by D-CAS to perform both abductive inference and imagine the consequences of counterfactual actions (i.e. simulations) is implemented in separate abduction and simulation units in the LEC. This follows our general implementation principle of separating functions to allow for a better adaptation. This principle is further extended to enable modularity.

4.2.3 A generic and adaptive platform

The LEC is directly connected to a SHC which can undergo different changes, either internal (changes of configuration, software update, goal changes) or external (displacement within the house, change of external conditions). At its level, the SHC implements context-aware self-adaptation as it is the case in many smart devices/controllers (Silverio-Fernández, Renukappa, and Suresh, 2018). (P. Lalanda, Gerber-Gaillard, and Chollet, 2017) introduce a smart home simulation platform integrating context-aware components, and (Marikyan, Papagiannidis, and Alamanos, 2019) even define the "smart" property of devices as being strongly correlated with this self-adaptation capability. Therefore, this requirement of SHC being able to locally handle changes is coherent with state-of-the-art development. In consequence, the LEC should also adapt to these changes and integrate them into its knowledge to propose them in subsequent explanatory reasoning. This is done by observing the exposed state of the SHC and analyzing it, in the same way as other data is processed by the LEC. Thus, self-observation-related events can be extracted by the Interpretation Unit of the LEC: e.g. an event labeled `event/move` is created after the LEC observes that the exposed location of the component changed. Here, the *characteristics* field of this event includes the `from` an `to` positions of the device, the name of the device that moved and its type. As it is detected and stored in the event memory of the LEC, this event can then be proposed and integrated to the explanatory rationale as any other event.

Knowledge of the attached component is prone to change following software updates and context modifications. In addition, as events of different nature can be recorded and handled by the same LEC (e.g. configuration events, error events, temperature events can all be recorded by the LEC attached to a thermometer), it is reasonable to adopt a modular approach to this knowledge representation as it allows for better configuration and customization. The LEC can be seen as a generic local platform, where each of the main units, namely Interpretation, Abduction and Simulation, are divided into smaller specialized "plug-and-play" modules. Figure 4.7 shows this organization: upon the common Communication Interface, different modules can be added, updated or removed at runtime. A Manager integrated to the LEC keeps track of the changes and accordingly stores changes, maintaining an up-to-date Internal Context. Similar to SHC observation, this self-observation is recorded as events in the event memory. Thus, addition or update of LEC capabilities can be integrated into future rationale.

Another role of the LEC adaptation logic is to keep track of meaning modification of predicates. As these changes transcribe a meaning shift for certain concepts, it is important to keep track of them. This capability is achieved via monitoring when updates are made to predicates in the predicate memory. Thus, when an interpretation module updates a predicate to adapt to a new environment or knowledge, this update is recorded by the manager. Subsequently, another Interpretation Module can analyze the recorded modification and create a corresponding event (and predicates). This event can thereafter be integrated into the explanatory rationale as any other would. To illustrate this, consider Example 4.2: the predicate p_{cold} has changed its threshold as a reaction from the user request that the room *feels* cold. The change is recorded as a second event of type `predicate_change`, with characteristics such as the old and

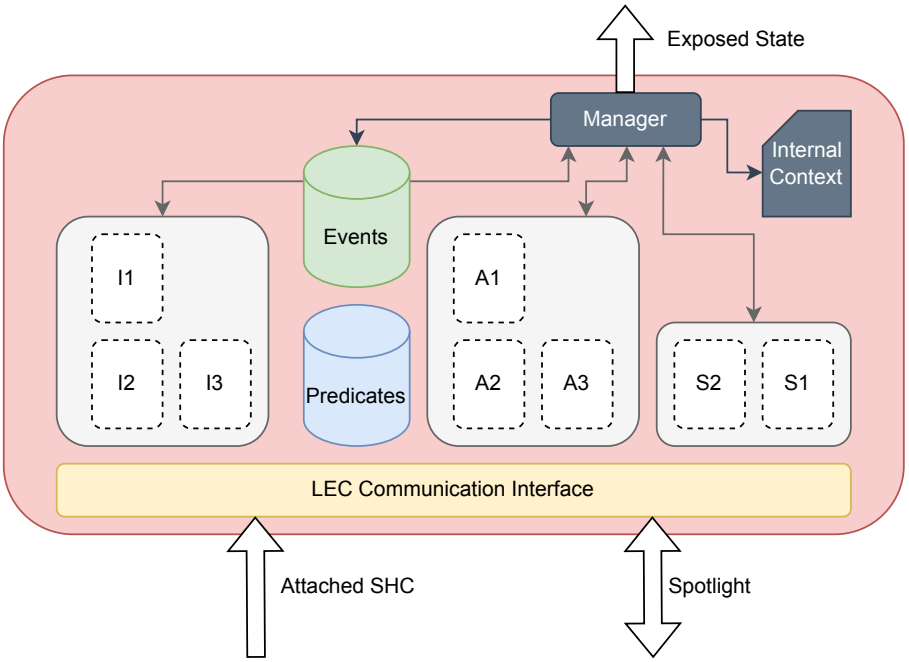


Figure 4.7: The modular architecture of the LEC. Here, modules I1, I2 and I3 are interpretation modules; modules A1, A2 and A3 are abduction modules; modules S1 and S2 are simulation modules. A local manager keeps track of the internal configuration and keeps an updated context which can be accessed by the different modules. This context contains information regarding the types and capabilities of modules (e.g. their vocabulary in terms of variables and propositions, their software version). The manager can then expose some of the internal changes of the LEC to the exterior, using Boolean propositions resulting to the application of predicates to recorded events.

new threshold values. Then, it can be integrated into the rationale, allowing D-CAS to propose the following answer: "it was cold because the temperature threshold was inadequate and had to be changed".

As the LEC is made of many specialized modules, these modules have to be defined. Since they require specific knowledge of the SHC, namely its variables and logic, it seems adequate to rely on a component-based approach. We suppose that, alongside their usual specifications, SHC's are able, when they are integrated into the system, to expose the required modules and their characteristics to their LEC. The Spotlight, which will be further detailed in Section 4.3, then creates the LEC as requested and ensures it is connected. The responsibility of the SHC's explainability is left to the component's manufacturer, which arguably possesses expertise over the component's workings, use cases and abilities. This specification is described in Section 4.5

Example 4.2

A user experiences unusually low temperature in a fully-equipped smart room. He/she inquires the explanatory system for this situations. D-CAS identifies the issue to be the competence of the temperature controller and forwards the request to it. However, the temperature controller does not consider the room to be cold. But given the high discomfort of the user, the intensity of the conflict is high enough for the controller's LEC to reconsider the meaning of its predicates. It changes the detection threshold of the cold predicate. This change is monitored by the LEC's self-adaptation capabilities: in reaction, an event is recorded to store this meaning change. This event can later be proposed as the final explanation: the room was cold because the temperature threshold was too low and had to be reconsidered.

The exact inner working of the Interpretation, Abduction and Simulation Modules is considered out of the scope of this thesis. Each of these topics raises scientific questions that would require significant developments. For instance, the problem of mapping human-understandable concepts such as D-CAS propositions to streams of measured variables could be addressed through data mining, anomaly detection or NLP; or the abductive inference problem could be examined by using current XAI solutions. For instance, feature-relevance methods, such as LIME (Ribeiro, Singh, and Guestrin, 2016), can identify the most significant parameters to a component's decision and generate corresponding propositions as possible causal hypotheses. Some naive implementations that were used for the development of our proof-of-concept demonstrator will be described in Chapter 5. Chapter 6 will focus on a novel method for abductive inference based on proposing memorable events as relevant causes.

4.2.4 Preserving knowledge locality and privacy

To preserve privacy, inter-component communications do not disclose measures from the SHC, nor events or predicates. Rather, they exchange the outputs of the predicate functions, i.e. the propositions and their values (in our implementations, Strings and Boolean values). This isolation allows to keep potentially critical information at the local level, and avoids unmaintainable knowledge base: for instance the thermometer LEC from Example 4.1 will only disclose the information $(hot(room), true)$, without further indicating the meaning of this word. This separation between syntax and semantics is analogous to what is observed in the "Chinese Room" counter-argument to AI: *processing* Chinese characters is a different ability than actually *understanding* them (Searle, 1980).

In the smart home explanatory system, this distinction allows communications between devices while preserving the locality of knowledge: only propositions, i.e. symbolic representation of the observation and interpretation of the LEC, are communicated, while their actual meaning remains localized within the relevant component. The existing twin-memory architecture can be further improved towards privacy and shallowness: by using an additional `disclosure_level` to the predicates stored in a LEC, it is possible to only disclose their associated propositions to selected components. This is similar to existing

methods that preserve critical data within smart home systems (B. M. Jakobsson and A. K. Jakobsson, 2021). Example 4.3 below presents a possible application of such levels to the explanatory system.

Example 4.3

A heat pump is a critical equipment in a smart home. During winter, a component of the heat pump defects, provoking perceptible consequences on the house temperature. This defect is identified by some autonomic component which raises some error code, e.g. 102. The user feels the temperature drop and investigates why this situation occurred. By asking the D-CAS system, he/she is at first presented with the simple `failure(pump)` proposition. This may be sufficient for some occupants, but an experienced user or a technician would gain more insight from the proposition `internal_failure(pump, code102)`.

Here, different comprehension levels are set to enable distinct options depending on the target of the explanation, for instance between an average user and an expert. In such circumstances, the handling of the deeper investigation level is left to D-CAS: by setting a higher necessity threshold on the predicate with higher disclosure level, D-CAS will first propose the high-level predicates. Then, if the user reiterates his/her request with more intensity, this will allow disclosing lower-level predicates. This possibility is in line with the shallowness goal of the system, as well as a first response to the question posed by (Maxwell et al., 2020a) concerning the existence of several levels of explanation.

4.3 The Spotlight

In Chapter 3, we introduced the Spotlight as a central component orchestrating the D-CAS algorithm to generate a system-wide explanatory reasoning. To implement the interface presented in Figure 4.2, the Spotlight does not need to have knowledge about the state of the system and its environment, but only to keep an updated base of pointers to the different LECs in the system. To acquire more knowledge, for instance about the different components capabilities, the Spotlight relies on calls to the different components[2]. Given that the Spotlight must already keep this knowledge of the system and its central position, it makes sense to localize the naming and directory service of the explanatory system in the Spotlight.

Relying on a single component for system-wide adaptation may introduce vulnerability with regards to reliability of the system. However, given that the Spotlight does not need to keep a specialized knowledge, its role can be easily overtaken by a back-up component in case of failure. This possible redundancy of the Spotlight is a common paradigm within Cyber-Physical Systems (Rajkumar et al., 2010), where robustness of the system is a key feature, especially for critical applications (electric grid management, heavy industry applications).

[2]While there is no requirement for it, implementations can include some kind of buffered storage of the LECs' capabilities to decrease the number of communications.

The Spotlight itself presents self-adaptation features, as it updates its knowledge to the configuration of the system. Similar to LECs, its capabilities can be upgraded or updated following the same modular architecture. Its internal organization is shown in Figure 4.8: a base communication interface serves to the D-CAS module, which implements the algorithm seen in Chapter 3. An Explanatory Architecture Manager is responsible for tracking the different changes occurring in the rest of the system: addition, removal, change, failure of one or several LECs. Additionally, a module manager allows different extension modules to use the communication interface to eventually add new capabilities to the explanatory system at runtime. For instance, an "intercom" module can be added to allow communication between LECs, turning the Spotlight into a "middle agent" (Decker, Sycara, and Williamson, 1997): e.g. the window LEC sends a message, via the Spotlight, to the thermometer LEC to gain some knowledge, which could potentially improve the component's overall explanatory capability. This kind of additional communications is enabled by the specification of a wildcard communication type in our specification in Table 4.1.

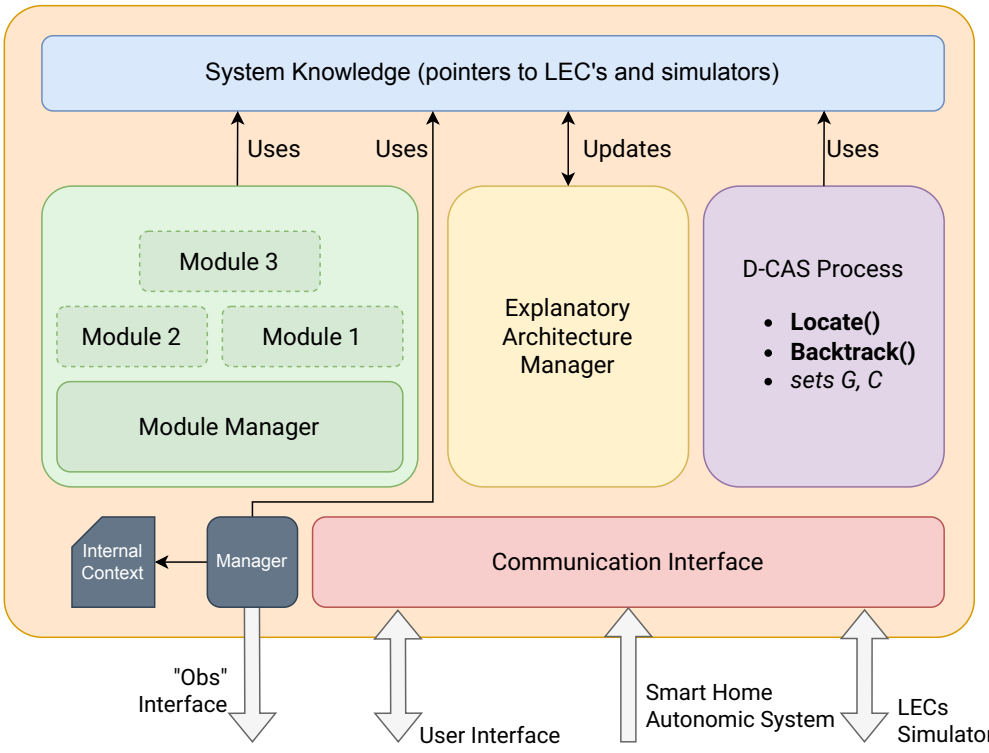


Figure 4.8: The spotlight is composed of a fixed D-CAS operating module, a knowledge base of pointers to all LECs in the system, an autonomic management unit and a communication interface with all other components in the system. Additional modules can be plugged in to offer additional services to LECs.

However, contrary to LECs, the Spotlight's self-observation capability is not directly integrated into the explanatory process: when an internal change occurs, it is not translated into usable D-CAS knowledge. This choice is in accordance with the principle of generic design exposed in Chapter 3 which implies that the Spotlight has only minimal

knowledge. Allowing the Spotlight to have internal predicates and events would have breached this design. However, the Spotlight implements the "Obs" interface: some of its internal context is exposed via a manager, which allows to attach a LEC to the Spotlight, resulting in what is shown in Figure 4.3. This Spotlight LEC is registered into the Spotlight's naming and directory service and contains knowledge regarding the general organization of the explanatory and control system. Hence, it can be included, as any other LEC, in a D-CAS reasoning. This allows to include self-observation propositions into the explanation, e.g. `device_added(dev_id)` or `device_removed(dev_id)`.

4.4 Simulators

D-CAS integrates simulation as the continuation of the Negation operation that is originally present in CAN (Dessalles, 2016). By simulation, we designate the operation that considers the potential outcomes of an alternative scenario, similar to counterfactual thinking. In D-CAS, simulation, as abduction and interpretation of propositions, is delegated to expert components. To account for the diversity of available tools to simulate the system's behavior, the method described in Chapter 3 left their implementation and characteristics unspecified.

The requirements defined by D-CAS for the simulators are that they can expose the range of their knowledge, i.e. whether they are able to perform a given action, encoded as a conflict (P, N), perform this action and then broadcast its results action to the other LECs present in the system. This broadcast operation can be achieved by relying on the Spotlight's knowledge of the system's composition. To prevent unnecessary computations in trivial cases, simulators can implement a safeguard that is similar to the propagation condition used for abduction modules. If the intensity $|N|$ of the action (P, N) to simulate is not high enough, the simulator can refuse to perform the operation. Here, necessity is used as a mutability score, similar to one of its original use in CAN.

A first range of simulation methods relies on localized expert knowledge, for instance in the form of a finite set of rules, Bayesian models or case-based reasoning. Numerical methods relying on neural networks have long been considered too complex and power-hungry to be realistically embedded in edge devices. However, the emergence of parsimonious neural networks enables high-performance simulators which can be embedded into low-power devices (Bompard et al., 2020). As previously stated, defining the inner working of these modules is considered out of the scope of this research. Given their small computational requirements, it is possible for such methods to be implemented at the LEC level, similarly to abduction methods. In Figure 4.7, a simulation module is present in the LEC to provide such capabilities. The presence of these lightweight simulation modules directly in the LEC can provide tools for abduction validation. For instance, the module can perform a quick simulation to check if its proposed hypothesis is coherent with simulation results.

It is also possible to rely on heavier simulation methods, often modeling the entirety of the building. Among them, we can cite the paradigm of the "digital twin" (Negri et al., 2019), which aims to reproduce the behavior of an entire system. Since this kind

of simulation requires heavier computations, it can be hosted on a distinct device, as in Figure 4.3.

Simulating interactions between devices is a challenging issue for scalability: the number of possible one-to-one interactions increases as the square of the number of variables within the system. However, models of the system may include prior knowledge about its organization which would decrease the complexity of the simulation model: for instance by considering interactions within the same room, and room interactions at a greater level, one can simplify the simulation of a large building. This kind of multi-scale approach is already widely-used in complex systems simulation, for instance for the study of grid-level power consumption (Albouys-Perrois et al., 2022).

Integration of new devices into existing simulators is handled the same way as classical handling of devices. In case a dedicated module handles system-wide simulations, it is notified of the new device addition, and can access its exposed characteristics. Thus, if need be, the newly added device can be injected into the simulation model. For instance, our first implementations of the demonstrator were based on a smart home simulator named iCasa (Philippe Lalanda and Hamon, 2020). This software handles runtime injection of new modules and components, which would be suited for its adoption as a system-wide simulator, adding new devices as described by their specifications.

Regardless of their hosting device, all simulators must be able to broadcast their results to their surroundings. This can be achieved by consider context-aware components: simulators know with which components they can send their results to. Simulators' output can consist of data streams, similar to data generated by real devices. This similarity allows LECs to process the data regardless of its origin, be it real observations or simulations. It is also possible that a simulator requires a certain kind of interpretation module to make sense of its results; in which case the simulator module should expose this requirement in one of its characteristics to allows its integration into the system.

4.5 Self-* capabilities

Self-awareness, implies many self-* capabilities (e.g. self-observation or self-adaptation) and the ability for the system to model its own behavior and integrate it in its reasoning (Kounev et al., 2017). In Chapter 3, we identified this ability as a target goal for a smart home explanatory system. This is particularly exemplified with what we call the "plug-and-explain" capability. Since the introduction of a new device can lead to unprecedented situations which may require explanation, this feature is particularly set as a target for a convincing smart home explanatory system. We propose to study how the architecture described in this chapter is compliant with the principles of self-awareness.

The sequences presented in Figures 4.9, 4.10 and 4.11 show the integration process of a new component in the smart home control system, its typical use over its lifespan and its possible removal. The prerequisite for these sequences is that the new device presents a description exposing its capabilities in terms of explanation, and that this description is somehow handled by the naming and directory service of the smart home autonomic system which is accessible to the Spotlight.

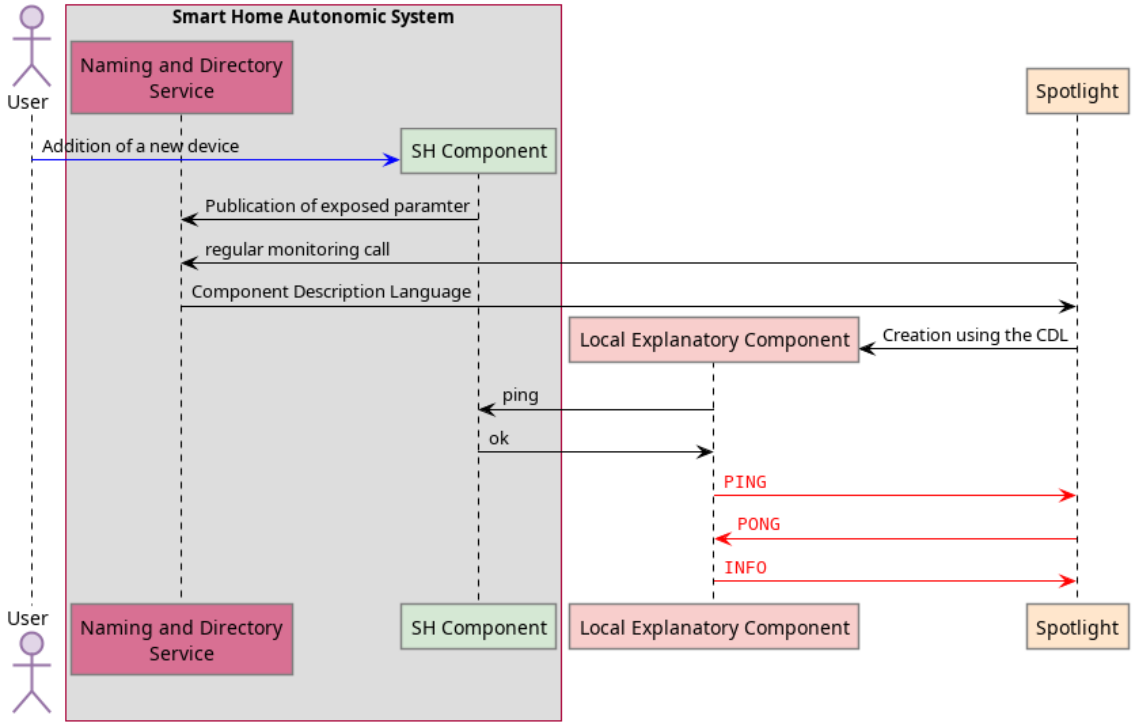


Figure 4.9: Sequence Diagram illustrating how the explanatory system integrates a new SHC into it by creating the corresponding LEC. A blue arrow indicates actions initiated by the user, black arrows interactions involving the Smart Home Autonomic System and red arrows intra-explanatory system communications, which are detailed below in Table 4.1.

The sequence of Figure 4.9 is typical of self-integration: upon its addition, a newly introduced component exposes its characteristics to an autonomic registry of the control system. This registry is exposed and periodically observed by the Spotlight. Upon observing a change, the Spotlights then triggers the creation of a new LEC. This creation uses the information exposed by the SHC as the "recipe" for this new LEC. Here, the information exposed by the SHC should therefore include the number and types of abduction, interpretation and simulation modules, which SHC variables are exposed, how to access them and other possibly useful characteristics.

Figure 4.10 also shows the process of a configuration change at the level of a LEC: in case the observed component presents self-adaptive abilities, the LEC observes these changes via the exposed state and context of the component, similar to what is proposed by (P. Lalanda, Gerber-Gaillard, and Chollet, 2017). In this case, the necessary configuration changes are handled locally by the autonomic manager of the LEC. Once such changes have been performed, the necessary information is exposed to the Spotlight through an `INFO` message (see Table 4.1) which contains a description of the current state, configuration and perceived context of the LEC.

This sequence illustrates how the explanatory engine implements Adaptation Logic at two distinct levels: LECs can individually follow changes occurring within their knowledge domain or internal to their attached components, as occurs during their typical

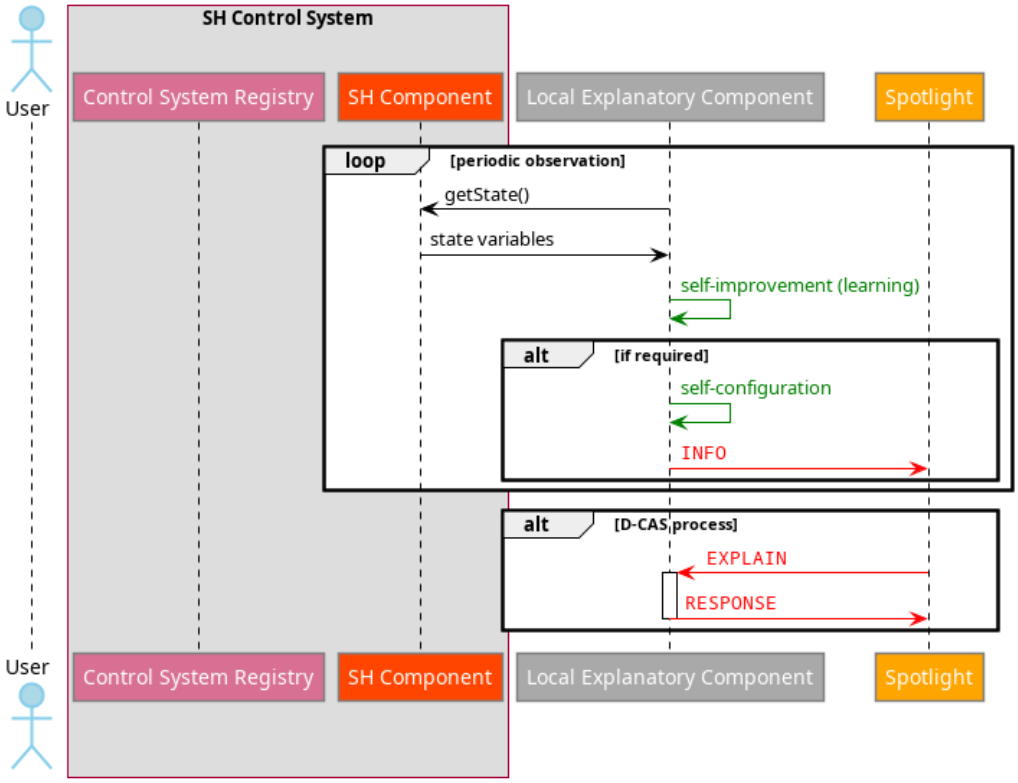


Figure 4.10: Typical use case of a LEC: the component periodically observes its SHC, and performs the required configuration changes when needed. In case a D-CAS rationale is run, it exchanges information with the Spotlight.

use cases; the Spotlight tracks and monitors changes happening in the entire system and facilitates communication between the different components, which is required for runtime integration, i.e. "plug-and-explain". The final step of a LECs life cycle is shown in Figure 4.11: its deletion follows the one of its attached SHC, and it notified to the Spotlight.

Our self-observation logic is based on a generic description language for the different components of the explanatory system. Our choice here is to design this description as an extension of existing description languages for smart components, such as the Smart Object Description Language proposed by (Burmeister, Burmann, and Schrader, 2017). This language enables integration and observation of smart devices within a smart home: for instance, components expose their capabilities (e.g. temperature management, window control, security control). A similar exposure is available in the smart home simulator (Philippe Lalanda, McCann, and Diaconescu, 2013) where an autonomic manager is able to detail representations of the different components in the house. Our resulting description language is specified in Figure 4.12.

As previously mentioned, all these communications follow the principle of locality of knowledge: events and predicates are *never* communicated between components, as they remain limited to their host LEC.

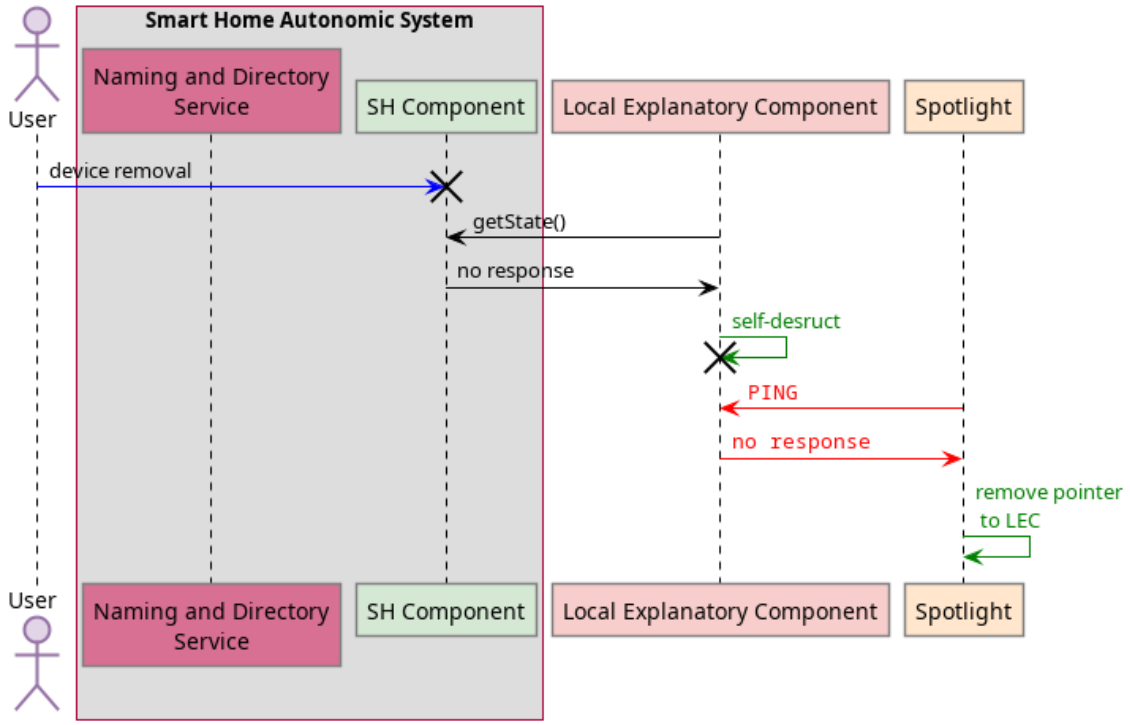


Figure 4.11: Deletion of a LEC following the removal of its attached SHC from the smart home system.

Overall, the architecture organization of the explanatory system fulfills the target goals defined in Chapter 3: it relies on generic platforms which are specialized by implementing specifications that are exposed by SHCs via their description. This specialization consists of: i) specific interpretation modules, which associate D-CAS knowledge model to observed physical variables; ii) specific abduction and simulation modules which implement local causal knowledge. The twin-memory basis of the LECs enables the locality of events and predicates meanings, as only the propositions are exchanged between components without disclosing their meaning. Self-adaptation of the explanatory system is performed via observation of the SHC and self-observation: eventual changes are locally recorded as events, which can be used to define propositions and integrated into explanatory reasoning.

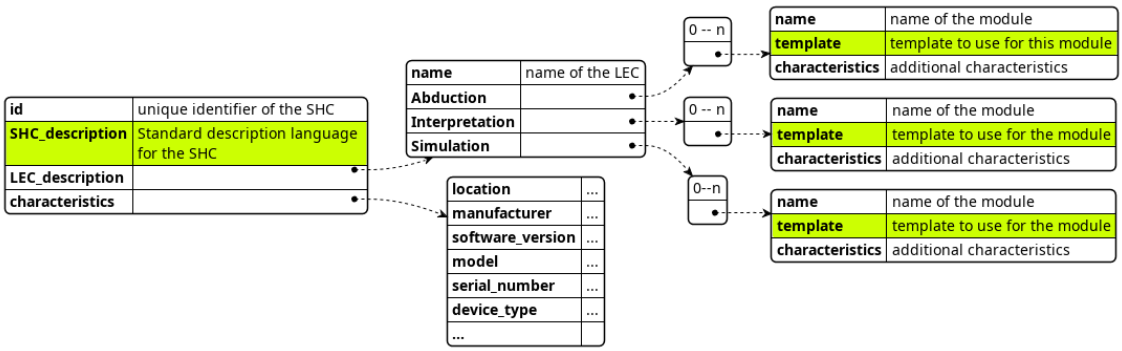


Figure 4.12: Description of an SHC enriched with the description of the attached LEC. This kind of description can be used both for the integration and change adaptation of either the SHC or the attached LEC (addition or change of a module). The highlighted fields may not be publicly disclosed, as they contain potentially critical information regarding the structure of the component. By contrast, the characteristics fields are exposed to other components in both the control and the explanatory system.

Chapter 5

Realization of a Proof-of-Concept Demonstrator

Summary

To test the feasibility and the performance of both the D-CAS algorithm exposed in Chapter 3 and the explanatory system architecture detailed in Chapter 4, we realize a proof-of-concept demonstrator. This implementation models a smart home equipped with the explanatory system. In this chapter, we describe the three successive versions of the demonstrator that were realized over the course of this research. We motivate changes between versions.

The latest implementation is built upon a physical smart home model using physical sensors and edge computing units. The control system uses an event-based framework to provide the necessary observation and adaptation capabilities. The explanatory system is written in Python and observes the control system via REST requests. An additional web interface is added to interact with both the control system and the explanatory system. This interface includes a visual representation of the output rationale of D-CAS, where the different steps are nodes of a tree graph. This representation allows to grasp the entirety of the explanatory rationale.

We evaluate the performance of the demonstrator qualitatively by testing different situations that reproduce the examples exposed in Chapter 1. These examples are inspired from on-field experience of EDF employees and motivate the development of the smart home explanatory system. As such, they represent simple and yet challenging situations to explain, showing the potential of the system along with its limits. To evaluate the performance of the demonstrator, we tested different situations which we deemed interesting to explain. Most of them reproduce the examples exposed in Chapter 1 which are inspired from field reports by EDF employees.

5.1 Implementation choices

Over the course of the research, three consecutive versions of the demonstrator have been realized. Each one targets specific aspects of the explanatory system. These various goals, and the lessons learnt from the limitations of the previous trials are visible in a wide range of aspects: communication protocols, hardware, software and data storage differ between versions. Before describing the latest and most advanced version to greater extent, we first briefly review the previous drafts.

5.1.1 Previous versions of the demonstrator

First Version The first version of the demonstrator aimed at showing the feasibility of using a decentralized knowledge of predicates to generate system-wide explanations. It was implemented in Java upon the iCasa platform. iCasa is a pervasive computing platform (Philippe Lalanda and Hamon, 2020; P. Lalanda, Gerber-Gaillard, and Chollet, 2017) that proposes a smart home simulator implementation. This simulator contains a context manager which allows the addition of custom self-aware, context-aware components. It is based on OSGi (Tavares and Valente, 2008), a toolkit to enable component and service-oriented programming in Java. As such, this technology is used for various IoT applications (Wu, Liao, and Fu, 2007; C. Lee, Nordstedt, and Helal, 2003). In this version, each component, both of the control system and the explanatory system, consisted of an iPOJO object (Escoffier, Hall, and Philippe Lalanda, 2007), i.e. a Java class augmented with annotations to specify the OSGi-specific capabilities of each component. Thanks to the iCasa-provided managers, each of these components can publish its services and requirements and expose its state, which enabled the realization of a first adaptive explanatory system. In addition, a Graphical User Interface (GUI) was also implemented in this component-based approach. Figure 5.1 shows this interface.

This version allows to play basic scenarios and run a first draft of the D-CAS algorithm to generate explanations in this context. It validated the main D-CAS principles: separation of knowledge (predicates encoding the state of the world are stored in different components) and a rationale based on conflict identification and propagation. However, this implementation presented some major issues that required modifications and, ultimately, the development of another distinct version.

First, despite being written as different components, the LECs are operating in the same runtime environment as the SHC, and are monitored by by a unique manager. This close relationship questions our genericity principle: the LECs are too much integrated within the iCasa environment to be considered generic, they rely on the same technology as the control system implementation. Second, this demonstrator is written in Java, which, while being an industry-grade programming language, can be seen as too rigid to suit the purpose of a proof-of-concept demonstrator. As the research progresses, the implementation undergoes major revisions and changes, at the same time for the D-CAS algorithm itself, knowledge representation and explanatory component capabilities. Third, as iCasa runs on a single machine, it is not possible to demonstrate the compliance of our architecture "physically" with smart-home hardware capabilities and inter-device communications; similarly, it not possible to experience the "plug-and-explain" feature

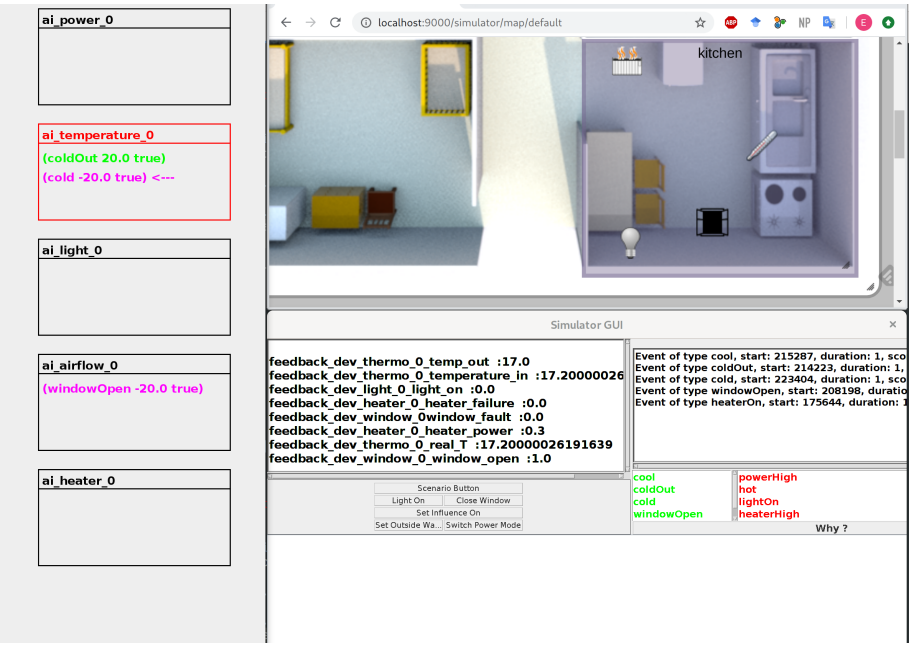


Figure 5.1: Interface of the first version of the demonstrator, written in Java. On the left hand side, a panel shows the LECs and the propositions they know, each assigned with a value and a necessity. In the top right corner, the existing iCasa web interface shows the simulated room with four devices. In the bottom right corner, a list of true and false propositions, and buttons allowing basic interaction with the simulation (opening a window, changing outdoor temperature to a fixed low value).

of the explanatory system, as the addition of new hardware at runtime is not managed. Fourth, the interface between propositions and variables is not satisfactory: propositions were limited to predefined threshold comparisons and events were associated to a proposition's value switch, which limited the observation capability of the explanatory system. This change alone requires major modifications, and motivates a new draft of the demonstrator.

Second version To tackle the first two limitations identified in the demonstrator's first version, a new implementation was realized. The layered architecture presented in Figure 4.1 is strictly followed by isolating the explanatory engine and the control system. Again, the smart home physical environment corresponds to the iCasa smart home simulation, and controllers are implemented as iPOJO components upon this basis. The explanatory system is implemented in Python, a language that is well-known within the Computer Science community for its flexibility and that appears as a prime choice as a prototyping language. Its main downside is its speed, as it is an interpreted language. However, this is not an issue in the current context of designing architectures and algorithms. The Python layer interacts with the iCasa instance hosting the smart home environment and control system by using the REST API exposed by iCasa. This directly follows the guideline that the explanatory merely observes the control system, without the latter being aware of this observation. In addition, a web interface shown

in Figure 5.2 is also implemented. As the GUI is now based on web technologies and languages (JavaScript and Bootstrap), it is arguably more polished than the previous one. It also introduces the visual representation of D-CAS explanatory output as a tree, which will be further presented in Section 5.3.

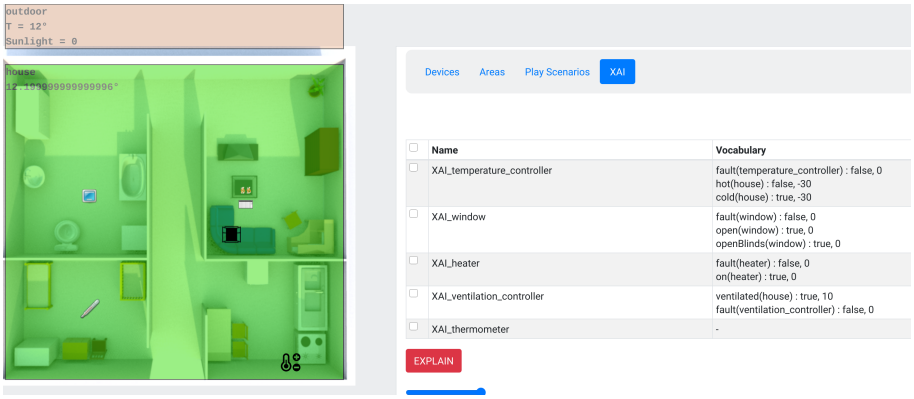


Figure 5.2: Interface of the second version of the demonstrator. It controls both the Python-written explanatory system and the iCasa simulator underneath.

The goal of this version is to instantiate the possible genericity and self-adaptation allowed by the architecture in an implementation. With a better isolation of the explanatory system from the control system technologies, this version is able to generate the same explanations than the first one, using D-CAS to coordinate calls between different LECs. It also introduces LECs as generic platforms upon which different modules can be added, depending on the SHCs exposed characteristics. However, this adaptability is for now limited to only predefined modules, limiting its capability.

While this version brings more flexibility and independence between the different layers of the architecture, it still faces some issues. First, it lacks the desired adaptive characteristics and fails to implement the architecture in a realistic setup: all components are still run on the same machine, a mid-range laptop that is not representative of existing smart-home hardware. Notably, no specific communication protocol is defined between components as they all are hosted in the same application. Second, the variables-proposition interface does not differ from the first draft, and still lacks the desired adaptability and flexibility.

5.1.2 Implementation choices for the third version

As the main goal of the demonstrator is to show the ability of D-CAS and the underlying explanatory system to work in realistic environment, we realized a third version of the demonstrator that aims to rely on hardware sensors and on the physical environment. Contrary to previous versions, the explanatory engine is now scattered across several hardware devices and the control system interacts with physical devices.

Hardware

A wide range of hardware computing units can be found in smart home projects: from task-specific Field Programmable Gate Array (FPGA) that are highly efficient (Alhafidh et al., 2018) to generic x86-based or ARM-based CPU. The main characteristic of smart home computing units is to be a low-power equipment that can be embedded directly into devices. For the purpose of our demonstrator, we decided to settle on using Raspberry Pi's (RPI): these small ARM-based computers are frequently found in smart home projects (Wen and Wang, 2018) and benefit from a large community, which means that most software is ported and can run on RPI. Notably, specific GNU/Linux distributions exist for RPIs, allowing to use a well-known and reliable Operating System (OS). Being the size of a credit card and costing less than a hundred dollars, it is reasonable to consider a smart home system relying on several RPIs, each in charge of a handful of devices. To allow several devices to be handled by a single RPI, we use "hats", i.e. extension cards that plug into the RPI'pins and add separate slots for different devices, visible on Figure 5.3.

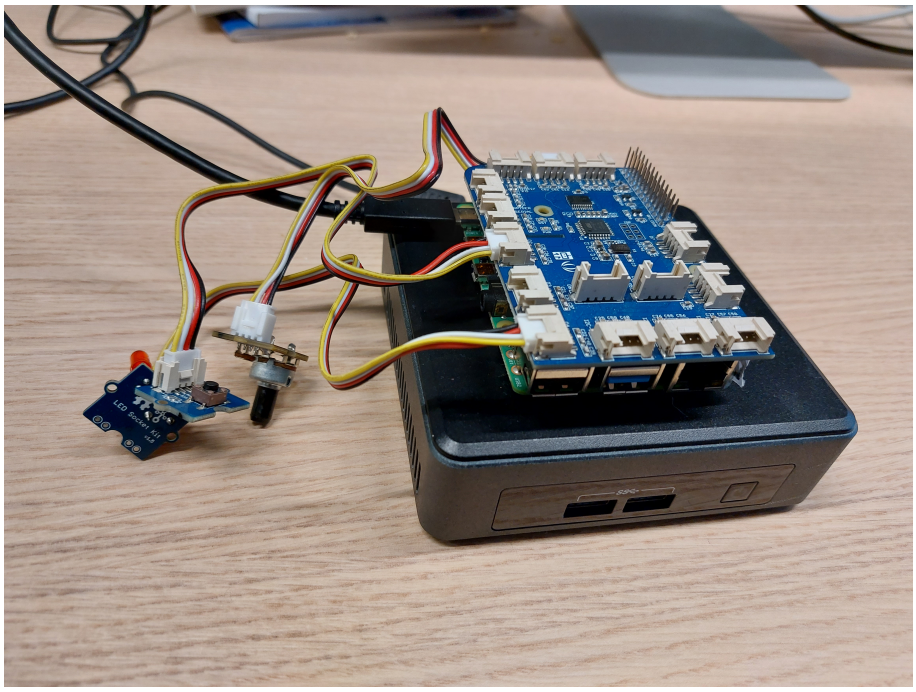


Figure 5.3: A RPI equipped with its hat and connected sensors. Below, a NUC hosts the Spotlight, user interface and general manager of the system. Both hardware devices are connected to the local network over WiFi.

The central component of the explanatory system, which hosts the user interface, the Spotlight and the general manager for the smart home control system, requires more power than RPIs. Notably, it can host additional system-wide simulation engines that require heavy computations; also, as it hosts the web interface, it has to handle modern front-end frameworks, adding overweight. In a realistic implementation this component can be a small central unit integrated to a fixed location within the house (e.g. within

the electric panel) or in a central control touchscreen tablet. In our implementation, we use an Intel NUC, which is a fully-fledged x86-based computer that is able to handle heavier operations. Figure 5.3 shows the relative size of a NUC compared to a RPI.

Software technologies

To store data, facilitate data transfer between controllers and track the evolution of the model system's state (see Section 5.2), it is necessary to rely on a database. Here, our main decision criteria are: i) the choice of a lightweight DataBase Management System (DBMS) that can be hosted on the RPIs; ii) existing adoption within IoT applications to be in line with existing technologies; iii) ease of use; iv) support of data streams. We settle to use a key-value based DBMS, as they are identified to be suited for cache-like operations (Kuzochkina, Shirokopetleva, and Dudar, 2018) which corresponds to our application: the database is designed to cache values recorded by the various components. Our final choice is to rely on Redis (Amghar, Cherdal, and Mouline, 2018), which is a in-memory DBMS; i.e. data are stored on RAM which results in fast operations. In addition, Redis provides native support for Raspberry Pi's and data-stream objects, which suits our needs to store sensor data from the control system. Redis' identified downside is a poor scalability (Amghar, Cherdal, and Mouline, 2018) but this poses no issue regarding the desired scope of our demonstrator.

Inter-device communication is operated using the standard TCP/IP framework, which is most frequently represented as a 5-layered protocol: application, transport, internet, link and physical layers, each layer encapsulating the previous one, guaranteeing communication integrity and routing to the right agent (Forouzan, 2002). We rely either on WiFi connectivity or on wired connection to the local network. Other protocols are common in IoT applications: Bluetooth Low Energy (BLE) (Gomez, Oller, and Paradells, 2012) and ZigBee (Safaric and Malaric, 2006) are two widely used ones, often used in low-power devices running on battery (Siekkinen et al., 2012). However, the universal use of TCP/IP, its native support by nearly all devices and computers favored its adoption in our demonstrator. At the application layer, we use two different communication technologies.

Between components of the explanatory system, communications are handled by using an implementation of the communication messages described in Table 4.1 over the MQTT protocol. MQTT is a widespread communication pub/sub protocol specially targeting IoT applications or other resource-limited devices (Hunkeler, Truong, and Stanford-Clark, 2008). MQTT uses a broker to route the messages between different connected clients, allowing the latter to subscribe and publish to defined communication channels. In our implementation, the broker is integrated to the NUC, which corresponds to its generic role as system manager. Each component of the explanatory system then subscribes to a channel corresponding to its name. Therefore, a LEC wishing to communicate to the Spotlight simply has to publish a message to the "`Spotlight`" channel. Note that MQTT can rely either on TCP/IP (which we use) or ZigBee or BLE, which covers most IoT and smart home applications.

On the contrary, communication between the explanatory system and the control system relies on a HTTP REST API. This implementation is coherent with the obser-

vation data flow implied by our layered organization depicted in Figure 4.1: the control system is observed, but does not interact with the explanatory system. As they are widely adopted for server-client applications, REST requests are complying with this approach. Further, REST allows to easily integrate system control into a web interface and allows for distant control and monitoring of the smart home model.

Whereas the explanatory system is implemented in Python to benefit from the modularity and the ease-of-use of this scripting language, the control system requires a more lightweight approach. As there is no clear consensus of programming language for smart home systems (Mekuria et al., 2019), we settled on using a Golang-written event-driven framework named Flogo (Software, 2020). This ultra lightweight software handles various triggers (REST requests, periodic events) and executes small re-usable pieces of code called "actions". An application can be defined as a collection of triggers and actions linked together. Its light weight and re-usability are targeted specifically towards IoT applications (Valtolina et al., 2019).

Table 5.1 summarizes the different technologies used in the successive demonstrator versions, alongside the goals defined for each version.

	Version 1	Version 2	Version 3
SHC Language	Java (iCasa)	Java (iCasa)	Go (Flogo)
Physical environment	iCasa	iCasa	Home model
Hardware	PC	PC	NUC and RPIs
Explanatory System Language	Java	Python	Python
External Communication	CLI	REST API	REST API and CLI
Internal Communication	None	None	MQTT
Goals	Decentralized LECs D-CAS prototype	Genericity Self-adaptation	Privacy Twin-memory LEC Realistic hardware

Table 5.1: Comparison between objectives and characteristics of the three consecutive demonstrator versions.

5.2 Description of the demonstrator

The demonstrator is based on a Smart Home model that was originally built for a previous PhD thesis conducted at Télécom Paris (Frey, 2013). This smart home model is composed of several rooms that are visible on Figure 5.4. Each room is equipped with various hardware sensors and hosts the devices described in Table 5.2. As presented, these devices all correspond to Flogo application that is run in a designated thread and hosted on a RPI. Given the number of available RPIs, we settled to use one RPI for each room of the home. This corresponds to a realistic smart home architecture: low-power edge computing units hosting one or several components (Hadwan and Reddy, 2016).

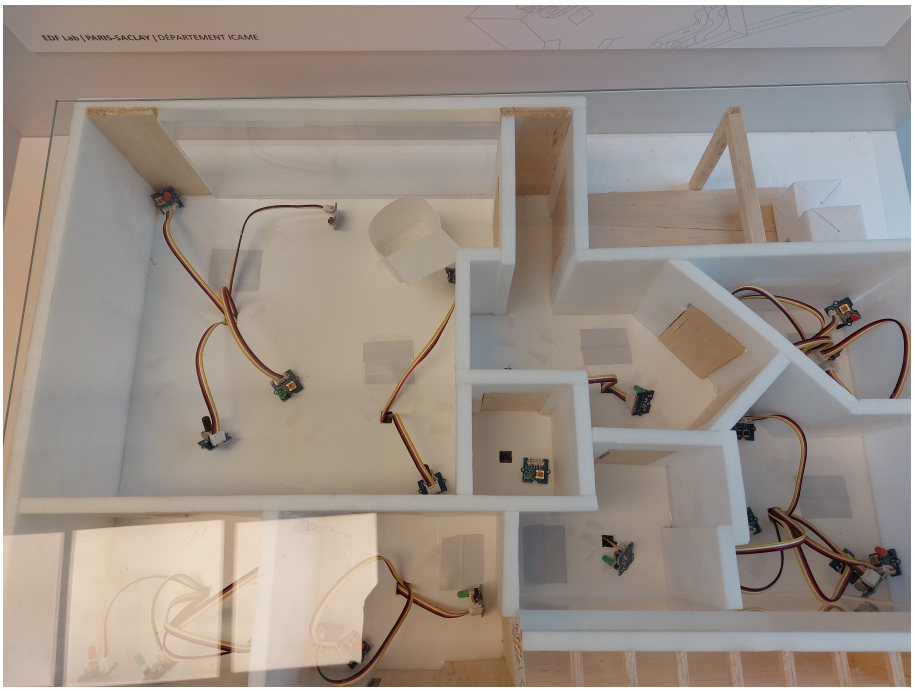


Figure 5.4: The smart home model, built upon the basis originally from (Frey, 2013). The NUC and RPIs are located beneath the model, wired sensors and actuators are visible in several equipped rooms. Other rooms only have a LED indicating the presence of the user.

The variety of implemented devices allows to model some interesting interactions between goals, actions and devices: the presence of the user can modify the temperature target set for the temperature controller, the light, the CO_2 concentration in the room. Each device Flogo application gathers data and writes its output into a designated stream on the Redis database hosted on the RPI. In this implementation, the Redis database is used to implement the "Obs" interface specified in our architectural design (see Figure 4.3): it provides a convenient way to standardize the access to the data. One local control manager runs on each of the RPI and handles requests for device addition, deletion or modification. The manager is written in Flogo, too. In addition, it also handles a designated field on the local Redis database. This field can be accessed by other SHCs, thus providing basic context-aware capabilities for the components (P. Lalanda, Gerber-Gaillard, and Chollet, 2017). This makes the local control manager observable, so it is possible to pair a LEC to it, to integrate its knowledge into the explanatory system).

The resulting general implementation is shown in Figure 5.5. On the same RPI hardware, several SHCs are running alongside their attached LEC, following the architecture principles from Chapter 4. Here, the LECs subscribe to stream entries on the Redis database, which are where the SHCs write their values. In case an update occurs in a SHC, the information flow is as follows: the local control manager exposes the new characteristics of the component, which are observed by the general manager located on the NUC. Then, the Spotlight observes the change, in particular if the LEC-related field

Type	Hardware	Description
External Thermometer	Potentiometer	Sets the outdoor temperature using the potentiometer
Internal Thermometer	None	Measures the temperature computed for the room
Light	LED (button)	Represents a light Can be manipulated either via the button or a presence sensor
Temperature Controller	Potentiometer	Selects a target temperature for the room and controls the heaters
Presence Sensor	LED	Lights up if a person is in the room
Window	LED (button)	Operated either via a controller or the button
CO_2 Sensor	None	Measures the CO_2 concentration computed for the room
CO_2 Controller	None	Can open the window if CO_2 concentration is too high

Table 5.2: The different types of devices implemented in the model house. Each device corresponds to a Flogo application hosted on the RPI corresponding to its room.

is modified. When needed, it then transmits a POST request to the local LEC manager, which installs, removes or updates the necessary modules on the affected LECs.

The NUC hosts the Spotlight, along with the global manager for the smart home control system. This latter exposes a rest API that can be used by other components to access to the current organization of the control system: RPIs' IP addresses, hosted devices and their characteristics. In addition, the NUC hosts the MQTT broker and a Redis database, in which the Spotlight registers additions and changes of configuration, which allows, as previously mentioned, to pair up a designated LEC to analyze these readings and integrate them into the explanatory rationale. The NUC also hosts the web server providing the GUI. Screenshots from this interface are shown in Figure 5.6.

While using physical devices such as light sensors, potentiometers, thermometers, the scale of the model and the desired evolution speed make it unpractical to rely on physical variables only: for instance, thermal phenomena are hard to replicate at this scale. To cope with this, we instead rely on computed variables from a physical model. Thus, room temperatures, CO_2 concentration and user presence are computed room-wise: a specific script runs on the NUC to compute each of these variables for each room of the model, and writes the variables into the room's RPI local database. Each variable can then be read by the various hosted devices. For instance, the evolution of the temperature T_k of each room is defined by a heat equations which takes into account heat transfer with

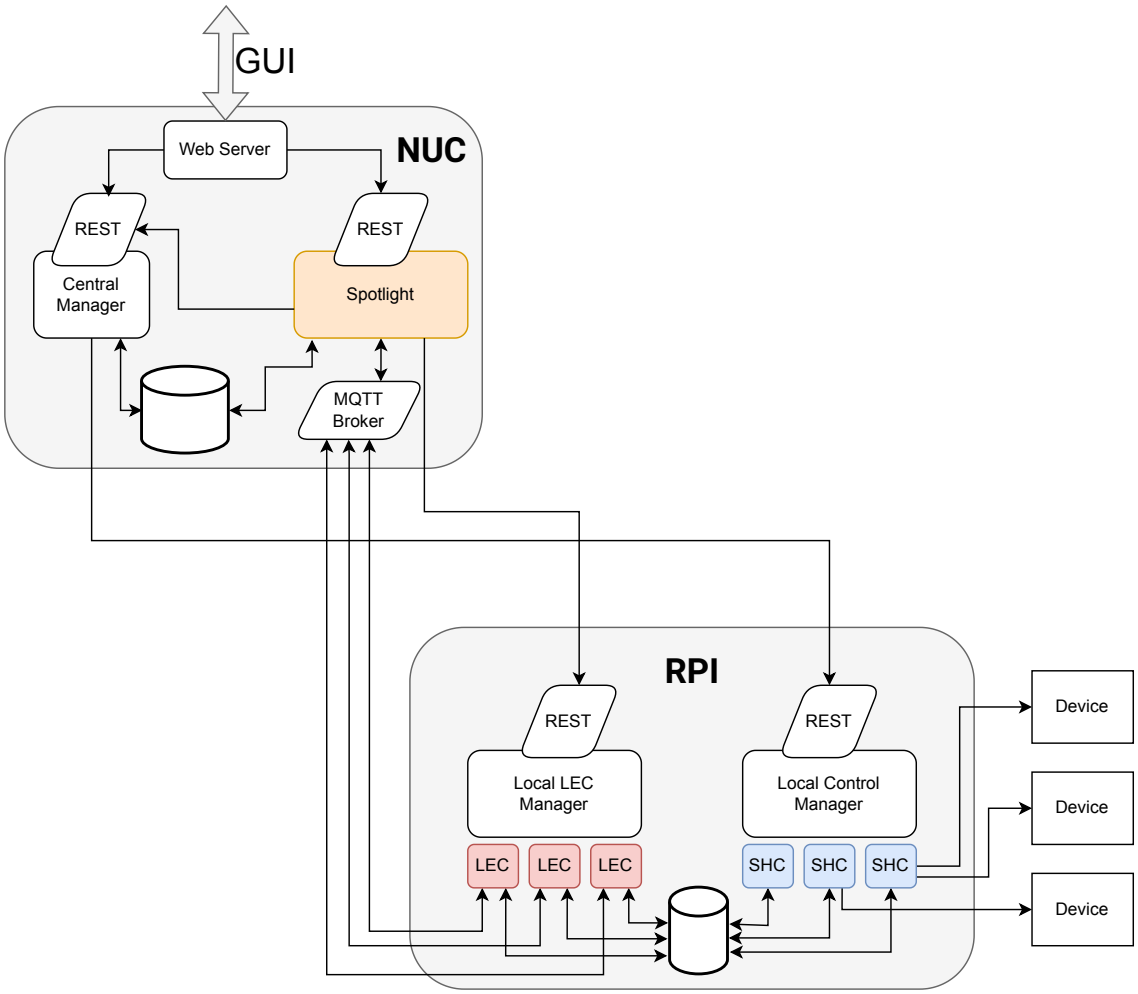
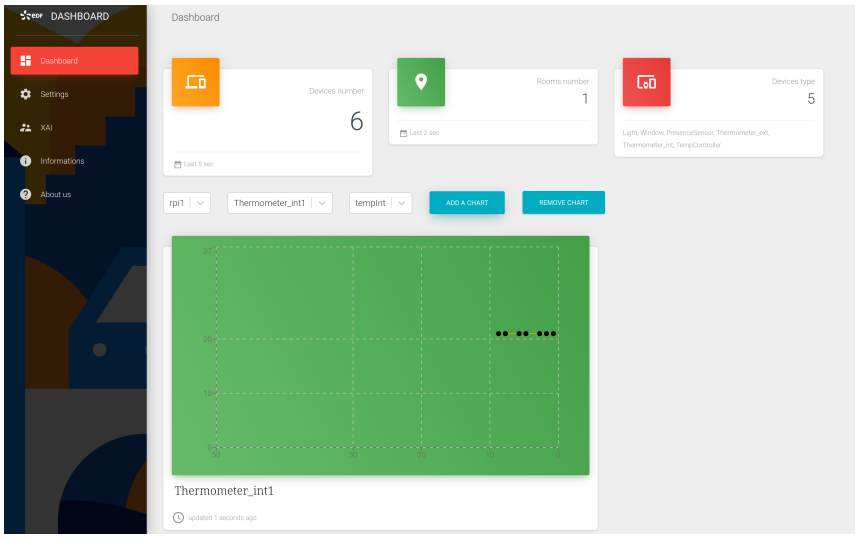
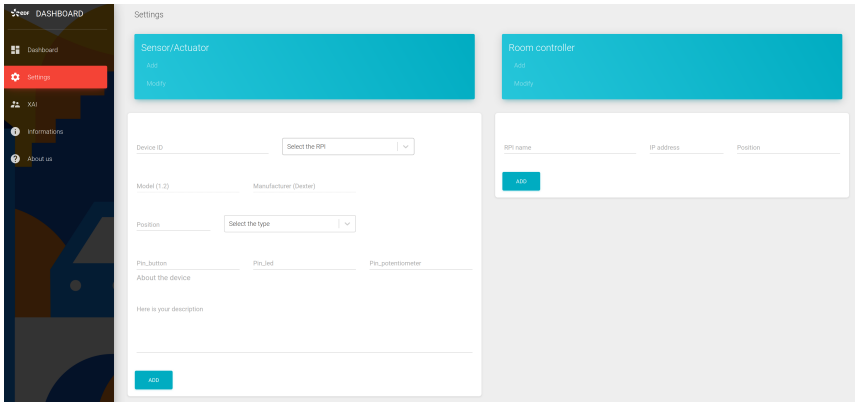


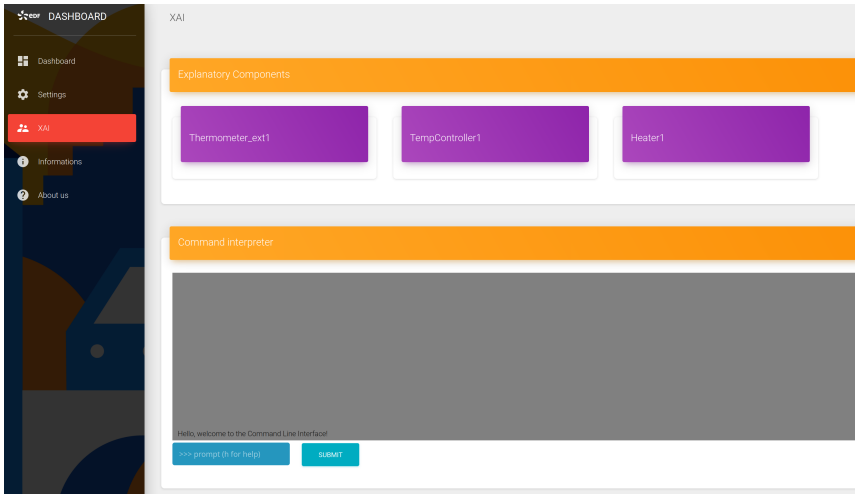
Figure 5.5: Implementation overview. The NUC hosts the Spotlight and the general manager of the control system, both presenting a REST interface used for observation. It also hosts a database to store past and current general system state, the MQTT broker for explanatory system communications and the web server providing the GUI. A RPI hosts a local LEC manager that presents a REST interface: this allows the Spotlight to command the creation of new LECs. It also hosts the local control manager that supervises the integration and configuration of several SHCs: these latter can be connected to hardware devices (buttons, potentiometers, LED's), and store their measures onto a local database.



(a)



(b)



(c)

Figure 5.6: Three views of the GUI application to monitor and control the demonstrator. (a) presents an overview of the control system components, and a plot of measures from a selected component. (b) is the interface to add or modify existing SHCs. (c) offers an overview of the LECs present in the system, their modules and recorded events, and the CLI to interact with the Spotlight, notably to start an explanatory rationale.

neighboring rooms, outdoor temperature and active heaters in the room:

$$T_k^{t+1} = K_1 \left(\sum_{h \in \text{heaters}} Q_h - k_2(T_k^t - T_{out}^t) - k_3 \sum_{k' \in N(k)} (T_k^t - T_{k'}^t) \right) . \quad (5.1)$$

Note that coefficient k_2, which states how much heat is lost to the outside, can vary, depending on whether a window is open in the room or not. The CO_2 concentration computation is simpler: it is increased by a fixed value at every iteration when a user is present and decreased when one or several windows are open. For clarity, the physical model script is not displayed on Figure 5.5.

5.3 D-CAS as a tree algorithm

D-CAS's design revolves around propagating conflicts and exposing the process as a potential explanation for the problematic situation. The visualization of the output is a key factor in the perception of the explanation. Visual explanations have been covered by some studies in XAI and presented as an approach to explanation on its own in our review in Chapter 2. It follows that sometimes visualization alone can help comprehension of a model (Maaten and G. Hinton, 2008). However, the problem here is different, in the sense that our objective is to find an adequate representation of a rationale rather than a model.

The topic of knowledge representation has been addressed in different disciplines. *Mind maps* are visualization tools that rely on associations and hierarchical organization of concepts to represent knowledge (T. Buzan and B. Buzan, 2006): from general concepts, specifications and related examples are presented in linked nodes. However, mind maps focus on representing related ideas rather than stories or reasoning. (Tsilionis et al., 2021) propose a standardized representation of users' stories that can be found in agile methods. This approach allows to identify similarities between stories. This technique originally avoids redundancy and helps the development process by improving the analysis of possible use cases. It uses a tree-based representation with different branches spanning for each identified sub-problem from an original issue. Similar graph representations are common when addressing causality: causal models are commonly represented using graphs (Peters, Janzing, and Schölkopf, 2017; Pearl, 2009; Akleman et al., 2015).

D-CAS complies with this tree representation, as a root issue, that is the starting point of the rationale, is propagated onto sub-problems via abduction and simulation, resulting in different branches. Our implementation of D-CAS in the Spotlight directly features this construct. Figure 5.7 shows the object descriptions of the ExplanationTree: each main element of D-CAS is associated with a Node type, which contains relevant information regarding the operation, such as the component that performed it, the examined conflict, etc. The general ExplanationTree object encapsulates the G and C sets from D-CAS which store the given-ups and the examined conflicts, respectively. Operations on these sets are then handled directly by the *append()* method of the tree: as a node is added into the rationale, its conflict (or give-up) is added to the

corresponding set. A pointer keeps track of the currently active node, which represents the part of the rationale that is currently examined (i.e. the ongoing conflict) by D-CAS; subsequent nodes are appended to this active node. Contrary to the rationale presented in the theoretical D-CAS algorithm in Chapter 3, we transfer here the entirety of the rationale tree to the LECs when they are requested to examine a conflict. As trees encapsulate information about considered and given-up conflicts, this prevents the need for further communication between the LECs and the Spotlight (so as not to propose already discarded hypotheses, for instance).

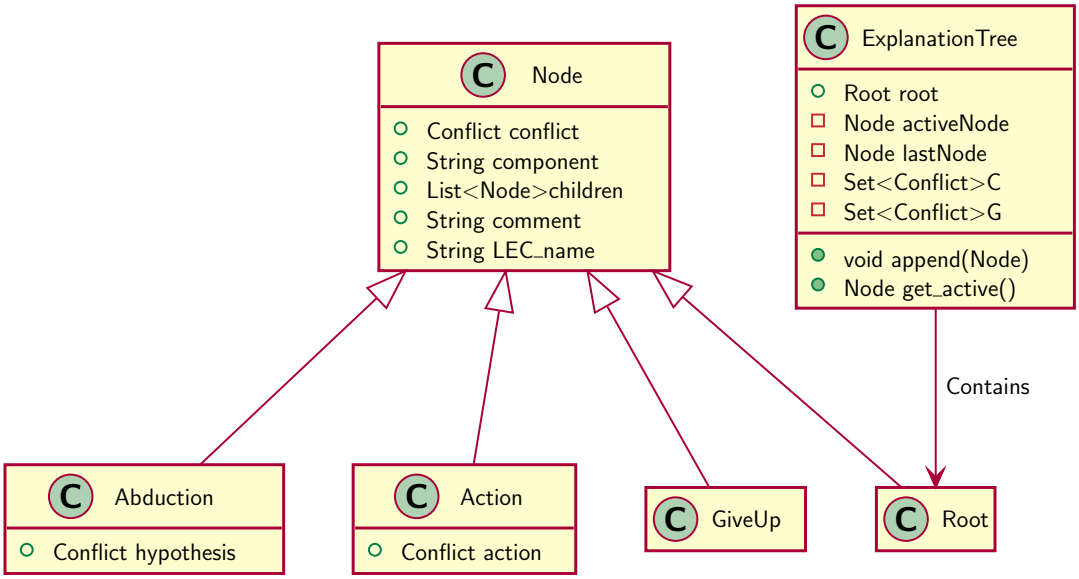


Figure 5.7: Class diagram of the ExplanationTree object handling D-CAS representation and rationale.

The implementation of D-CAS used in the demonstrator allows to interact with the rationale, gaining benefits from its step-by-step approach: since the state of D-CAS is contained in the sets G and C, it is possible, at every given step, to stop the current exploration, manipulate the current examinations and beliefs of the explanatory system, and resume the rationale with this new setup. The tree representation of the algorithm makes this process easier, as it allows to isolate and identify each step of the process with the ID of the corresponding node. As a result, a new version of D-CAS is presented in Algorithm 3. This version appears more concise as the $append$ method of the tree object contains most of the logic for the explanation: it appends a node to the tree and then, depending on the new node's type, it adds its conflict to the set of given-ups. Similarly, the $get\_active()$ method returns the active node of the tree, to which will be appended the next node.

At line 12 of Algorithm 3, the **broadcast** method is responsible for sending post-hoc feedback to all components. The tree object encapsulates the entirety of the reasoning so far: the conflicts that have been considered, the conflicts that have been abandoned, how which component handled which request and whether its handling was correct. Transmitting this information to all elements of the explanatory system enables the

Algorithm 3: The tree version of D-CAS.

Input:
Result: A Tree object containing the rationale
Data: Pointers to LECs in the system

```
tree ← Tree(root_conflict = (P, N));
while tree.get_active() ≠ tree.root do
    current_conflict ← tree.get_active().conflict;
    responsible ← locate(current_conflict);
    response_node = responsible.investigate(tree);
    tree.append(response_node);
    if type(response_node) = ActionNode then
        simulator.run(response_node);
        waitForConsequences();
    end
end
broadcast(tree);
return tree;
```

modules of the LECs to improve their parameters and organization for better performance in future requests.

The final output of the D-CAS method is the tree object which can subsequently be displayed and interacted with in the GUI. In the current version of the demonstrator, the visualization tools are made using the `React D3 Tree`[1] library.

5.4 Illustrative examples

The demonstrator's purpose is to provide examples to better understand the behavior of D-CAS in various situations. It is also to test its ability to generate an rationale that complies with our defined goals in simple yet realistic situations. These situations aim at being representative of typical use cases where questions regarding the system's behavior may arise. These examples remain theoretical, in that they have been purposefully designed to illustrate distinct situations and abilities of D-CAS.

All of the scenarios are possible continuations of Example 1.1 exposed in the introduction to this thesis. It represents the typical situation of a user coming back from work and finding him/herself into a cold room, which is surprising as the temperature control system is supposedly regulating it. As the motivation for this scenario, we argued in Chapter 1 that this setup can be the result of various causes, hence making the explainability of the system an important feature as it helps identifying the cause and possibly fixing the issue.

The experimental setup described in this chapter allows to model a situation where a window is open; or a faulty thermometer reports erroneous measures; or the temperature

[1] `https://github.com/bkrem/react-d3-tree`

setting is inadequate. These different scenarios which all share the common starting point of the room being surprisingly (or annoyingly) cold. This base conflict is encoded as $(\texttt{cold}(\texttt{room}), -30)$, the -30 necessity being arbitrarily set as the default value for an annoyance question from the user. In the following scenarios, we examine how D-CAS can unravel the situation and handle potential difficulties.

Readers may note that these examples are not meant to be limiting. Their simplicity comes from implementation limitations. They mostly involve simple causal relations that can be understood by human users without machine help. Notably, they do not correspond to situations where the AI is at fault for a situation (e.g. following an erroneous classification from a black box model). However, the presented examples mainly serve as proof-of-concept examples to demonstrate the ability of D-CAS to generate a system-wide reasoning without having any requirement regarding the nature of knowledge (rules, statistical models, etc.). Furthermore, we may also argue that the explanatory role of an AI in the context of a smart home would not be limited to its own decisions, but rather help identifying other interactions, such as an open window causing the room temperature to be lower than expected.

The implemented modules (abduction, interpretation and simulation) are kept as simple as possible throughout these scenarios: the goal here is to demonstrate the different possibilities offered by D-CAS rather than proposing advanced time series analysis or causal abduction methods. As previously stated, this is considered out of the scope of this research, as it would be worth another independent research. For this reason, all interpretation modules are designed as threshold-based event detectors and abduction modules rely on predefined rules corresponding to our intuitive listing of possible causes. E.g. the LEC attached to the temperature controller lists open windows, outdoor cold and heater being turned off as possible causes for indoor cold. Another possibility for abduction inference is to consider "memorable" events as possible causes: this will be studied in Chapter 6.

For all tree rationales displayed in this section, an interactive version is available online on our website `explainableai.fr`. This interactive display allows to present each tree step by step and, for each step, shows the sets of examined and given-up conflicts and the comment embedded in the tree's active node.

An open window The first, and arguably simplest case, is to identify a causal relation such as an open window causing the room to be cold. Such knowledge can be easily encoded into an abduction module on the temperature controller as a rule in the form

$$\texttt{open}(\texttt{w}) \wedge \texttt{type}(\texttt{w}, \texttt{Window}) \wedge \texttt{location}(\texttt{w}, \texttt{room}) \implies \texttt{cold}(\texttt{room})\,. \tag{5.2}$$

Using this inference mechanism, the temperature controller can propose the open window as a cause for the cold, without any further knowledge of the situation (since the temperature controller does not have access to the state of the window). The proposed hypothesis is sent back to the Spotlight, which then investigates the LEC attached to the window. This latter confirms that the window is open, and identifies a possible action to revert this state: $close(window)$. Here, however, the simulation module of the LEC refuses to perform the action: its intensity is not high enough to "bother" computing the

output. Therefore, the conflict is finally given up, resulting in the rationale presented in Figure 5.8.

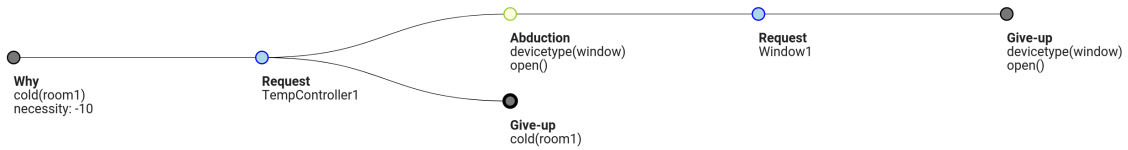


Figure 5.8: D-CAS output for the simple situation of the explanation: "It is cold in the room because a window is open"

Going deeper Following the previous example, the user wants more information about why was the window open, in the first place. He/she has two possibilities: either ask the system directly why the window is open, considering it as the starting conflict, or repeating the same question on the room's temperature, with a higher intensity. In both cases, the intensity will now be high enough to trigger the simulation of the closing of the window. This simulation first confirms that the hypothesis is correct, as the simulated room's temperature rises back. Second, it shows a possible reason why the window is open: as the simulation goes on, the CO_2 concentration becomes too high. This was an existing goal of the control system, therefore a conflict $(\texttt{high\_co}_2(\texttt{room}), -20)$ is raised by the LEC attached to the CO_2 controller. This conflict is examined by D-CAS, but its intensity of 20 is not enough to revise the previously examined window. Hence, as no other solution is found, it is given up. The resulting rationale is shown in Figure 5.9 and illustrates how forward inference using simulation enables to propose goals as causes for actions of the control system.

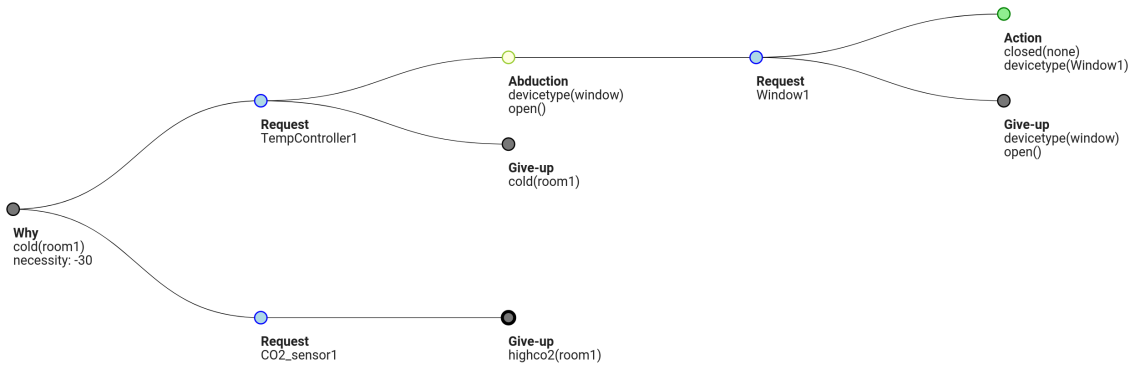


Figure 5.9: The tree rationale for the in-depth inquiry of the window hypothesis: a simulation is run to validate the inferred hypothesis and to find a cause for the window being open. Here, the corresponding explanation rationale is the following: "It is cold in room1 because the Window1 is open. It was open because closing it would provoke the CO_2 concentration to rise too high."

Handling erroneous hypotheses Abductive inference is by nature an inference process that is prone to mistakes. D-CAS offers a robust framework that can handle such mistakes. Abiding to our goal of transparency, such mistakes are not hidden in D-CAS, rather, they are explicitly displayed to the user, since they can provide informative feedback to understand the rationale of the system. Mistake handling in D-CAS is straightforward: in the process, each LEC remembers on which conflict it had been called. In the event the same LEC is asked a second time about the same conflict, it can analyze what its previous proposal was and see the effect on the rationale tree (using information embedded in nodes). For instance, it can see if its previously proposed hypothesis was not confirmed by other components, or if a simulation of its counterfactual proved unsuccessful. This reflection is made easier by our tree implementation, where the entirety of the rationale topology is transmitted to LECs. At this step, the LEC can operate on its modules to incorporate the new knowledge and improve their future behavior.

In Figure 5.10, we see an example of an erroneous abduction. Here, the implemented module of the Temperature Controller LEC proposed two hypotheses for the room being cold: the heater being turned off or the window being open. They are associated with a score that translates the frequency these causes have been observed by the manufacturer. Initially, the preferred cause is the heater being turned off. However, here, the proposed hypothesis cannot be confirmed by the observation from the heater LEC. Therefore, the hypothesis is given up, with metadata in the corresponding node indicating that this give-up was due to the conflict not being observed. Following this dead end, the Temperature Controller's LEC can examine the next possible hypothesis, the window being open. In addition, the knowledge of the rationale tree and embedded data is used by the LEC to update its modules: here, the scores associated with the two possible causes are revised to include this knowledge. In future calls, the window may be preferred to explain the cold in the room.

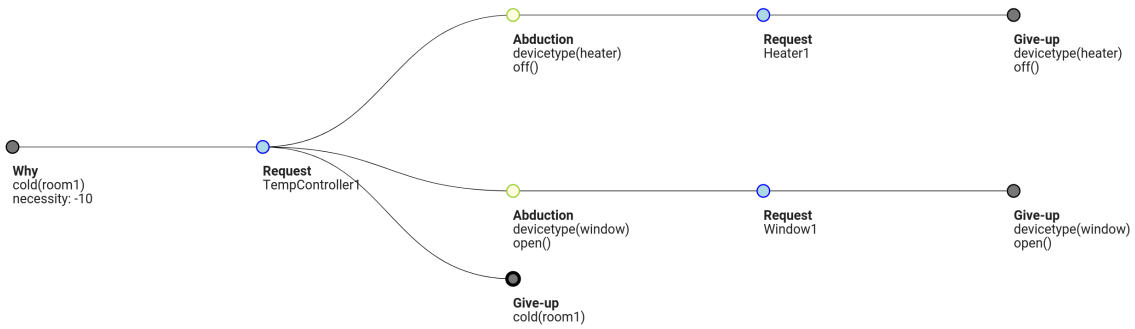


Figure 5.10: An example of how D-CAS handles the mistake of a component. An erroneous hypothesis is eventually given up while the initial conflict still persists, which leads the LEC to understand its mistake and modify accordingly its inference knowledge. Here, the global rationale may be: "It is cold in the room. The component proposed that the heater was turned off, but the Heater LEC did not confirm it. The Temperature LEC proposes another cause: some window is open. The window component confirmed the cause, but could not explore the conflict further."

Changing interpretation A typical case for the integration of self-adaptation into an explanatory rationale is when an adaptation change can be proposed as the cause for a strange phenomenon. For instance, consider the revision of a predicate's meaning, as this is allowed in our architecture. Here, the revision of the meaning can be exposed as a reason for the anomaly. If the definition for the word `cold` had to be revised, maybe this revision is the cause for the perception of the cold, since the system was wrongly understanding the temperature as being correct?

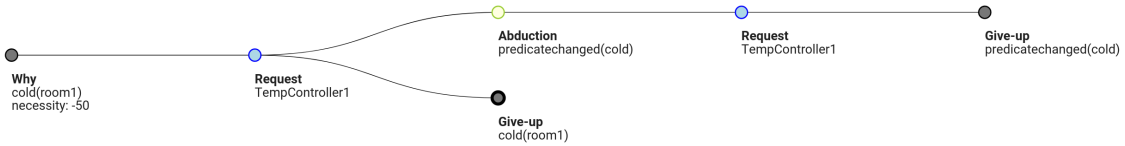


Figure 5.11: Since the interpretation of the world's events does not coincide with the upcoming request, the LEC attached to the temperature controller is able to revise its interpretation by increasing the threshold of the `cold` predicate.

In Figure 5.11, the exposed rationale follows this situation: the user felt cold in the room, but the LEC failed to associate any existing event with the predicates contained in the request. However, given the intensity of the request, the interpretation module in the LEC that is responsible for the understanding of cold observes that it simply has to change the predicate's threshold by a small amount to match the user's observation. The predicate's meaning is therefore modified, which is observed and recorded as a separate event in the LECs memory. An abduction module of the LEC then proposes that this change might be the reason for the room being cold. Therefore, the resulting output from D-CAS is that it is cold in the room because the definition of cold had to be changed to match the user's perception of the season.

Discovering a failure The last example we present in this chapter highlights a capability that is enabled by knowledge revision: discovering failures. Consider the rationale presented in Figure 5.12: the LEC attached to the temperature controller, within its abductive knowledge, has a rule stating that:

$$\texttt{failure(h)} \wedge \texttt{device\_type(h, heater)} \wedge \texttt{location(h, room)} \implies \texttt{cold(room)}\,, \quad (5.3)$$

which indicates that it is possible that a heater failure be responsible for the cold. Using this knowledge, the component designates this proposition as being the potential cause, and the Spotlight forwards the request to the LEC attached to the heater. The latter receives the instruction to examine the predicate `failure` which, however, does not corresponds to its current observations: the heater controller is not reporting any failure state.

However, depending on the incoming intensity, the LEC may revise its knowledge of failure: maybe some readings from the SHC can be interpreted as a failure? In case the intensity is high enough to modify its interpretation of failure, the LEC considers that its current observations are compatible with this state, and therefore validates the cause. The entire process is done without requiring the SHC to effectively report a failure: the

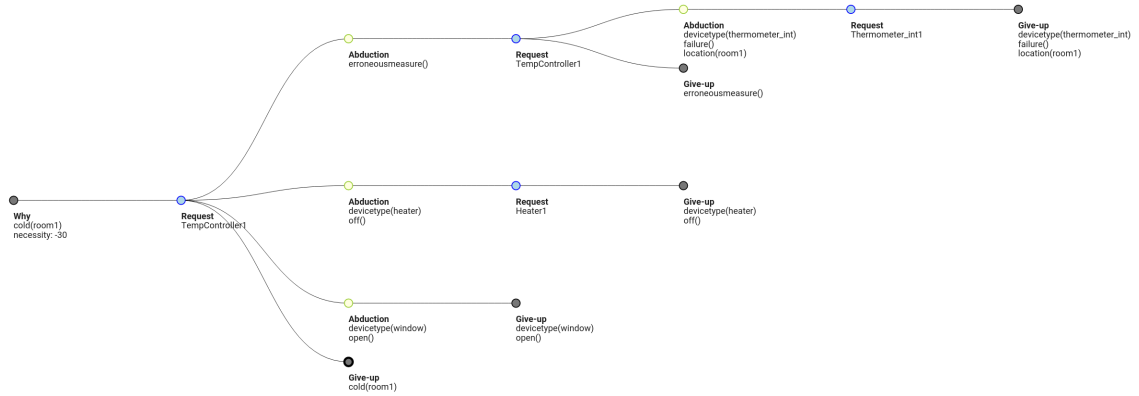


Figure 5.12: In this example, the rationale proposed by D-CAS finds a failure that had not been monitored by the control system: belief revision allows the Temperature Controller LEC to propose that its observation is erroneous, which is then traced back to a possible failure of the indoor thermometer (upper branch). Since no simulation nor abduction is then proposed by the thermometer LEC, D-CAS then explores the other possible hypotheses: the heater being turned off (middle branch) and the window being open (bottom branch).

LEC "guessed" that a failure occurred by considering the request from the spotlight and comparing its intensity with the estimated cost of accepting a failure. This particular case of knowledge revision shows the potential offered by D-CAS, which isolates observations from interpretations: it can identify otherwise undetected failures and integrate them into the explanatory rationale.

The rationale presented in Figure 5.12 is more complex: after proposing the failure of the thermometer as a first abduction hypothesis, the system is unable to prove that this cause is the correct one: here, no implemented module is able to simulate the consequences of a non-failure state, or the possible causes of the failure. Hence, this lead is given up. The rationale then goes on to two other possible causes that are discarded: the heater is proposed, but, as it is turned on, the conflict cannot be confirmed by the specialized LEC. Then, the window is proposed, but, this time, no window LEC has been placed in the system. Exploration of this branch thus stops for lack of specialized component to confirm or discard the proposition. Overall, the text generated by this rationale could be: *"It is cold because the measure may be erroneous. This may come from a failure in the thermometer. However, Thermometer_int1 LEC cannot be confirmed by simulation or further propagation. Another possible cause for cold(room1) is the heater being turned off. This cause is discarded because Heater LEC does not confirm the conflict. Another possible cause for cold(room1) is the window being open. This cause is discarded because no component can explore it. No other solution is found, giving up on the conflict."* This example illustrates the transparency and shallowness principles of the explanation: each tentative cause is presented, alongside its responsible component, and the entire reasoning is exposed, rather than merely the deepest cause.

Overall, the implementation presented in this chapter demonstrates the possibility of using D-CAS and the architecture described in Chapter 4 in a realistic setup. The

different examples show the complexity of the task, with numerous explanation variations emerging from the same initial situation. In all of these situations, the same D-CAS principles enable the development of a reasoning that can be visually presented to the user, detailing the causal chain and relevant components. However, the performance of this demonstrator is limited to the basic implementations of the specialized abduction, interpretation and simulation modules. The topic of abduction, which is essential for the explanatory reasoning, is particularly challenging. In the next chapter, we describe a method allowing the discovery of new hypotheses in abnormal situations, when other classical statistics-based methods may fail, due to the lack of previous data.

Chapter 6

Hypotheses for abductive inference

Summary

The question of abductive inference has been identified as one of the key mechanisms in explanation and is present as such in D-CAS, alongside simulation of counterfactual. However, while many simulation solutions can be realistically integrated into our explanatory system and handle integration of new devices, abduction remains a hard topic that we have not yet covered.

While existing XAI methods can be perceived as possibilities for abductive inference, e.g. using feature relevance technique such as LIME and propose the most relevant parameters as causal hypotheses, their use relies on past data. This approach is not suited for the cases where explanations are most useful, i.e. unusual and surprising situations that D-CAS aims at solving.

To tackle this challenge, we propose a novel approach to abduction. Given that D-CAS is robust to erroneous abductions, as it can give-up a lead if it proves unsuccessful, we propose an approach that, similar to humans, is able to propose hypotheses in situations where no minimal prior knowledge is available. This approach is based on a measure of memorability of events and proposes the most memorable event as a hypothesis for a situation. We've tested this approach on examples from the smart home context to illustrate how this measure fares in comparison with human understanding of memorability.

This chapter reproduces most of the content and results of a journal article (É. Houzé, Dessalles, et al., 2022).

6.1 Definition of the problem

6.1.1 "Naive" abductive inference

Abductive inference proposes to consider true causal hypotheses to an observed phenomenon. This reasoning tool has been identified as the major role in scientific methodology (Popper, 1963), where a researcher formulates hypotheses using abduction then tests them via experimentation. However, contrary to deduction, that is the inference model used in mathematics and formal reasoning, abduction is not sound and can be complex. (Magnani, 2011; Pearl and Mackenzie, 2018; T. Miller, 2018) identify that abduction is the most common inference in mundane life: humans often infer causes for their observations. However, abduction is also complex to understand, as it can come in various ways.

To illustrate this phenomenon, consider Example 6.1. The user is confronted with a surprising event. He/she uses abductive inference to hypothesize that the new TV is the cause of the strange behavior observed from the rest of the equipment. Depending on the information available to the user, this inference process can originate from three possible ways. First, if the user had read the manual of the TV or the equipment, he/she may have come across indications of the possible effects of switching on the TV: this knowledge is similar to having a rule-based model (or any other model) of the phenomenon and using backward chaining to identify possible causes (Leake, 1995). Alternatively, if the user previously observed similar behavior (e.g. in a store, at a friend's house), he/she might have inferred from this past observations that the cause may be the TV: here, this is similar to case-based reasoning for abductive inference (Sun, Finnie, and K. Weber, 2005). Finally, it may occur that the user had neither model knowledge nor previous observations for this phenomenon. However, he/she is still able to perform the abductive inference and identify the TV as the cause. In this context, the user relies on the observation that the new TV is *memorable*, as it is the first time it is switched on. As a consequence, it appear logical for him/her to propose it as the cause for a posterior memorable event (the dimming of the lights).

Example 6.1

A smart home user is sitting in his/her room, which is equipped with connected devices such as a smart speaker, light bulbs, presence sensors, blinds, etc. He/she switches on the brand new smart TV that he/she has just installed as a replacement to an old one. As the TV is switched on, the lights in the room dim and the window blinds roll down. The user quickly understands that the TV caused this reaction from the room equipment.

While the two first methods rely on previous knowledge specific to the encountered phenomenon, the third one only requires common sense to distinguish memorable events from "background noise". It then proposes something memorable as a possible cause: this kind of reasoning is what we call a "naive" abductive inference, in the sense that it requires no specific knowledge of the system. With this characteristic, naive abduction appears

suited to propose possible hypotheses when face with completely new, unexpected or mysterious phenomena.

6.1.2 The difficulty of defining memorability

Naive abduction, as we have seen, relies on a sense of memorability to select which events are most prone to be relevant hypotheses. Having a theoretical and computable measure for memorability would therefore open up new possibilities for abduction, especially in situations where other methods fail to provide insight. In addition, being able to compute a memorability score for recorded events can be useful in itself: for instance, a system may be able to summarize what happened in the house during the owner's absence by selecting only the most memorable events; or it could be used as a criterion to discard most events and avoid overloading the limited resources of embedded devices with non memorable events.

However, defining memorability is far from obvious, given the wide range of events to consider: how can one decide whether a temperature measure is more or less memorable than a light measurement or a movement? Given the number of devices to consider, many events of different natures, units and magnitudes are recorded by a smart home control system. This first difficulty is inherent to the heterogeneous nature of the data collected. However, the common sense of memorability is that record-breaking events are somehow more remarkable than average ones: for instance, the hottest day in the year is more remarkable than the average autumn day.

The second difficulty is that, to be useful in the context of "naive abduction", the measure of memorability should not rely on any previous knowledge regarding the events that are analyzed. For instance, no prior magnitude nor common values are known to the agent. The only available information is the memory of past events, with no existing order. By using the definition of events from Equation 4.1, an event is defined as a 3-tuple $E = (t, l, X)$. The knowledge of the state of the system available to the agent is a *memory* $\mathcal{M}$ that corresponds to an unordered set of recorded events.

6.2 Formalizing memorability

In our general endeavor of bridging sensor data to human perception of the world, we have relied on *events* and *predicates*. We will therefore re-use these notions here, as to provide a formal canvas for describing what happens in the house, what is measured, and how it can be deemed memorable or not.

For the sake of clarity and simplicity, for the rest of this chapter, we use the proxy that an integer n can be encoded using $L(n) = \log_2(n)$ bits. This approximation does not change the dominant term of the expression, which is $\log_2(n)$. In case of prefix-free encoding, the exact length is $L(n) = \lfloor \log_2(n) \rfloor + 2\lfloor \log_2(\lfloor \log_2(n) \rfloor + 1) \rfloor + 1$ (Elias, 1975).

6.2.1 Notions of Algorithmic Information Theory

Complexity as the length of the shortest program Many objects can be represented by using a finite binary string: think of the binary representation of numbers, binary encoding of Unicode characters or, more generally, digital representations of images, songs, videos, 3D models, etc. Therefore, we can restrict ourselves to finite binary strings $x \in \{0,1\}^*$ without loss of generality. How can one estimate the complexity of a string x? Length is not sufficient: consider $x_1 = 00000000$ and $x_2 = 10110001$. While both are 8-bit strings, x_1 can be considered simpler than x_2 as it is merely a repetition of 8 zeroes. Is there a method that could be used to compute a measure of this complexity?

Algorithmic Information Theory (AIT) studies the quantity of information contained in such objects. By contrast with the usual notion from Information Theory, such as Shannon's entropy (Shannon, 1948), the prime difference is that AIT studies objects, while Information Theory focuses on processes: as such Shannon's entropy is defined for random variables, while AIT provides definitions of information for any object. AIT originated with the concomitant discovery of what is now known as Kolmogorov Complexity by Solomonoff (Solomonoff, 1964), Kolmogorov (Kolmogorov, 1965) and Chaitin (Chaitin, 1966). This notion of complexity is distinct from time or space complexity measures that are often used in Computer Science.

Rather, Kolmogorov Complexity focuses on the quantity of information one would need to generate a binary string x. Formally, we consider a Universal Turing Machine U, i.e. an abstract computer-like machine that can perform any possible computation, given some input program p. Given such a machine U, complexity of a string x can be defined as the length of the shortest program that would result in x if used in the machine U.

$$K_U(s) = \min\{L(p) | U(p) = s\}\,. \tag{6.1}$$

To illustrate this definition, consider the string $x_1 = 00000000$ and $x_2 = 10110001$ that we have already mentioned. We can use a modern programming language, such as Python, to represent a Universal Turing Machine in a more readable way[1]. In Python, the shortest program that produces x_1 is $p_1 =$ "`return`$(8 * \text{`0'})$". On the other hand, the shortest program $p_2$ outputting $x_2$ is "`return`("`10110001`")": there is no more concise manner to write this string, in Python. Hence, using Equation 6.1, the complexity of each string is equal to the length of these shortest programs p_1 and p_2. $K_{Python}(x_1) = L(p_1) = 13$ characters and $K_{Python}(x_2) = L(p_2) = 18$ characters. Using this metrics, x_1 is simpler than x_2.

Universal Kolmogorov Complexity However, this definition poses the problem of introducing on an arbitrary Turing Machine U (or an arbitrary programming language). This is not an issue, considering the *Invariance Theorem* (M. Li, Vitányi, et al., 2008) which states that the choice of the Machine U ultimately does not matter. The sketch of the proof is as follows. Consider two Machines U and U'. Since they are both universal Turing Machines, there exists a finite program q which emulates U on U',

[1] Like all other common programming languages, Python is Turing-complete, i.e. it can emulate a Universal Turing Machine (within hardware limitations).

i.e. $\forall p, U(p) = U'(q :: p)$, where :: denotes the concatenation operation. Let p^* be the shortest program that yields a string x on the machine U. By definition, $L(p^*) = K_U(x)$. If we consider the program $q :: p^*$, we have that $U'(q :: p^*) = U(p^*) = x$. Therefore, the complexity of x on the machine U' is *at most* the length of $q :: p^*$. Formally, $K_{U'}(x) \leq L(q :: p^*) = L(p^*) + L(q)$. Which yields $K_{U'}(x) \leq K_U(x) + L(q)$. Note that $L(q)$ does not depend on the string x we consider. Therefore, in general, $K_{U'} \leq K_U + L(q)$. By considering the other direction, with the same reasoning, we can conclude that there exists a constant $C_{U,U'}$ such that, for any binary string x, $|K_U(x) - K_{U'}(x)| \leq C_{U,U'}$.

Thanks to the invariance theorem, we understand that, up to an additional constant, all Turing Machines produce the same complexity measure. Therefore, we can drop the specification of the choice of U in Definition 6.1 to get a universal definition:

$$K(s) = \min\{L(p) | U(p) = s\}\,. \tag{6.2}$$

Application to randomness While probabilistic approaches only consider generation processes, a major application of complexity is to define a sense of randomness of an object *once it has been generated*. Consider the first n decimals of π, that we denote π_n. A short computer program can provide any number of π digits by implementing an approximation formula. Therefore, it is possible to encode the π_n with only $\log_2(n)$ bits of information, plus the length L_π of the computation program, which does not depend on n: $K(\pi_n) = \log_2(n) + L_\pi$. Alternatively, consider a number x of n digits generated via a truly random process. By definition, there is no deterministic program that, given the first digits of x, can predict the next ones. Thus, the shortest program to output x is to explicitly write all n digits of x in the program. Hence $K(x) \approx L(x) = n$ (M. Li, Vitányi, et al., 2008). In this example, while both π_n and x have the same length, one can be encoded in significantly less bits than the other. More generally, in terms of Kolmogorov complexity, random strings are *incompressible*: there exist no significantly shorter program to describe them than enumerating all of their characters.

Using AIT to define unexpectedness Using the length of the shortest generating program as a definition for the complexity of objects allows to model cognitive perceptions. A case in point is the perceived complexity of random events, which cannot be accounted for by standard probabilities. For instance, consider the event of a national lottery resulting in the numbers 1, 2, 3, 4, 5 and 6 being drawn out of all 50 possible numbers[2]. This result would for sure appear somehow simple and surprising, or memorable[3]. However, when only considering probabilities, the 1,2,3,4,5,6 outcome is as likely as any other, e.g. 12,27,31,28,40,43, roughly 2.35×10^{-8}.

To account for such situations, (Dessalles, 2008a) introduces the notion of *unexpectedness*: the 1,2,3,4,5,6 sequence is unexpected because it is simpler to describe than expected: instead of having to spell 6 unrelated numbers, one only has to mention that it is the sequence of the first 6 numbers. This notion can be formalized using insights

[2]This situation occurred in South Africa on Dec, 1^{st} 2020, when the national lottery drew number 5, 6, 7, 8, 9 and 10.

[3]It did in South Africa! In fact, the result seemed so simple that it raised suspicion and an investigation was conducted.

from Kolmogorov Complexity: by modeling a person's cognition as a universal Turing Machine U, one can express the unexpectedness of an event, such as drawing 1,2,3,4,5,6 in the lottery as:

$$Unex(e) = C_W(e) - C(e) \tag{6.3}$$

where $C(e)$ denotes the shortest description the person's machine U needs to describe the event e, and $C_W(e)$ denotes the description that the "world machine" W requires to generate the event e. In the case of the lottery draw, this difference is large, which accounts for the surprising perception of the event.

While it models the cognitive perception of unusual phenomena better than usual probabilities, unexpectedness cannot be applied as such in a computational system. Indeed, this definition is impractical: it relies on the introduction of the world machine W that models the expected generation process the world requires to generate events. This notion is at best ill-defined for realistic applications, and would require an access to causal knowledge that would make the whole purpose of identifying memorable events for abduction obsolete. Also, Kolmogorov Complexity is non computable. As demonstrated in (M. Li, Vitányi, et al., 2008), computing the exact value of Kolmogorov complexity for any object implies the ability of finding the shortest programs running on a Turing Machine. However, some programs do not terminate, and their identification, also known as the Halting problem (Lucas, 2021) cannot be done programmatically. Hence, Kolmogorov complexity can be at best bounded, but not computed.

6.2.2 Memorability as a complexity gap

For the context of smart homes, we implement an approach similar to unexpectedness, defining memorability as a difference between the expected situation and the actual observation. We need to define a measure of the complexity of situations that is consistent with the perceptions described in 6.1 and that can be computed. We propose to start from the consideration that memorable events or object tend to require less words to be described. For instance, consider how "the day of my wedding" appears simpler that "the 10^{th} of September of 2022" even though both sentences unambiguously designate a single day. Similarly, "last year's hottest day" requires less information than "the 23^{rd} day of 8 months ago". Therefore, we propose to define a formal framework to estimate this description length for events, and compare it with an expected description length.

Here, the notion of event is similar to what we have previously defined in Equation 4.1: events are described by a label, a timestamp and a set of characteristics, and they are detected by LECs in the system. We consider an agent that has recorded an *unordered* set of event, that we call its *memory* M. We use the already-defined *predicates* to model human language. The agent knows a finite set of predicates, and their arguments, that we call its *vocabulary* V.

By associating words to predicates, the agent can describe an event as follows. A predicate π, with an additional argument k, maps events from the memory $\mathcal{M}$ to a Boolean value: $\pi(\cdot, k) : \mathcal{M} \mapsto \{0, 1\}$. We can define a corresponding *filter* $f_{\pi,k}$ as an operation selecting all events that are valued true by the predicate π and its argument k:

$$f_{\pi,k}(M) = \{e \in M \mid \pi(e, k) = 1\} \ . \tag{6.4}$$

As predicates can be interpreted as qualifiers, filters represent operations that return all events compatible with the qualifier of the corresponding predicate. It can also be understood as an SQL request retrieving elements matching certain conditions from a database.

Since filters yield another set of events, it is possible to compose them. The application of successive filters narrows down the number of considered event for each iteration, until eventually one or no event remains. The composite operation corresponds to using the conjunction of predicate-defined propositions to qualify the resulting events. Figure 6.1 illustrates this principle on the sentence "last year's hottest day": it results on consecutively applying filters associated with the predicates $\texttt{hot\_rank}(\cdot, 1)$, $\texttt{year}(\cdot, 1)$, $\texttt{event\_type}(\cdot, day)$.

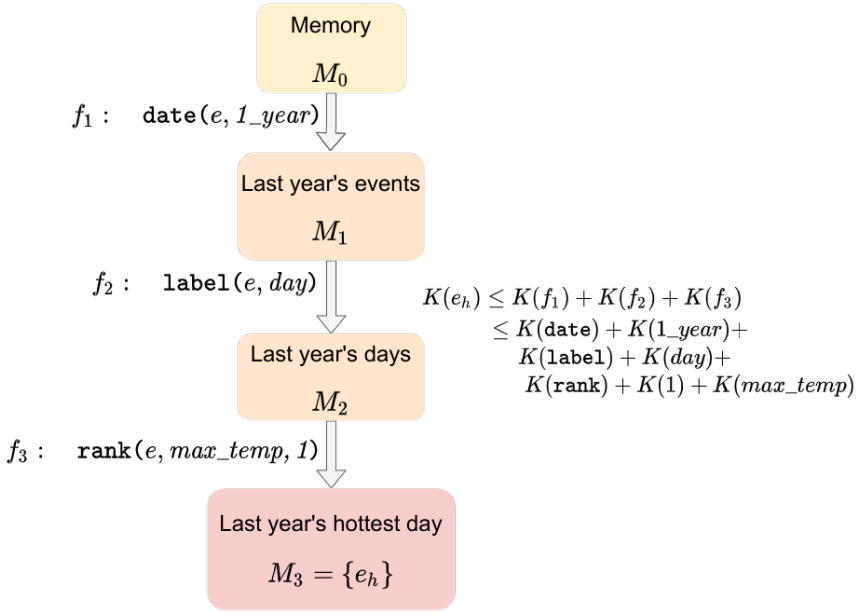


Figure 6.1: Principle of event retrieval. A memory of events is filtered using successive filters derived from predicates. At the end of the process, a single element is left: we say that this event has been *retrieved* by the filters sequence, called a *retrieval path*.

If a succession of filters $f_1, f_2, \ldots, f_n$, when applied to the memory M, results in a single event being included in the final output, we say that this event is *retrieved* by the sequence of filters (or predicates). In that case, the filters sequence $(f_1, f_2, \ldots, f_n)$ is called a *retrieval path* RP of event e, and we write $\text{RP}(M) = e$. If each filter f_i is described using $L(f_i)$ bits of information, then the description of the event e retrieved by the path $(f_1, f_2, \ldots, f_n)$ would be at most the sum of these complexities. Indeed, it is possible to unambiguously describe the event with the description of the filters. By considering the minimal length of such a retrieval path p,

$$K_{desc}(e) = \min_{\text{RP}|\text{RP}(M)=e} \sum_{f \in \text{RP}} L(f)\,. \tag{6.5}$$

The value $K_{desc}(e)$ defined in Equation 6.5 is what we call the *description complexity* of an event e, as it is approximating the length of the shortest description of the event e. This value is dependent on the vocabulary, i.e. the set of predicates known to the agent, from which filters are defined, and the memory, i.e. the past experience of the user. As such, it is a subjective metrics. However, this is a desired property: the naive notion of memorability that we try to coin with this approach is itself highly reliant on the user.

A filter being entirely defined by the predicate π and its argument list k, we can describe a filter $f_{\pi,k}$ by describing π and k. Since π comes from the agent's vocabulary, which is finite and well-defined, it is possible to describe it by giving its index within the vocabulary. Hence,[4] $L(\pi) \leq \log(|V|)$. On the other hand, the optional argument k can be described by using a binary string representation, e.g. if k is a number, then we consider $L(k)$ the length of the binary representation of k. Equation 6.5 becomes:

$$K_{desc}(e) = \min_{V} \left\{ \sum_{i} L(\pi_i) + L(k_i) \mid f_{\pi_1,k_1} \circ \dots f_{\pi_n,k_n}(M) = \{e\} \right\} . \tag{6.6}$$

The description complexity of an event e that has been recorded by the agent is defined as the shortest length of predicates from its vocabulary V that can unambiguously describe the event e from the agent's memory.

When considering an event e, what would be its expected complexity? A usual way to proceed is to consider the average complexity measured for other similar events from the memory M. The notion of similarity is not easy to define. Since we already use predicates to describe events, we consider the following definition: the *neighborhood* $N_{\pi,k}$ of an event e is the set of events from the memory M that satisfy the same predicate $\pi(\cdot, k)$:

$$N_{\pi,k}(e) = \{e' \in M | \pi(e', k) \wedge e' \neq e\} . \tag{6.7}$$

By using the convention that $N_{\pi,k} = \emptyset$ if e does not verify $\pi(\cdot, k)$, we can define the expected description complexity $K_{exp}(e)$ of an event as the average complexity of events across all neighborhoods:

$$K_{exp}(e) = \frac{\sum_{\pi,k} \sum_{e' \in N_{\pi,k}(e)} K_{desc}(e')}{\sum_{\pi,k} |N_{\pi,k}|} . \tag{6.8}$$

From this, we can define the memorability of an event e, according to the agent with memory M and vocabulary V, as:

$$M(e) = |K_{desc}(e) - K_{exp}(e)| . \tag{6.9}$$

Note that a major difference with the definition of unexpectedness in Equation 6.3 is the use of absolute values. This change is motivated by the observation that events that appear unusually complex can be deemed remarkable, too. The original definition of unexpectedness handled this situation by relying on a non-computable complexity. For

[4] It is possible to consider some predicates as being "simpler" than others: for instance, we can use Huffman-like encoding of predicates to consider the most frequent words simpler.

instance, in the "Pisa Tower effect" (Dessalles, 2008b), the Pisa Tower is considered very simple despite being, at first glance, more complex to describe (given that, in addition to its architectural description, one has to specify that it is leaning). This is because, when considering the human description complexity $C(e)$, this particular tower is considered part of the common knowledge. The Pisa Tower is famous and directly "stored" in the person's memory. Hence, it requires minimal information to be described, its name only being sufficient to describe it. On the other hand, any other tower in Italy would be more complex, as it would require to be described using the Pisa Tower as a basis. Hence, using the definition of Equation 6.3, the Pisa Tower is remarkable, because its unique complex feature (leaning), makes it more famous and simple than other comparable towers. However, this kind of self-reference implies non-computability of complexity. For this reason, we settle on introducing absolute values to account for the same kind of effects while keeping a computable metrics.

As all notions used for the computation of memorability in Equation 6.9 have been defined and rely only on a finite number of computations, it is possible to implement an algorithm to evaluate the memorability of events from a memory and a given vocabulary of predicates. Algorithm 4 is an example of such implementation: it iterates over possible retrieval paths constructed from the vocabulary, from shortest to longest. At each iteration, it considers all possible predicates, applies the corresponding filter (line 13). If the resulting memory is a singleton event, the complexity of this event is set accordingly (line 15). Otherwise, the memory resulting from the filter application is appended to the list of memories to explore during the following iteration (line 17). The termination of this algorithm comes from the condition in line 10: if the complexity of the currently considered retrieval path exceeds the maximum complexity of all events, which is bounded by some finite value `B`, the algorithm exits the innermost loop without appending anything to be explored during the next iteration. As the length of retrieval paths increases with the number of iterations, the algorithm is guaranteed to terminate once every path shorter that `B` has been explored, and there are finitely many such paths.

6.2.3 Defining relative memorability

For the purpose of performing "naive" abduction, as introduced in Section 6.1, one needs the refinement of the memorability we have defined. Indeed, during an abductive inference, the agent has more than the mere knowledge of its past observations and vocabulary that were supposed in the definition of memorability. The agent also has knowledge of the phenomenon it wants to explain. This additional knowledge should be included in some form in the definition of a metrics that can be used for abduction. We propose the definition of "conditional memorability", which comes from plain memorability that was defined in Equation 6.9.

Conditional Kolmogorov complexity $K(x \mid y)$ is defined as the length of the shortest program p that, when augmented by appending y to itself:

$$K(x \mid y) = \min_{p \in \{0,1\}^*} \{L(p) \mid U(p :: y) = x\} \ , \tag{6.10}$$

where :: denotes the concatenation operation. Non-computability and invariance theorem can be directly extended to conditional complexity.

Algorithm 4: Iterative computation of the approximate complexity

```
1  current_explore ← [(M, 0)] ;
2  future_explore ←[ ] ;
3  pass ← 0 ;
4  K(e) ← B ;
5  while current_explore ≠ [ ] and pass < max_pass do
6      for (M_prev, K_prev) ∈ current_explore do
7          for π ∈ P do
8              for k ∈ {0,1}* do
9                  K_current ← L(π) + L(k) + K_prev ;
10                 if K_current > max_complex then
11                     break ;
12                 end
13                 M' ← f_{π,k}(M_prev) ;
14                 if M' = {e} then
15                     K(e) ← min(K(e), K_current) ;
16                 else
17                     future_explore.append((M', K_current));
18                 end
19             end
20         end
21     end
22     current_explore ← future_explore ;
23     future_explore ← [ ] ;
24     pass ← pass + 1;
25 end
```

Considering the definition of conditional complexity, we introduce the *conditional memorability* $M(c|e)$ as its counterpart. We apply the same definitions as plain memorability, but append a term k_e to all description complexity formulas as an additional argument to all predicate filters used in the original equation 6.5. The string k_e encodes the event e that we want to account for. Conditional description complexity is defined as:

$$K_{desc}(c \mid e) = \min_{V} \left\{ \sum_{i} L(\pi_i) + L(k_i) \mid f_{\pi_1,k_1::k_e} \circ \ldots f_{\pi_n,k_n::k_e}(M) = \{c\} \right\} . \quad (6.11)$$

Note that the addition of k_e is "free" in terms of complexity: the length of this description of e is not included in the total sum, which corresponds to the fact that this information is an input of the problem and is therefore excluded from the computation. An example of the difference between plain complexity and conditional complexity is a time stamp retrieval predicate: an event A that occurred a long time ago is *a priori* complex to describe. However, given another event B that occurred the same day as A, A becomes simpler to describe, given that information about B is already provided.

The definition of conditional memorability directly comes from conditional complexity, and is similar to the interpretation we made of plain memorability (Equation 6.9):

$$M(c \mid e) = |K_{desc}(c \mid e) - K_{exp}(c \mid e)| \ . \tag{6.12}$$

This definition of conditional complexity can be used to perform abductive inference. In Example 6.1, we expect that, given the observation of the light dimming, the usage of the brand new TV will be remarkable: it occurred just prior to the light dimming, in the same room and is unique as this object was never used before. Therefore, conditional memorability should highlight this event as being a good candidate hypothesis to account for the strange phenomenon.

6.3 Related Works

Our examination of memorability as a tool for abductive inference is new, to the best of our knowledge. However, other practical applications have already been considered for AIT, some of them related to the notions of causality and knowledge discovery.

Inductive inference was one of the original motivations that led to the definition of the notion of complexity by R. Solomonoff (Solomonoff, 1964; M. Li, Vitányi, et al., 2008). In his works, he introduces the notion of algorithmic probability of a string $m(x)$ as the chance of a randomly chosen program would generate the output x. This probability is closely related to complexity, as the relation $-\log(m(x)) = K(x) + O(1)$ directly links the two notions together. Deriving from this concept, Solomonoff introduced a universal predictive machine: a conceptual computer that could provide the best possible guess regarding the future state of an evolving world, thus "solving" the induction problem. This has notably been applied to create a "Universal Artificial Intelligence" (Hutter, 2004).

Regarding Causal Inference, Kolmogorov complexity can be relevant, too. Notably, it is consistent with the formal definition of causal relations between variables. A causal relation $A \to B$ means that there exists a random noise N_B and a deterministic function f_B such that $B = f_B(A, N_B)$ (Peters, Janzing, and Schölkopf, 2017). In this case, (Janzing and Schölkopf, 2010) showed that the complexity of the joint probability distribution is simpler in the direct rather then inverse direction: $K(\mathbb{P}(A)) + K(\mathbb{P}(B \mid A)) < K(\mathbb{B}) + K(\mathbb{P}(A \mid B))$. Here, complexity confirms the intuition that, when facing a causal phenomenon, it is simpler to consider conditional probabilities that are consistent with the causal direction.

"Isolation Forests" (F. T. Liu, Ting, and Z.-H. Zhou, 2008; Hariri, Kind, and Brunner, 2021) relies on a method that is globally similar to ours for identifying outlier points in datasets. Here, a set of random binary tree classifiers are generated: dichotomous cuts are randomly realized until all points have been singled out. The number of cuts required to isolate each point its called the isolation depth of the point for this random binary tree classifier. By repeating the process, (M. Li, Vitányi, et al., 2008) estimates the average isolation depth for each point, and labels as outliers those that have low depth. This method can be interpreted using AIT: each cut corresponds to information, regarding which variable and value is used. Therefore, points that require less cuts (and have lower isolation depth), require less information to be singled out. Thus, this method

also identifies as outliers points that require less information to be described. However, compared to ours, it is restrained to homogeneous data and does not define a relative complexity for points. By contrast, our usage of predicates allows to circumvent these two limitations.

6.4 Experiments and results

All experiments presented in this section are available as Jupyter Notebooks on `https://github.com/EtienneHouze/memorability_code`.

6.4.1 Illustration of the approach

To illustrate how our approach to memorability works, we first introduce a simple example: consider that an agent has a memory of the past days. One of them is the agent's wedding, while the others have nothing particular. Thus, we consider a memory M of several thousand events, all labeled "day". The agent's vocabulary contains 2 time predicates: $\texttt{days\_ago}(e, k)$ is true if the event e occurred k days ago; $\texttt{days\_from}(\texttt{e}, \texttt{k}, \texttt{e}')$ is true if the event e occurred k days relative to another event e'; $\texttt{marriage}(e)$ is true if the event corresponds to the agent's wedding.

The resulting memorability computed on these events is shown in Figure 6.2. Several observations regarding our method can be made from this plot. First, the wedding day (circled) corresponds to a peak in the computed memorability, as do the most recent days on the right end. This is the expected behavior, as these days are memorable, either by their uniqueness or by their recentness. Second, the "derivative" can be discontinuous around 0. This also was to be expected, as the absolute value operation in the definition of Equation 6.9 introduces discontinuity in the derivative when reaching the origin. Third, most of the events are part of a "main sequence" that corresponds to designating them with the time of their occurrence; this sequence is logarithmic, as the binary code length of a number is logarithmic.

Also, this experiment illustrates, as we have already evoked, the ill-definition of the "least memorable" events: consider, for instance, the event on the far left side of the graph. The corresponding day, which is the oldest recorded, is the most complex to describe with our vocabulary. However, its memorability, given that it is a difference with the average complexity, is not 0. We can quantity this phenomenon. By considering a situation where most events' complexity is a logarithm, the average complexity K_{exp} of an event in a memory of size N is

$$\begin{aligned} K_{exp}(e) \approx \sum \frac{1}{N} \log(k) = \frac{1}{N} \log(N!) &= \frac{1}{N} \log\left[\left(N^N e^{-N} \sqrt{2\pi N}\right)(1 + o(1))\right] \\ &= \log(N) + 1 + \frac{1}{2N} \log(2\pi N) + o\left(\frac{1}{N}\right). \end{aligned} \tag{6.13}$$

Considering that the most complex event has a complexity of $K_{desc}(e) = \log(N) + C$, where C is a constant with regards to the size N of the memory (C accounts for the

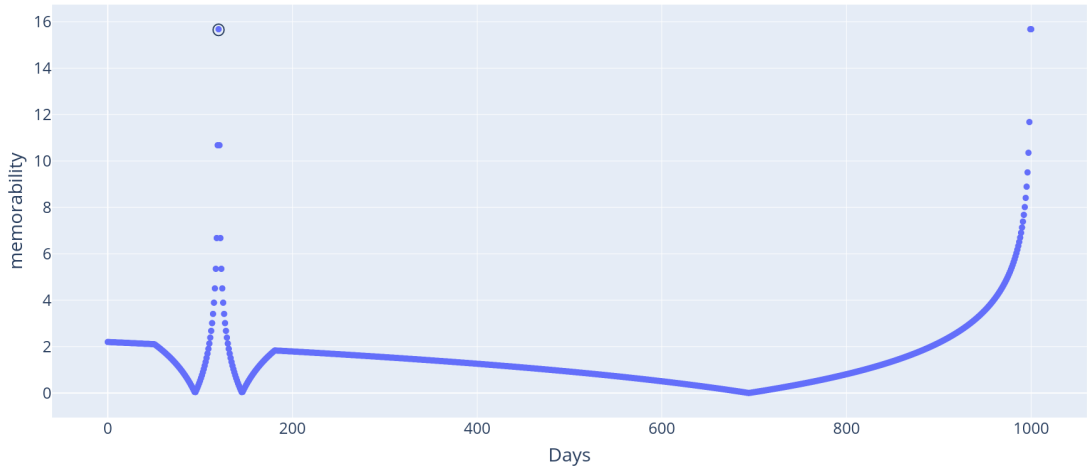


Figure 6.2: Memorability of days in the example of the agent's wedding. The wedding (circled) is clearly visible as being a remarkable event.

complexity of the predicate, not its argument), we can estimate the memorability of this most complex event to be around $C + \frac{1}{2N}\log(N) + o\left(\frac{\log(N)}{N}\right)$. While this memorability gap is set to be bounded within a finite range, we can extrapolate that complexity is ill-defined when it is within this range.

6.4.2 Illustration in smart home setups

We have conducted two distinct experiments to reproduce typical situations that may arise in the smart home context and observe how our memorability metrics can identify remarkable events and be used for abduction, following the "naive abduction" principle of proposing relevant causes.

A house with several rooms

Setup description In this first setup, we used the iCasa simulator (Philippe Lalanda and Hamon, 2020) to run a simulation spanning over a long time period of more than a year (420 days). A 4-room house is modeled, each room's temperature being controlled by a basic system. We monitor the temperature for all rooms and the outdoor environment. The resulting time series, visible in Figure 6.3 is then processed by a basic event detection module: an event is recorded to describe each day (maximum and minimum outdoor temperature), as well as when temperatures rise over or lower under predetermined threshold or when a device is added or removed. Over the course of the simulation, outdoor temperature randomly varied to introduce some noise. At specific times, we intervened to create anomalies (unusually high or low outdoor temperatures, addition/removal of a component). This resulted in 560 events, spanning over a duration of 420 days, which are stored in the agent's memory.

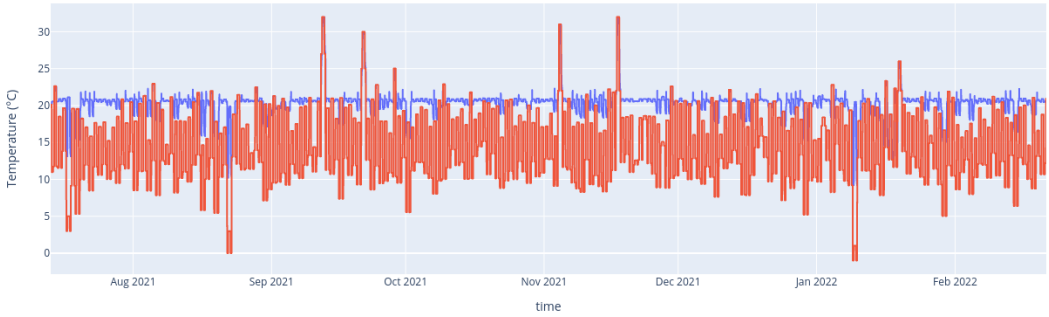
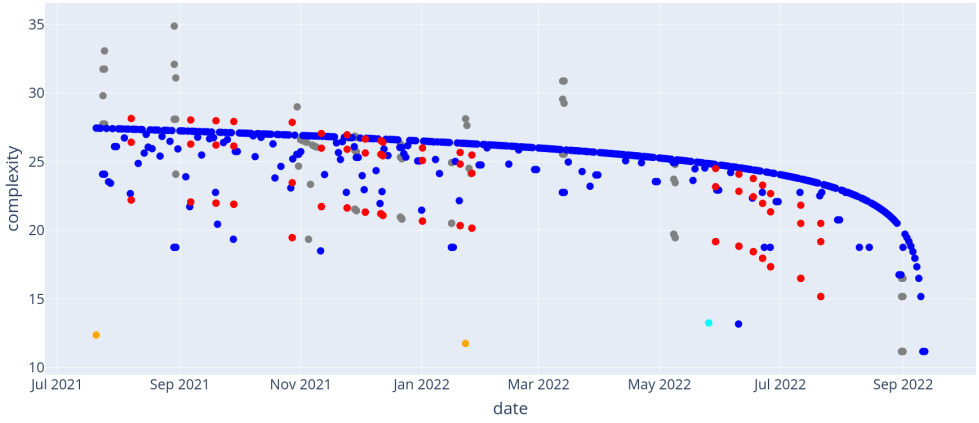


Figure 6.3: Time series of recorded temperatures over 420 days of simulation in iCasa. We display outdoor temperatures (in red) and indoor temperature of one room. Some handmade anomalies are visible on the graph (temperature highs and lows).

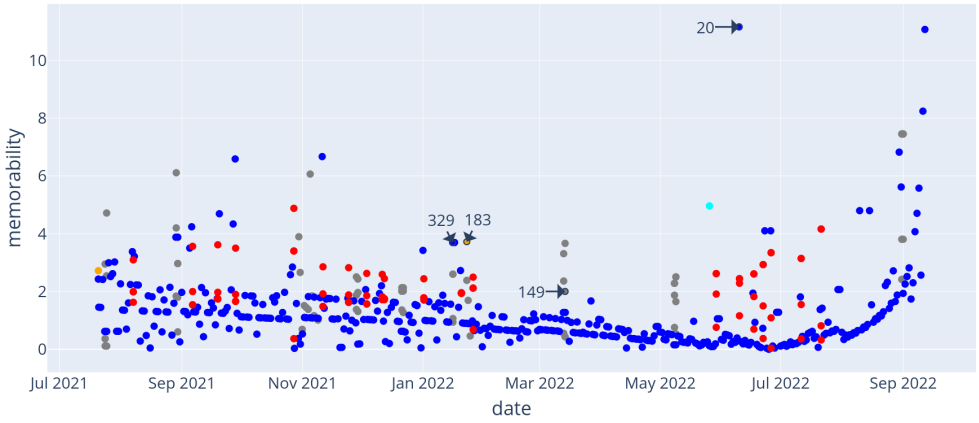
Analysis The agent has a vocabulary composed of the predicate $\texttt{rank}(\texttt{e}, \texttt{k}_1, \texttt{k}_2)$ which is true if the event e is ranked k_1^{th} alongside the k_2^{th} axis, $\texttt{day}(\texttt{e}, \texttt{k})$ which indicates whether the event occurred on day k, $location(e, k)$ indicates whether e occurred in location k and $\texttt{label}(\texttt{e}, \texttt{k})$ whether e is labeled k. For categorical arguments, such as the label, the axis or the label, the encoding of k is frequency-based: for instance, the most frequent label is encoded as 1. The computed complexities for events are displayed in Figure 6.4a.

Similar to Figure 6.2, a general sequence of events with logarithmic complexity is visible: this corresponds to events for which the best description is to specify their type and date of occurrence. In the corresponding memorability (Figure 6.4b), the phenomenon of the ill-defined "least memorable" events is also visible, notably on the left end of the graph. As predicted by Equation 6.13, this phenomenon is limited in amplitude. Some events stand out as particularly memorable: recent events, which is a common feature of our approach, and exceptional phenomena which correspond to handmade events in the simulation: for instance, event labeled 20 corresponds to the hottest day recorded, which was manually set.

Our memorability score does not aim at replacing anomaly detection techniques. A memorability-based detection fares quite well: it flags events with a memorability above a given threshold. We can then compare, for a given threshold, the flagged events with the events that were manually generated, thus computing the False Positive Rate (FPR) and the True Positive Rate (TPR) of the method. By varying the threshold, we can obtain values, which can then be plotted to create the Receiver Operator characteristic (ROC) curve of the detector (Figure 6.5). This method is often used to assess the performance of a classifier: a good method will hold a high TPR across many FPR values. Here, we can see that the memorability-based classifier is able to identify most of the manually-generated events. However, high scores are given to recent events. This phenomenon is a voluntary feature that mimics a short-term bias, but it hinders the performance of a detection method by making old memorable events harder to detect with a simple threshold comparison.



(a)



(b)

Figure 6.4: Computed complexity and memorability of events in the 4-room house scenario. (a) shows the complexity of events with label "day" (blue), "device_removed" (orange), "device_added" (cyan), "cold" (gray) and "hot" (red). (b): memorability of the same events. Some remarkable events are annotated.

A brand new TV

Setup description For the second scenario, we implement Example 6.1 that was presented in the introduction to this chapter. For this, we generate a series of events with two different labels: "light" events correspond to the lighting of the room over a duration of one hour; "TV" events occur when the TV is switched on, and they contain the TV model in their characteristics. After having generated similar events over a period of several months, we introduce a new TV and we switch it on. This is recorded as a "TV" event, with the new TV name as a characteristic. Subsequently, the dimming of the light is also recorded as a "light" event. In total, as the record spans over 100 days, around 2500 events are considered, most of them being "light" events.

The vocabulary of predicates contains the following elements: $\texttt{daytime}(\texttt{e})$ states whether e occurred during daytime, $\texttt{nighttime}(\texttt{e})$ whether it occurred during nighttime,

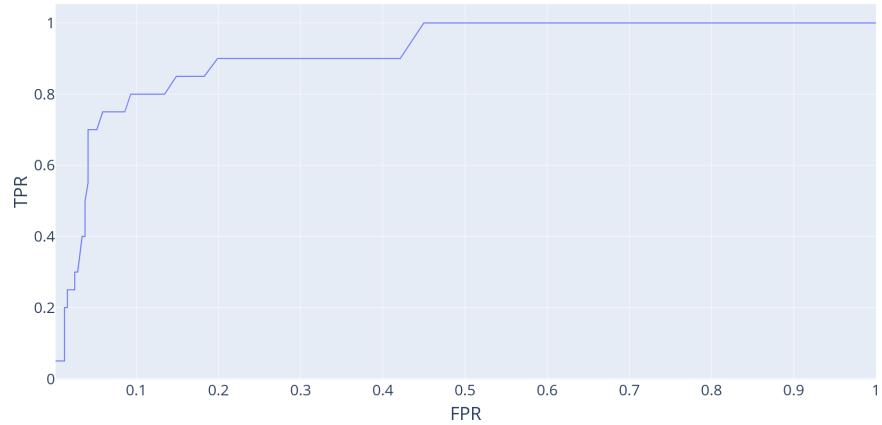


Figure 6.5: The ROC curve of the memorability-based classifier. This result is obtained by varying the detection threshold. Ground truth classification consists of 20 manually annotated events that correspond to hand-made perturbations in the simulation

$low\_light(e)$ and $high\_light$ whether e is an event where the light is low or high, $day(e, k_D)$ and $hour(e, k_H)$ whether e occurred k_D days ago at hour k_H. A predicate $\texttt{device}(\texttt{e}, \texttt{k})$ indicates the type of the device corresponding to the event e; notably, this predicate is used to discriminate the new TV.

Analysis The resulting memorability scores are shown in Figure 6.6. The same main sequence of logarithmic events appears, similar to Figures 6.2 and 6.4b. In addition, two other sequences are visible. Some light events appear notably more memorable than the rest: they correspond to "dark" events occurring during daytime: since they are rare, they require less information to be described than the majority of light events, thus appearing more memorable. TV events, as they are a rarer event label than light events, appear as being more memorable, in overall. The smart TV usage, being unambiguously described by referring to the TV's model, appears as significantly memorable. So does the light dimming, based on its recentness and its occurrence during daytime.

Table 6.1 exposes the three most memorable events relative to the light dimming event following the use of the smart TV, using the conditional memorability defined in Equation 6.12. It appears that this metrics identifies the smart TV usage as the most memorable event, knowing the subsequent light-dimming event. Here, the result of the computation follows the scenario of Example 6.1: the most memorable event, relative to the observation of the dimming of the lights, is the fact that the brand new smart TV has just been turned on. Therefore, based on this sense of memorability only, one can propose that this event is the cause of the dimming, without any additional knowledge regarding how the TV operates.

6.4.3 A subjective metrics

By essence, memorability remains a subjective notion. Each individual may consider different events as being memorable, based on their knowledge and past experience. For

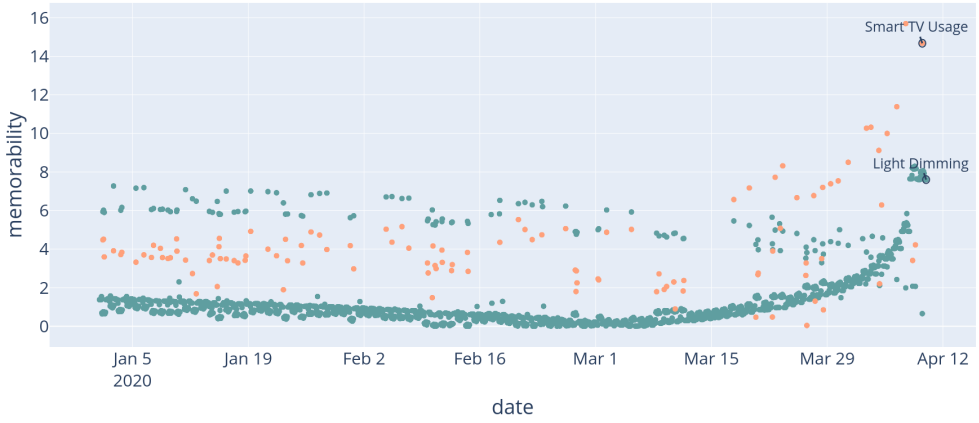


Figure 6.6: Memorability of the events in the TV scenario. The events corresponding to the first use of the new TV and the subsequent light dimming are circled. They are considered as memorable by our metrics.

Event ID	Description	Relative memorability (bits)
2513	Use of smart TV	16.76
2427	Last use of the old TV	14.81
2411	Second-last use of the old TV	11.21

Table 6.1: The three most memorable events, relatively to the light dimming event.

instance, a person living in France will consider some ancient buildings as being non memorable, while they may stand out from the point of view of a foreign tourist. This phenomenon is accounted for in the definition of unexpectedness (Dessalles, 2013).

Our metrics, while being formally defined in Equation 6.9, is subjective, too. Here, memorability is defined relative to a given memory $\mathcal{M}$ of events and vocabulary $\mathcal{V}$ of predicates. To understand the effect of vocabulary, we compute the description complexity for events from the TV scenario with different vocabularies. The results are presented in Figure 6.7: the introduction of `daytime` and `nighttime` predicates between Figures 6.7a and 6.7b leads to visible changes as low-light daytime events can be described more simply. In Figure 6.7c, the introduction of a `select` predicate, which can select any event from the memory at the cost of $\log(N)$ bits creates an upper bound for event complexity. This introduces a threshold phenomenon: all events that would otherwise cost more than $\log(N)$ bits can be described more effectively by designating them by their ID among all other events. The relevance of allowing this ID-based retrieval method is not clear, as the exact functioning of human memory is not comparable to that of a computer (Rubin and Umanath, 2015).

6.5 Integration into the explanatory system

In this chapter, we have defined and formalized the notions of memorability and conditional memorability. We argue that they can be used for abductive inference in some

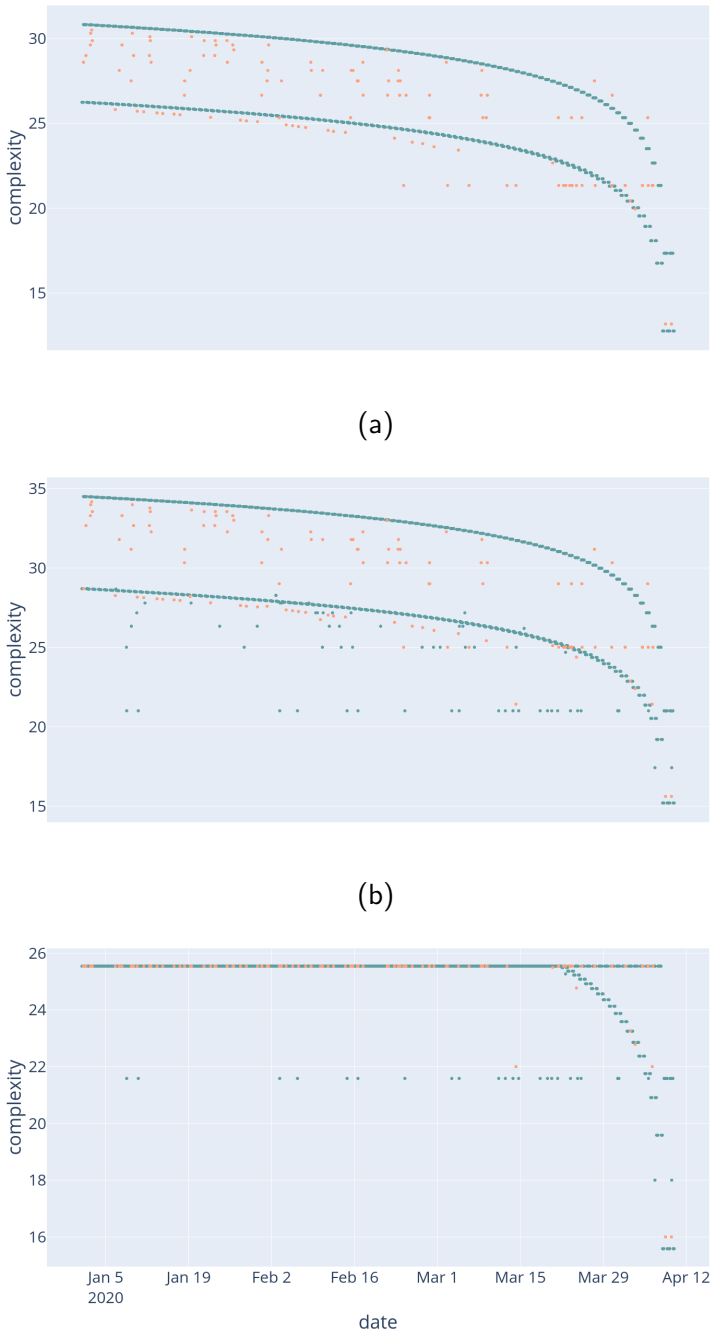


Figure 6.7: Description complexity computed with different vocabularies for the TV example scenario. In (a), the vocabulary only covers the event type and time of occurrence. In (b), a predicate to describe daytime or nighttime occurrence is added, which discriminates some low-light events as being less complex. In (c), a predicate to select any event with its ID is added: this introduces an upper limit cor complexity, which corresponds to the cost of this generic selection predicate.

circumstances. Notably, memorability can be used to propose original hypotheses when other abductive forms of reasoning fail, due to lack of prior knowledge, for instance. As such, memorability-based abduction can be integrated into our smart home explanatory system as abduction modules that detect internal memorable events and propose them as causes. A low score can be given to this method, so it is only used in last resort. However, the relevance of this approach is limited: by design, most internal events are already handled by the LECs' understanding of its functioning and hence integrated into other abduction methods (i.e. having causal model of its working).

Another possibility is to implement memory-based abduction across several components. This is shown in Figure 6.8. First, an interpretation module on a LEC identifies a recorded event as being remarkable, given its knowledge of past observations and predicates. Then, this module notifies other LEC's of its observation by sending the Boolean proposition describing it, and how memorable this is. Components that are equipped with the corresponding abduction module can register this proposition. Then, in a subsequent D-CAS call, these modules can propose the memorable proposition as a hypothesis.

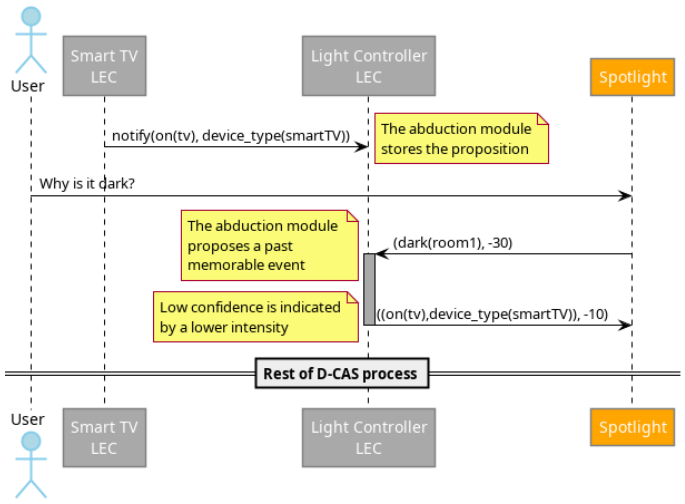


Figure 6.8: Principle of a memorability-based abduction. A first LEC identifies an event as being memorable, and notifies other LECs of this occurrence by sending the propositions describing the event. In a subsequent D-CAS call, this Boolean proposition can be suggested as a causal hypothesis by the abduction module.

Note that this situation could also be handled by considering the addition of the Smart TV: when the TV is integrated into the smart home system, the Spotlight supervises its addition to the explanatory system. This configuration change is observed by the LEC attached to the Spotlight. If this LEC is using the memorability module, it will interpret this component addition as remarkable, since this is the first time such module is in the system. Hence, it will notify other modules of the Boolean proposition $\texttt{added}(\texttt{t}) \wedge \texttt{devicetype}(\texttt{t}, \texttt{smartTV})$. This process results in a rationale identifying the TV's addition as the cause for the strange light dimming event. While being arguably different from what is proposed in Figure 6.8, this output is likely to inform the user about the situation.

Memorability-based abduction bears no guarantee regarding the accuracy of its hypotheses. However, as we have discussed in Chapter 5, erroneous abductive proposals can be handled by D-CAS: the high-level rationale identifies a hypothesis as being erroneous if subsequent simulations are inconclusive or if the proposed conflict is not confirmed. It is possible for future refinements of the system to integrate feedback into memorability-based abduction to improve predictions. However, this is considered out of scope, as the first goal of naive abduction is to provide hypotheses in no-knowledge context.

Chapter 7

Conclusion

Summary

As explanation is identified as a multidisciplinary issue, this research has made progress alongside different axes. In this concluding chapter, we present an overview of the different contributions. Then we discuss the main limitations of the current state of the explanatory system, which pave the way towards future improvements.

Four main contributions can be identified in this thesis. First, we have defined and illustrated the different requirements for an explanatory engine in Cyber-Physical Systems such as smart homes. Second, we have provided a general framework for the generation of explanations in the context of complex systems, where knowledge is scattered across multiple components. Third, we presented an architecture that enables the previously defined algorithm into realistic smart home systems. Fourth, we provided a novel way to propose candidate hypotheses in abductive inference based on a newly introduced measure of memorability: this aims to implement the intuitive human approach of proposing past memorable events as relevant hypotheses when faced with unprecedented phenomena.

The main objective of this research was to demonstrate the feasibility of a smart home explanatory system and design its main elements. While the proposed architecture and algorithm meet the expectations we set, many open questions remain before realizing an entirely explainable smart home system.

7.1 Contributions

The main question posed in the introduction to this thesis was "How can we design an explanatory system that is compatible with the different characteristics of a smart home system?" In this thesis, we have broken down the question into different steps, to each one we have brought an answer:

First, we defined, in Chapter 3, the six main characteristics that the explanatory system should present to comply to the desires and restrictions of a smart home. This list resulted from the review of existing constraints, approaches and goals of XAI in Chapter 2. The six defined goals are: explanation's shallowness, transparency, contrastive nature, privacy compliance, self-adaptation and genericity.

The second major contribution of this thesis is the definition of the D-CAS algorithm, exposed in Chapter 3, and its tree-based variation in Chapter 5. D-CAS is based on the principle that contrastive explanation can be generated by exposing the trace of a conflict-solving process. It brings a minimal representation of the system's knowledge as *Boolean propositions* that can be given *necessities* to represent opinions. The centralized part of the process merely redirects requests to local experts that are able to perform the operations of *conflict identification*, *abduction* and *simulation* using their knowledge. The original D-CAS process is refined by proposing to integrate it tightly with a tree representation of the rationale: this allows for a visual representation of the output.

A third contribution is the design of the architecture enabling the proposed D-CAS algorithm and preserving the self-adaptive features of the existing smart home. In the solution that we present in Chapter 4 and implement in Chapter 5, the general organization is to define one LEC to each existing Smart Home Component. The LEC is itself a generic platform, upon which are integrated specialized modules that are defined by specifications exposed by the SHC. These modules are responsible for the interpretation (i.e. mapping measures to propositions), abduction (proposing causes) and simulation (imagining consequences).

A fourth contribution, exposed in 6, is a novel way to apprehend abduction. As we have identified abduction as an essential part of explanations, it is necessary for the explanatory system to be able to propose hypotheses. In case a hypothesis proves to be erroneous, it can later be discarded by the D-CAS algorithm. In case there is no previous knowledge regarding past occurrences of the phenomenon, nor understanding of the underlying physics, the system must still be able to propose relevant ideas. Here, we bring forth the argument that, when facing similar situations, humans can consider memorable past events as being possible causal hypotheses. We defined a notion of memorability based on Algorithmic Information Theory.

These various contributions resulted in several communications over the course of the research. Table 7.1 lists them with their corresponding references.

In addition to the contributions that are exclusive to this PhD thesis, another research was conducted on a method to identify causal relations in a Smart Home environment and expose them as a Causal Bayesian Network. This work comes from an original idea from Mr. Kanvaly Fadiga, then a student at Télécom Paris, and led to a communication in the 2021 ACSOS Conference (Fadiga et al., 2021). The Causal Bayesian Graph generated by this approach can be integrated as inference knowledge for abductive reasoning. This

Type	Description	Reference
Workshop Paper	Communication focused on the decentralized architecture	(E. Houzé, Diaconescu, Dessalles, Menga, and Schumann, 2020)
Doctoral Symposium Paper	Brief description of the overall approach of the thesis	(É. Houzé, 2021)
Workshop Paper	Using D-CAS in Multi-Agent causal discovery for Policy-Making	(E. Houzé, Diaconescu, and Dessalles, 2022)
Conference Paper	Communication focused on the description of D-CAS	(E. Houzé, Dessalles, et al., 2021)
Conference Paper	(Second author) Learning causal Bayesian Networks in smart homes using *do-operations*	(Fadiga et al., 2021)
Journal Article	The concept of memorability and its use for abduction	(É. Houzé, Dessalles, et al., 2022)
Conference Paper	(Accepted) In-depth presentation of the generic architecture for the explanatory system	(E. Houzé, Diaconescu, Dessalles, and Menga, 2022)
Patent	(Submitted, under review) Design of a LEC, twin-memory	(É. Houzé, Menga, et al., 2021)

Table 7.1: List of contributions (communications, article and patent)

research can be envisioned as a novel way to discover possible causal relations between variables in the smart home. Starting from considering all possible relations (i.e. a fully connected graph of causal influence), the method performs simulations to test the outcomes of *do-operation*, as defined by (Pearl, 2009). The method is able to handle non-doable variables, i.e. variables which cannot be simulated, to some extent.

An extended version of the work, currently under review, has been submitted to the journal *"Transactions on Adaptive and Autonomic Systems"* in early 2022. This extension tackles the challenge of scalability to larger systems. As is, the method originally presented at ACSOS 2021 scaled as the number of variables to consider squared. Using existing knowledge of the hierarchical organization of the system can discard many relations, effectively lowering the complexity of the process to linear.

7.2 Limits and possible future development

The main focus of this research was to develop the base architecture and algorithm to power explanatory systems that are able to handle the various types of situations that arise during spontaneous use of a smart home. While the examples provided in Chapter 5 illustrate some singular use cases and interaction possibilities, several major limitations remain in the system. These limitations can lead to further research to develop specific aspects of the overall system.

7.2.1 Limitation of the underlying system

When we introduced the architectural principles of our explanatory system in Chapter 4, we motivated the one-to-one pairing of existing smart home components with LECs by the fact that this allows self-organization capabilities of the underlying control system to be used by the explanatory system as well. However, this choice comes with limitations: as the structure of the explanatory system follows that of the underlying control system, its handling of proposition is limited to the latter. For instance, in its current state, the explanatory system cannot handle propositions spanning over multiple components, or combinations: consider Example 7.1. This limit results of a design choice, as explained in Section 3.3.5: considering the possibility of multi-component propositions results in a significant increase of proposition combinations to explore, bringing a complexity that infringes on the shallowness principle. This example is similar to the "Firing Squad" causal structure (Akleman et al., 2015), where causal responsibility is shared among different actors: the solution here can be to identify the common confounding factor.

Example 7.1

An unusually low temperature is felt in the room. It comes from the fact that several windows are open. Here, using D-CAS correctly identifies a window as being the responsible cause, but closing just window is not enough to end the conflict. Therefore, D-CAS fails to a correct solution for this conflict, which would be: "It is cold because all windows are open in the room"

To circumvent this issue, we propose two possibilities. First, shifting the organization responsibility entirely to the underlying control system. Self-organizing systems (Banzhaf, 2009) aim to best adapt their organization to requirements and constraints of the environment. This can be useful in smart home context, and has already been considered in (Diaconescu, Mata, and Bellman, 2018). In Example 7.1, a self-organizing control system can identify that there is a requirement for a room-wide window manager, which would later be integrated into the explanatory system and be responsible of a `any_window_open` predicate. A second possibility is to directly adapt the organization of the explanatory system to account for these situations without modifying the underlying control system. For instance, it is possible to consider certain types of feedback that would lead to the addition of new LECs in charge of specific inter-components propositions, rather than being attached to just one component.

7.2.2 User interaction

We have defined shallowness and transparency as two requirements for the general explanatory system presented in this thesis. This may result in arguably unnecessarily heavy outputs (as seen in the failure discovery example from Chapter 5 where all possibilities are displayed). Future improvements to the method can be made by considering better integration of interaction to learn from the user's specific needs and wishes. In the proposed implementation of Chapter 5, user interaction uses a CLI and a GUI that displays past rationales as trees. Interaction is here limited to asking questions about predicates

that are already known to the system, and, once an explanatory tree output is presented, explore another possible path by changing considered necessities. It remains far from the Natural Language that is targeted by the explanatory system. Ease of interaction comes as a major requirement since many smart home applications target elder people (Zimmermann, Ableitner, and Strobbe, 2017; Mekuria et al., 2019) who are more prone to use voice interaction and other NLP-based interfaces (Jivani, Malvankar, and Shankarmani, 2018; Kowalski et al., 2019).

By design, our approach is meant to be coupled with a Natural Language Processing (NLP) unit (which can be comprised in the *User Interface* layer in Figure 4.1). This coupling is facilitated by the use of Boolean propositions in the explanatory rationale. However, these propositions need to be in line with the user's experience to be relevant. A future development is to identify and learn predicates at runtime, based on previous user interactions. A proposed lead for further research is to identify the *contrastive* dimensions of events, and design predicates to name them. For instance, learning the concepts of `cold` or `hot` as deviations from room temperature prototypes (Dessalles, 2015), discarding other dissimilar dimensions, which limits the impact of the curse of dimensionality[1].

7.2.3 Module implementation

The performance of the explanatory system presented in this thesis is hard to evaluate: even though it satisfies the six main goals we defined, as we have seen in the implemented examples of Chapter 5, its ability to explain situations is limited to the performance of its constituting components. Namely, the abduction, interpretation and simulation modules. While the issue of multi-component propositions has been discussed above, individual modules remain a major axis of improvement for the system. In Chapter 6, we presented a novel approach for abduction that is consistent with our global design. Other approaches for abduction can be derived from popular XAI approaches. For instance, one could use Anchors (Ribeiro, Singh, and Guestrin, 2018) to derive meaningful predicates, which can then be proposed as possible causal hypotheses when looking for the cause of an observation.

The approach of learning causal models by performing do-operations on a digital twin that we explored in (Fadiga et al., 2021) can also be used as a basis for abduction modules. By discovering the causal relations between variables, this method can generate abductive hypotheses for observation, without relying on pre-existing rule base knowledge.

Our architecture is modular and is intended to benefit from the addition of several tools for abductive reasoning, simulation or predicate interpretation. As we have illustrated with Example 6.1 concerning abduction, parallel methods can be used depending on the knowledge available to the system. The same is true with interpretation and simulation: threshold-based interpretation, similar to what is implemented in the demonstrator, is adequate for notions such as temperature or window states. More ad-

[1] This "curse" accounts for counter-intuitive phenomena occurring when considering high-dimensional data. For instance, two points that differ by a small amount across many dimensions are far apart when using standard euclidean metrics.

vanced processing, such as anomaly detection (Cook, Mısırlı, and Fan, 2019) can be used in parallel to detect device failures from time series data. As many approaches can be used, the overall explanatory system will benefit from future advances in XAI, data analysis and simulation tools.

7.3 Final words

Explaining a machine-made decision is an important challenge for AI for the years to come. In this thesis, we tackled the question of designing an explanatory system for smart homes. While the target application is quite narrow, the general principles and methods presented in this thesis aim to be generic. The reach of these advances goes beyond the initial application domain, and should remain relevant to future explainable AI systems which focus on interactions with a physical environment: autonomous cars and vehicles, all kinds of "smart" devices and smart-device-based systems, self-aware control systems, etc.

In smart homes, the challenge does not come from the intrinsic complexity of the models used (they are often simple) but rather from the variety of devices, the interactions with the user, the possible failures and the importance and uniqueness of the context. Therefore, we propose an approach that differs from most of state-of-the-art XAI systems: we focus on a high-level organization that generates an explanation as a conflict-solving rationale, composing from knowledge scattered across various specialized components. The intuition behind this approach is that explainability in complex systems requires two kinds of operations. First, low-level operations that identify the causes and consequences of phenomena by standard means that are not necessarily explainable or transparent. Second, a higher-level explanation generation process that uses these operations to build a system-wide reasoning. This process must be understandable by the user, as the explanation consists of the generation itself. It must be a transparent and shallow process, allowing inter-component communication to solve system-wide problematic situations.

The realization of a demonstrator allowed us to illustrate the behavior of our approach in various situations in which the issue to explain originated either from a conflict between goals, from a component's failure or just from the default behavior of the system that was not understood by the user. The general D-CAS algorithm allows to handle this variety of situations while providing standard operations and requirements, component-wise. This approach aims to pave the way towards future improvements by allowing the development and integration of more advanced interpretation, abduction and simulation modules into the explanatory system. The modular and generic architecture developed in this thesis revolves around the possibility to add new features, vocabulary and reasoning tools without major modification to the system.

Bibliography

Abu-Nasser, Bassem (2017). "Medical expert systems survey". In: *International Journal of Engineering and Information Systems (IJEAIS)* 1.7, pp. 218–224.

Adadi, Amina and Mohammed Berrada (2018). "Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)". In: *IEEE access* 6. Publisher: IEEE, pp. 52138–52160.

Akleman, Ergun et al. (2015). "A theoretical framework to represent narrative structures for visual storytelling". In: *Proceedings of bridges 2015: mathematics, Music, art, architecture, culture*, pp. 129–136.

Albouys-Perrois, Jérémy et al. (2022). "Multi-agent simulation of collective self-consumption: Impacts of storage systems and large-scale energy exchanges". In: *Energy and Buildings* 254, p. 111543. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2021.111543. URL: https://www.sciencedirect.com/science/article/pii/S0378778821008276.

Alhafidh, Basman M Hasan et al. (2018). "FPGA hardware implementation of smart home autonomous system based on deep learning". In: *International Conference on Internet of Things*. Springer, pp. 121–133.

Alirezaie, Marjan et al. (2017). "An ontology-based context-aware system for smart homes: E-care@ home". In: *Sensors* 17.7. Publisher: Multidisciplinary Digital Publishing Institute, p. 1586.

Amghar, Souad, Safae Cherdal, and Salma Mouline (2018). "Which NoSQL database for IoT Applications?" In: *2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, pp. 131–137. DOI: 10.1109/MoWNet.2018.8428922.

Antunes, Pedro et al. (2008). "Structuring dimensions for collaborative systems evaluation". In: *ACM computing surveys (CSUR)* 44.2. Publisher: ACM New York, NY, USA, pp. 1–28.

Arrieta, Alejandro Barredo et al. (2020). "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58. Publisher: Elsevier, pp. 82–115.

Augasta, M Gethsiyal and Thangairulappan Kathirvalavakumar (2012). "Reverse engineering the neural networks for rule extraction in classification problems". In: *Neural processing letters* 35.2. Publisher: Springer, pp. 131–150.

Austin, Johanna et al. (2016). "A smart-home system to unobtrusively and continuously assess loneliness in older adults". In: *IEEE journal of translational engineering in health and medicine* 4. Publisher: IEEE, pp. 1–11.

Banerjee, Amit et al. (2018). "Centralized framework for controlling heterogeneous appliances in a smart home environment". In: *2018 International Conference on Information and Computer Technologies (ICICT)*. IEEE, pp. 78–82.

Banzhaf, Wolfgang (2009). "Self-organizing Systems." In: *Encyclopedia of complexity and systems science* 14. Publisher: Citeseer, p. 589.

Barredo-Arrieta, Alejandro and Javier Del Ser (2020). "Plausible Counterfactuals: Auditing Deep Learning Classifiers with Realistic Adversarial Examples". In: *arXiv preprint arXiv:2003.11323*.

Bench-Capon, Trevor JM (2020). "Before and after Dung: Argumentation in AI and Law". In: *Argument & Computation* 11.1-2. Publisher: IOS Press, pp. 221–238.

Bex, Floris and Douglas Walton (2016). "Combining explanation and argumentation in dialogue". In: *Argument & Computation* 7.1. Publisher: IOS Press, pp. 55–68.

Biran, Or and Courtenay Cotton (2017). "Explanation and justification in machine learning: A survey". In: *IJCAI-17 workshop on explainable AI (XAI)*. Vol. 8, p. 1.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Blumreiter, Mathias et al. (2019). "Towards self-explainable cyber-physical systems". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, pp. 543–548.

Bompard, Manuel et al. (May 2020). "Procédé de construction de réseau de neurones pour la simulation de systèmes physiques". WO2020094995. URL: `https://data.inpi.fr/brevets/WO2020094995?q=adagos#WO2020094995`.

Bose, Bimal K (2017). "Power electronics, smart grid, and renewable energy systems". In: *Proceedings of the IEEE* 105.11. Publisher: IEEE, pp. 2011–2018.

Brown, James Robert and Yiftach Fehige (2019). "Thought Experiments". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2019. Metaphysics Research Lab, Stanford University. URL: `https://plato.stanford.edu/archives/win2019/entries/thought-experiment/`.

Burgess, John (2020). "Rtx on—the nvidia turing gpu". In: *IEEE Micro* 40.2. Publisher: IEEE, pp. 36–44.

Burmeister, Daniel, Florian Burmann, and Andreas Schrader (2017). "The smart object description language: modeling interaction capabilities for self-reflection". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, pp. 503–508.

Buzan, Tony and Barry Buzan (2006). *The mind map book*. Pearson Education.

Cai, Carrie J, Jonas Jongejan, and Jess Holbrook (2019). "The effects of example-based explanations in a machine learning interface". In: *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pp. 258–262.

Cambridge Dictionary (2020). *Explanation*. URL: `https://dictionary.cambridge.org/dictionary/english/explanation` (visited on 05/22/2020).

Castelvecchi, Davide (2016). "Can we open the black box of AI?" In: *Nature News* 538.7623, p. 20.

Cave, Stephen and Kanta Dihal (2018). "Ancient dreams of intelligent machines: 3,000 years of robots". In: *Nature* 559.7715. Publisher: Nature Publishing Group, pp. 473–475.

Chaitin, Gregory J (1966). "On the length of programs for computing finite binary sequences". In: *Journal of the ACM (JACM)* 13.4. Publisher: ACM New York, NY, USA, pp. 547–569.

Chaput, Rémy, Amélie Cordier, and Alain Mille (2021). "Explanation for Humans, for Machines, for Human-Machine Interactions?" In: *AAAI-2021, Explainable Agency in Artificial Intelligence WS*.

Chatzimparmpas, Angelos, Rafael M Martins, and Andreas Kerren (2020). "t-viSNE: Interactive Assessment and Interpretation of t-SNE Projections". In: *IEEE transactions on visualization and computer graphics* 26.8. Publisher: IEEE, pp. 2696–2714.

Chomsky, N. (Sept. 1956). "Three models for the description of language". In: *IRE Transactions on Information Theory* 2.3, pp. 113–124. ISSN: 2168-2712. DOI: 10.1109/TIT.1956.1056813.

Confalonieri, Roberto et al. (2019). "An Ontology-based Approach to Explaining Artificial Neural Networks". In: *arXiv preprint arXiv:1906.08362*.

Cook, Andrew A, Göksel Mısırlı, and Zhong Fan (2019). "Anomaly detection for IoT time-series data: A survey". In: *IEEE Internet of Things Journal* 7.7. Publisher: IEEE, pp. 6481–6494.

DARPA (Aug. 2016). *Explainable Artificial Intelligence*. Broad Agency Anouncement DARPA-BAA-16-53. DARPA.

Decker, Keith, Katia Sycara, and Mike Williamson (1997). "Middle-agents for the internet". In: *IJCAI (1)*, pp. 578–583.

Dessalles, Jean-Louis (2008a). "Coincidences and the encounter problem: A formal account". In: *Proceedings of the 30th Annual Conference of the Cognitive Science Society*. Ed. by Bradley C. Love, Ken McRae, and Vladimir M. Sloutsky. Austin, TX: Cognitive Science Society, pp. 2134–2139.

Dessalles, Jean-Louis (2008b). *The Pisa Tower effect*. URL: https://simplicitytheory.telecom-paris.fr/Pisa.html (visited on 03/02/2022).

Dessalles, Jean-Louis (2013). "Algorithmic simplicity and relevance". In: *Algorithmic probability and friends - LNAI 7070*. Ed. by David L. Dowe. Berlin, D: Springer Verlag, pp. 119–130. DOI: 10.1007/978-3-642-44958-1_9.

Dessalles, Jean-Louis (2015). "From conceptual spaces to predicates". In: *Applications of conceptual spaces: The case for geometric knowledge representation*. Ed. by Frank Zenker and Peter Gärdenfors. Dordrecht: Springer, pp. 17–31. DOI: 10.1007/978-3-319-15021-5_2.

Dessalles, Jean-Louis (2016). "A Cognitive Approach to Relevant Argument Generation". In: *Principles and Practice of Multi-Agent Systems, LNAI 9935*. Ed. by Matteo Baldoni, Cristina Baroglio, and Floris Bex. Springer, pp. 3–15.

Dhanorkar, Shipi et al. (2021). "Who needs to know what, when?: Broadening the Explainable AI (XAI) Design Space by Looking at Explanations Across the AI Lifecycle". In: *Designing Interactive Systems Conference 2021*, pp. 1591–1602.

Diaconescu, Ada, Louisa Jane Di Felice, and Patricia Mellodge (2019). "Multi-Scale Feedbacks for Large-Scale Coordination in Self-Systems". In: *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, pp. 137–142.

Diaconescu, Ada, Sylvain Frey, et al. (2016). "Goal-oriented holonics for complex system (self-) integration: Concepts and case studies". In: *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, pp. 100–109.

Diaconescu, Ada, Pembe Mata, and Kirstie Bellman (2018). "Self-integrating organic control systems: from crayfish to smart homes". In: *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*. VDE, pp. 1–8.

Doran, Derek, Sarah Schulz, and Tarek R Besold (2017). "What does explainable AI really mean? A new conceptualization of perspectives". In: *arXiv preprint arXiv:1710.00794*.

Doshi-Velez, Finale and Been Kim (2017). "Towards a rigorous science of interpretable machine learning". In: *arXiv preprint arXiv:1702.08608*.

Doshi-Velez, Finale, Mason Kortz, et al. (2017). "Accountability of AI under the law: The role of explanation". In: *arXiv preprint arXiv:1711.01134*.

Douven, Igor (2021). "Abduction". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2021. Metaphysics Research Lab, Stanford University. URL: `https://plato.stanford.edu/archives/sum2021/entries/abduction/`.

Dung, Phan Minh (1995). "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games". In: *Artificial intelligence* 77.2. Publisher: Elsevier, pp. 321–357.

Ehsan, Upol and Mark Riedl (2020). "Human-centered Explainable AI: Towards a Reflective Sociotechnical Approach". In.

Elias, P. (1975). "Universal codeword sets and representations of the integers". In: *IEEE Transactions on Information Theory* 21.2, pp. 194–203. DOI: `10.1109/TIT.1975.1055349`.

Escoffier, Clément, Richard S Hall, and Philippe Lalanda (2007). "iPOJO: An extensible service-oriented component framework". In: *IEEE International Conference on Services Computing (SCC 2007)*. IEEE, pp. 474–481.

Fadiga, Kanvaly et al. (2021). "To do or not to do: finding causal relations in smart homes". In: *Proceedings of the 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. _eprint: 2105.10058. Online: IEEE, pp. 110–119.

Floridi, Luciano and Massimo Chiriatti (2020). "GPT-3: Its nature, scope, limits, and consequences". In: *Minds and Machines* 30.4. Publisher: Springer, pp. 681–694.

Forouzan, Behrouz A (2002). *TCP/IP protocol suite*. McGraw-Hill Higher Education.

Frey, Sylvain (2013). "Generic architectures for open, multi-objective autonomic systems: application to smart micro-grids". PhD Thesis. Paris, ENST.

Frey, Sylvain, Ada Diaconescu, and Isabelle Demeure (2012). "Architectural integration patterns for autonomic management systems". In: *9th IEEE International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems (EASe 2012)*.

Frey, Sylvain, Ada Diaconescu, David Menga, et al. (2015). "A generic holonic control architecture for heterogeneous multiscale and multiobjective smart microgrids". In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10.2, pp. 1–21.

Friedman, Jerome H (2001). "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics*. Publisher: JSTOR, pp. 1189–1232.

Gärdenfors, Peter (2004). *Conceptual spaces: The geometry of thought*. MIT press.

Gentile, Davide (2021). "Evaluating human understanding in XAI systems". In: CHI 21 Conference.

Ghadakpour, Laleh (2003). "Le système conceptuel, à l'interface entre le langage, le raisonnement et l'espace qualitatif: vers un modèle de représentations éphémères". French. PhD thesis. Palaiseau: École polytechnique.

Gill, T Grandon (1995). "Early expert systems: Where are they now?" In: *MIS quarterly*. Publisher: JSTOR, pp. 51–81.

Ginsberg, Matthew L (1986). "Counterfactuals". In: Publisher: Citeseer.

Glass, Alyssa, Deborah L. McGuinness, and Michael Wolverton (2008). "Toward Establishing Trust in Adaptive Agents". In: *Proceedings of the 13th International Conference on Intelligent User Interfaces*. IUI '08. event-place: Gran Canaria, Spain. New York, NY, USA: Association for Computing Machinery, pp. 227–236. ISBN: 978-1-59593-987-6. DOI: 10.1145/1378773.1378804.

Goldstein, Alex et al. (2015). "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation". In: *journal of Computational and Graphical Statistics* 24.1. Publisher: Taylor & Francis, pp. 44–65.

Gomez, Carles, Joaquim Oller, and Josep Paradells (2012). "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology". In: *Sensors* 12.9. Publisher: Molecular Diversity Preservation International, pp. 11734–11753.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016a). *Deep learning*. MIT press.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016b). "Machine learning basics". In: *Deep learning* 1.7. Publisher: MIT press Cambridge, MA, USA, pp. 98–164.

Grigorescu, Sorin et al. (2020). "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37.3. Publisher: Wiley Online Library, pp. 362–386.

Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, et al. (2018). "Local rule-based explanations of black box decision systems". In: *arXiv preprint arXiv:1805.10820*.

Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Franco Turini, et al. (2018). "A survey of methods for explaining black box models". In: *ACM computing surveys (CSUR)* 51.5. Publisher: ACM New York, NY, USA, pp. 1–42.

Hadwan, H Hamid and YP Reddy (2016). "Smart home control by using raspberry pi and arduino uno". In: *Int. J. Adv. Res. Comput. Commun. Eng* 5.4, pp. 283–288.

Han, Dae-Man and Jae-Hyun Lim (2010). "Design and implementation of smart home energy management systems based on zigbee". In: *IEEE Transactions on Consumer Electronics* 56.3. Publisher: IEEE, pp. 1417–1425.

Hariri, Sahand, Matias Carrasco Kind, and Robert J. Brunner (2021). "Extended Isolation Forest". In: *IEEE Transactions on Knowledge and Data Engineering* 33.4, pp. 1479–1489. DOI: 10.1109/TKDE.2019.2947676.

Harrison, Michael A (1978). *Introduction to formal language theory*. Addison-Wesley Longman Publishing Co., Inc.

Hartigan, John A and Manchek A Wong (1979). "Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the royal statistical society. series c (applied statistics)* 28.1. Publisher: JSTOR, pp. 100–108.

Hawthorne, James (2021). "Inductive Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2021. Metaphysics Research Lab, Stanford University. URL: https://plato.stanford.edu/archives/spr2021/entries/logic-inductive/.

Hempel, Carl G et al. (1965). "Aspects of scientific explanation". In.

Hinchey, Michael G and Roy Sterritt (2006). "Self-managing software". In: *Computer* 39.2. Publisher: IEEE, pp. 107–109.

Hoffman, Robert R and Gary Klein (2017). "Explaining explanation, part 1: theoretical foundations". In: *IEEE Intelligent Systems* 32.3. Publisher: IEEE, pp. 68–73.

Hoffman, Robert R, Shane T Mueller, and Gary Klein (2017). "Explaining explanation, part 2: Empirical foundations". In: *IEEE Intelligent Systems* 32.4. Publisher: IEEE, pp. 78–86.

Horne, Zach, Melis Muradoglu, and Andrei Cimpian (2019). "Explanation as a Cognitive Process". In: *Trends in Cognitive Sciences* 23, pp. 187–199. DOI: 10.1016/j.tics.2018.12.004.

Hossain, Eklas et al. (2019). "Application of big data and machine learning in smart grid, and associated security concerns: A review". In: *Ieee Access* 7. Publisher: IEEE, pp. 13960–13988.

Houzé, Étienne (2021). "A generic and decentralized approach to XAI for autonomic systems: application to the smart home". In: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pp. 301–303. DOI: 10.1109/ACSOS-C52956.2021.00079.

Houzé, Etienne, Jean-Louis Dessalles, et al. (2021). "A Decentralized Explanatory System for Intelligent Cyber-Physical Systems". In: Available on demand.

Houzé, Etienne, Ada Diaconescu, and Jean-Louis Dessalles (Jan. 2022). "Using Decentralized Conflict-Abduction-Negation in Policy-Making". In: JURIX Conference.

Houzé, Etienne, Ada Diaconescu, Jean-Louis Dessalles, and David Menga (2022). "A generic and modular architecture for self-explainable smart homes". In: *Proceedings of the 2022 ACSOS Conference*. Ed. by IEEE.

Houzé, Etienne, Ada Diaconescu, Jean-Louis Dessalles, David Menga, and Mathieu Schumann (Aug. 2020). "A Decentralized Approach to Explanatory Artificial Intelligence for Autonomic Systems". In: *ACSOS 2020 Conference Proceedings, Companion*.

Houzé, Étienne, Jean-Louis Dessalles, et al. (2022). "What Should I Notice? Using Algorithmic Information Theory to Evaluate the Memorability of Events in Smart Homes". In: *Entropy* 24.3. ISSN: 1099-4300. DOI: 10.3390/e24030346. URL: https://www.mdpi.com/1099-4300/24/3/346.

Houzé, Étienne, David Menga, et al. (2021). "Dispositif explicatif du fonctionnement d'un équipement".

Hunkeler, Urs, Hong Linh Truong, and Andy Stanford-Clark (2008). "MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks". In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, pp. 791–798.

Hutter, Marcus (2004). *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media.

IBM Co. (2005). *An architectural blueprintfor autonomic computing*. Tech. rep.

Jain, Sarika and San Murugesan (2021). "Smart Connected World: A Broader Perspective". In: *Smart Connected World*. Springer, pp. 3–23.

Jakobsson, Bjorn Markus and Arthur Kwan Jakobsson (Dec. 2021). "Privacy and the management of permissions".

Janzing, Dominik and Bernhard Schölkopf (2010). "Causal inference using the algorithmic Markov condition". In: *IEEE Transactions on Information Theory* 56.10. Publisher: IEEE, pp. 5168–5194.

Jivani, Farzeem D, Manohar Malvankar, and Radha Shankarmani (2018). "A Voice Controlled Smart Home Solution with a Centralized Management Framework Implemented Using AI and NLP". In: *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*. IEEE, pp. 1–5.

Kafle, Ved P. et al. (2017). "Scalable Directory Service for IoT Applications". In: *IEEE Communications Standards Magazine* 1.3, pp. 58–65. DOI: 10.1109/MCOMSTD.2017.1700027.

Kephart, Jeffrey O and David M Chess (2003). "The vision of autonomic computing". In: *Computer* 36.1. Publisher: IEEE, pp. 41–50.

Kim, Been, Rajiv Khanna, and Oluwasanmi O Koyejo (2016). "Examples are not enough, learn to criticize! criticism for interpretability". In: *Advances in neural information processing systems* 29.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kolmogorov, Andrei Nikolaevich (1965). "Three approaches to the definition of the concept "quantity of information"". In: *Problemy peredachi informatsii* 1.1. Publisher: Russian Academy of Sciences, Branch of Informatics, Computer Equipment and ..., pp. 3–11.

Kounev, Samuel et al. (2017). *Self-Aware Computing Systems*. English. 1st ed. Springer, Cham. ISBN: 978-3-319-47472-4.

Kowalski, Jaros\law et al. (2019). "Older adults and voice interaction: a pilot study with Google Home". In: *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–6.

Kramer, Jeff and Jeff Magee (2009). "A rigorous architectural approach to adaptive software engineering". In: *Journal of Computer Science and Technology* 24.2, pp. 183–188.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25.

Krupitzer, Christian et al. (2015). "A survey on engineering approaches for self-adaptive systems". In: *Pervasive and Mobile Computing* 17. Publisher: Elsevier, pp. 184–206.

Kuzochkina, Anna, Mariya Shirokopetleva, and Zoia Dudar (2018). "Analyzing and Comparison of NoSQL DBMS". In: *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S T)*, pp. 560–564. DOI: 10.1109/INFOCOMMST.2018.8632133.

Ladkin, Peter and Karsten Loer (1998). "Analysing aviation accidents using wb-analysis–an application of multimodal reasoning". In: *AAAI Spring Symposium. AAAI*.

Lalanda, P., E. Gerber-Gaillard, and S. Chollet (2017). "Self-Aware Context in Smart Home Pervasive Platforms". In: *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pp. 119–124. DOI: 10.1109/ICAC.2017.1.

Lalanda, Philippe and Catherine Hamon (2020). "A service-oriented edge platform for cyber-physical systems". In: *CCF Transactions on Pervasive Computing and Interaction* 2.3. Publisher: Springer, pp. 206–217.

Lalanda, Philippe, Julie A McCann, and Ada Diaconescu (2013). *Autonomic computing: principles, design and implementation*. Springer Science & Business Media.

Larasati, Retno, Anna De Liddo, and Enrico Motta (2020). "The Effect of Explanation Styles on User's Trust". In: IUI 2020.

Leake, David B (1995). "Abduction, experience, and goals: A model of everyday abductive explanation". In: *Journal of Experimental & Theoretical Artificial Intelligence* 7.4. Publisher: Taylor & Francis, pp. 407–428.

Leake, David B (2014). *Evaluating explanations: A content theory*. Psychology Press.

Lee, Choonhwa, David Nordstedt, and Sumi Helal (2003). "Enabling smart spaces with OSGi". In: *IEEE Pervasive computing* 2.3, pp. 89–94.

Lewis, David (1987). "Philosophical Papers: Volume 2". In.

Lewis, David (2013). *Counterfactuals*. John Wiley & Sons.

Li, Ming, Paul Vitányi, et al. (2008). *An introduction to Kolmogorov complexity and its applications*. Vol. 3. Springer.

Li, Nianyu et al. (2020). "Explanations for human-on-the-loop: A probabilistic model checking approach". In: *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 181–187.

Lippi, Marco, Stefano Mariani, and Franco Zambonelli (2021). "Developing a "Sense of Agency" in IoT Systems: Preliminary Experiments in a Smart Home Scenario". In: *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, pp. 44–49.

Lipton, Peter (1990). "Contrastive explanation". In: *Royal Institute of Philosophy Supplements* 27. Publisher: Cambridge University Press, pp. 247–266.

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou (2008). "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

Liu, Yang et al. (2017). "Review on cyber-physical systems". In: *IEEE/CAA Journal of Automatica Sinica* 4.1. Publisher: IEEE, pp. 27–40.

Lombrozo, Tania (2012). "Explanation and abductive inference." In: Publisher: Oxford University Press.

Longo, Luca, Bridget Kane, and Lucy Hederman (2012). "Argumentation theory in health care". In: *2012 25th IEEE International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE, pp. 1–6.

Lucas, Salvador (2021). "The origins of the halting problem". In: *Journal of Logical and Algebraic Methods in Programming* 121. Publisher: Elsevier, p. 100687.

Lundberg, Scott M and Su-In Lee (2017). "A unified approach to interpreting model predictions". In: *Advances in Neural Information Processing Systems*, pp. 4765–4774.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

Magnani, Lorenzo (2011). *Abduction, reason and science: Processes of discovery and explanation*. Springer Science & Business Media.

Maher, Mary Lou (1987). "Expert systems for civil engineers: technology and application". In: ASCE.

Manikonda, Lydia, Aditya Deotale, and Subbarao Kambhampati (2018). "What's up with privacy? User preferences and privacy concerns in intelligent personal assistants". In: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 229–235.

Marcus, Gary (2020). *The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence*.

Marikyan, Davit, Savvas Papagiannidis, and Eleftherios Alamanos (2019). "A systematic review of the smart home literature: A user perspective". In: *Technological Forecasting and Social Change* 138. Publisher: Elsevier, pp. 139–154.

Maulud, Dastan and Adnan M Abdulazeez (2020). "A Review on Linear Regression Comprehensive in Machine Learning". In: *Journal of Applied Science and Technology Trends* 1.4, pp. 140–147.

Maxwell, Winston et al. (2020a). "Identifying the'Right'Level of Explanation in a Given Situation". In: *Proceedings of the First International Workshop on New Foundations for Human-Centered AI (NeHuAI), Santiago de Compostella, Spain*, p. 63.

Maxwell, Winston et al. (2020b). "Identifying the'Right'Level of Explanation in a Given Situation". In: *Proceedings of the First International Workshop on New Foundations for Human-Centered AI (NeHuAI), Santiago de Compostella, Spain*, p. 63.

McCarthy, John et al. (Aug. 1955). "A proposal for the dartmouth summer research project on artificial intelligence". In: *AI magazine (reprint)* 27.4, pp. 12–12.

McKeon, Matthew W (2013). "On the rationale for distinguishing arguments from explanations". In: *Argumentation* 27.3. Publisher: Springer, pp. 283–303.

Mekuria, Dagmawi Neway et al. (2019). "Smart home reasoning systems: a systematic literature review". In: *Journal of Ambient Intelligence and Humanized Computing*. Publisher: Springer, pp. 1–18.

Menzies, Peter and Huw Price (1993). "Causation as a secondary quality". In: *The British Journal for the Philosophy of Science* 44.2. Publisher: Oxford University Press, pp. 187–203.

Miller, B.A. et al. (2001). "Home networking with Universal Plug and Play". In: *IEEE Communications Magazine* 39.12, pp. 104–109. DOI: 10.1109/35.968819.

Miller, Tim (2018). "Explanation in artificial intelligence: Insights from the social sciences". In: *Artificial Intelligence*.

Miller, Tim, Piers Howe, and Liz Sonenberg (2017). "Explainable AI: Beware of inmates running the asylum or: How I learnt to stop worrying and love the social and behavioural sciences". In: *arXiv preprint arXiv:1712.00547*.

Mohseni, Sina, Niloofar Zarei, and Eric D Ragan (2021). "A multidisciplinary survey and framework for design and evaluation of explainable AI systems". In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 11.3-4. Publisher: ACM New York, NY, pp. 1–45.

Moor, James (2006). "The Dartmouth College artificial intelligence conference: The next fifty years". In: *Ai Magazine* 27.4, pp. 87–87.

Mueller, Shane T et al. (2019). *Explanation in human-AI systems: A literature meta-review, synopsis of key ideas and publications, and bibliography for explainable AI*. English. Tech. rep. Defense Technical Information Center.

Negri, Elisa et al. (2019). "FMU-supported simulation for CPS digital twin". In: *Procedia manufacturing* 28. Publisher: Elsevier, pp. 201–206.

Nesvijevskaia, Anna et al. (2021). "The accuracy versus interpretability trade-off in fraud detection model". In: *Data & Policy* 3. Publisher: Cambridge University Press.

Nilsson, Nils J and Nils Johan Nilsson (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann.

Ning, Huansheng et al. (2021). "A Survey on Hybrid Human-Artificial Intelligence for Autonomous Driving". In: *IEEE Transactions on Intelligent Transportation Systems*. Publisher: IEEE.

Nomura, Tatsuya and Kayoko Kawakami (2011). "Relationships between robot's self-disclosures and human's anxiety toward robots". In: *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. Vol. 3. IEEE, pp. 66–69.

Norvig, Peter and Stuart Russel (2010). *Artificial Intelligence: a Modern Approach*. 3rd ed. Prentice Hall.

Nothdurft, Florian, Tobias Heinroth, and Wolfgang Minker (2013). "The impact of explanation dialogues on human-computer trust". In: *International Conference on Human-Computer Interaction*. Springer, pp. 59–67.

Papastratis, Ilias (2021). "Introduction to Explainable Artificial Intelligence (XAI)". In: *https://theaisummer.com/*. Accessed: December 2021. URL: `https://theaisummer.com/xai/`.

Park, Eunil et al. (2017). "Comprehensive approaches to user acceptance of Internet of Things in a smart home environment". In: *IEEE Internet of Things Journal* 4.6, pp. 2342–2350.

Patil, Tejashri, Sweta Pandey, and Kajal Visrani (2021). "A review on basic deep learning technologies and applications". In: *Data science and intelligent applications*. Springer, pp. 565–573.

Pearl, Judea (2009). *Causality: Models, Reasoning and Inference*. 2nd. USA: Cambridge University Press. ISBN: 0-521-89560-X.

Pearl, Judea and Dana Mackenzie (2018). *The book of why: the new science of cause and effect*. Basic Books.

Peirce, Charles Sanders (1931). *Collected papers of charles sanders peirce*. Harvard University Press.

Peters, Jonas, Dominik Janzing, and Bernhard Schölkopf (2017). *Elements of causal inference: foundations and learning algorithms*. MIT press.

Popper, Karl R (1963). "Science as falsification". In: *Conjectures and refutations* 1. Publisher: Routledge and Keagan Paul London, pp. 33–39.

Rahman, S and O Hazim (1996). "Load forecasting for multiple sites: development of an expert system-based technique". In: *Electric Power Systems Research* 39.3. Publisher: Lausanne [Switzerland]: Elsevier Sequoia, 1977-, pp. 161–170.

Rajani, Nazneen Fatema and Raymond J. Mooney (2018). "Ensembling Visual Explanations". In: *Explainable and Interpretable Models in Computer Vision and Machine Learning*. Ed. by Hugo Jair Escalante et al. Cham: Springer International Publishing, pp. 155–172. ISBN: 978-3-319-98131-4. DOI: 10.1007/978-3-319-98131-4_7. URL: https://doi.org/10.1007/978-3-319-98131-4_7.

Rajkumar, Ragunathan et al. (2010). "Cyber-physical systems: The next computing revolution". In: *Design Automation Conference*, pp. 731–736. DOI: 10.1145/1837274.1837461.

Ras, Gabriëlle, Marcel van Gerven, and Pim Haselager (2018). "Explanation methods in deep learning: Users, values, concerns and challenges". In: *Explainable and interpretable models in computer vision and machine learning*. Springer, pp. 19–36.

Rathi, Shubham (2019). "Generating counterfactual and contrastive explanations using SHAP". In: *arXiv preprint arXiv:1906.09293*.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). "Why should I trust you?: Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp. 1135–1144.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2018). "Anchors: High-precision model-agnostic explanations". In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ricquebourg, Vincent et al. (2006). "The smart home concept: our immediate future". In: *2006 1st IEEE international conference on e-learning in industrial electronics*. IEEE, pp. 23–28.

Rojat, Thomas et al. (2021). "Explainable Artificial Intelligence (XAI) on TimeSeries Data: A Survey". In: *arXiv preprint arXiv:2104.00950*.

Rubin, David C and Sharda Umanath (2015). "Event memory: A theory of memory for laboratory, autobiographical, and fictional events." In: *Psychological review* 122.1. Publisher: American Psychological Association, p. 1.

Rudin, Cynthia (2019). "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature Machine Intelligence* 1.5. Publisher: Nature Publishing Group, pp. 206–215.

Safaric, Stanislav and Kresimir Malaric (2006). "ZigBee wireless standard". In: *Proceedings ELMAR 2006*, pp. 259–262. DOI: 10.1109/ELMAR.2006.329562.

Safavian, S Rasoul and David Landgrebe (1991). "A survey of decision tree classifier methodology". In: *IEEE transactions on systems, man, and cybernetics* 21.3. Publisher: IEEE, pp. 660–674.

Salehie, Mazeiar and Ladan Tahvildari (2009). "Self-adaptive software: Landscape and research challenges". In: *ACM transactions on autonomous and adaptive systems (TAAS)* 4.2. Publisher: ACM New York, NY, USA, pp. 1–42.

Schlegel, Udo et al. (2019). "Towards a rigorous evaluation of xai methods on time series". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, pp. 4197–4201.

Searle, John R (1980). "Minds, brains, and programs". In: *Behavioral and brain sciences* 3.3. Publisher: Cambridge University Press, pp. 417–424.

Selvaraju, Ramprasaath R et al. (2017). "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*, pp. 618–626.

Shannon, Claude Elwood (1948). "A mathematical theory of communication". In: *The Bell system technical journal* 27.3. Publisher: Nokia Bell Labs, pp. 379–423.

Shapley, Lloyd Stowell (1953). "A Value for n-Person Games". In: *Contribution to the Theory of Games* 2, pp. 303–317.

Shi, Weisong et al. (2016). "Edge computing: Vision and challenges". In: *IEEE internet of things journal* 3.5, pp. 637–646.

Siekkinen, Matti et al. (2012). "How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4". In: *2012 IEEE wireless communications and networking conference workshops (WCNCW)*. IEEE, pp. 232–237.

Silver, David et al. (2017). "Mastering the game of go without human knowledge". In: *Nature* 550.7676. Publisher: Nature Publishing Group, pp. 354–359.

Silverio-Fernández, Manuel, Suresh Renukappa, and Subashini Suresh (2018). "What is a smart device?-a conceptualisation within the paradigm of the internet of things". In: *Visualization in Engineering* 6.1. Publisher: Springer, pp. 1–10.

Software, TibCo (2020). *The Flogo Project*. URL: `https://www.flogo.io/` (visited on 02/27/2022).

Sokol, Kacper and Peter Flach (2020). "Explainability fact sheets: a framework for systematic assessment of explainable approaches". In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pp. 56–67.

Solomonoff, Ray J (1964). "A formal theory of inductive inference. Part I". In: *Information and control* 7.1. Publisher: Elsevier, pp. 1–22.

Sukor, Abdul Syafiq Abdull et al. (2018). "Semantic knowledge base in support of activity recognition in smart home environments". In: *International Journal of Engineering and Technology* 7.4. Publisher: Science Publishing Corporation, pp. 67–72.

Sun, Zhaohao, Gavin Finnie, and Klaus Weber (2005). "Abductive case-based reasoning". In: *International Journal of Intelligent Systems* 20.9. Publisher: Wiley Online Library, pp. 957–983.

Suykens, Johan AK and Joos Vandewalle (1999). "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3. Publisher: Springer, pp. 293–300.

SWALE Project (1992). `https://homes.luddy.indiana.edu/leake/projects/swale/`.

Swartout, William R (1983). "XPLAIN: A system for creating and explaining expert consulting programs". In: *Artificial intelligence* 21.3. Publisher: Elsevier, pp. 285–325.

Tan, Sarah et al. (2018). "Distill-and-compare: Auditing black-box models using transparent model distillation". In: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 303–310.

Tan, Ying, Mehmet C. Vuran, and Steve Goddard (2009). "Spatio-Temporal Event Model for Cyber-Physical Systems". In: *2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, pp. 44–50. DOI: `10.1109/ICDCSW.2009.82`.

Tavares, Andre L.C. and Marco Tulio Valente (Aug. 2008). "A Gentle Introduction to OSGi". In: *SIGSOFT Softw. Eng. Notes* 33.5. Place: New York, NY, USA Publisher: Association for Computing Machinery. ISSN: 0163-5948. DOI: 10.1145/1402521.1402526. URL: https://doi.org/10.1145/1402521.1402526.

Tintarev, Nava and Judith Masthoff (2011). "Designing and Evaluating Explanations for Recommender Systems". In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 479–510. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_15.

Trout, Joseph D (2002). "Scientific explanation and the sense of understanding". In: *Philosophy of Science* 69.2, pp. 212–233.

Tsilionis, Konstantinos et al. (2021). "Conceptual Modeling Versus User Story Mapping: Which is the Best Approach to Agile Requirements Engineering?" In: *International Conference on Research Challenges in Information Science*. Springer, pp. 356–373.

Turing, Alan M. (1950). "Computing Machinery and Intelligence". In: *Mind* 59, pp. 433–460.

Valtolina, Stefano et al. (2019). "Facilitating the development of iot applications in smart city platforms". In: *International Symposium on End User Development*. Springer, pp. 83–99.

Van Bouwel, Jeroen and Erik Weber (2002). "Remote causes, bad explanations?" In: *Journal for the Theory of Social Behaviour* 32.4. Publisher: Wiley Online Library, pp. 437–449.

Verma, Sahil, John Dickerson, and Keegan Hines (2020). "Counterfactual explanations for machine learning: A review". In: *arXiv preprint arXiv:2010.10596*.

Verma, Sahil, John P. Dickerson, and Keegan Hines (May 2021). "Counterfactual Explanations for Machine Learning: Challenges Revisited". In: *CHI21 Extended Proceedings*. Yokohama, Japan: ACM.

Wachter, Sandra, Brent Mittelstadt, and Chris Russell (2017). "Counterfactual explanations without opening the black box: Automated decisions and the GDPR". In: *Harv. JL & Tech.* 31, p. 841.

Welsh, Kris et al. (2014). "Self-explanation in adaptive systems based on runtime goal-based models". In: *Transactions on Computational Collective Intelligence XVI*. Springer, pp. 122–145.

Wen, Xinlong and Yunliang Wang (2018). "Design of smart home environment monitoring system based on raspberry Pi". In: *2018 Chinese Control And Decision Conference (CCDC)*, pp. 4259–4263. DOI: 10.1109/CCDC.2018.8407864.

Wenbo, Yan, Wang Quanyu, and Gao Zhenwei (2015). "Smart home implementation based on Internet and WiFi technology". In: *2015 34th Chinese Control Conference (CCC)*. IEEE, pp. 9072–9077.

Weyns, Danny (2019). "Software Engineering of Self-adaptive Systems". In: *Handbook of Software Engineering*. Ed. by Sungdeok Cha, Richard N. Taylor, and Kyochul Kang. Cham: Springer International Publishing, pp. 399–443. ISBN: 978-3-030-00262-6. DOI: 10.1007/978-3-030-00262-6_11.

Weyns, Danny, Sam Malek, and Jesper Andersson (2010). "On decentralized self-adaptation: lessons from the trenches and challenges for the future". In: *Proceedings of the 2010*

ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, pp. 84–93.

Weyns, Danny, Bradley Schmerl, et al. (2013). "On patterns for decentralized control in self-adaptive systems". In: *Software Engineering for Self-Adaptive Systems II*. Springer, pp. 76–107.

Wing, Jeannette M. (2020). *Trustworthy AI*.

Winograd, Terry (1972). "Understanding natural language". In: *Cognitive psychology* 3.1. Publisher: Elsevier, pp. 1–191.

Woodward, James (2019). "Scientific Explanation". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2019. Metaphysics Research Lab, Stanford University. URL: `https://plato.stanford.edu/archives/win2019/entries/scientific-explanation/`.

Wu, Chao-Lin, Chun-Feng Liao, and Li-Chen Fu (2007). "Service-oriented smart-home architecture based on OSGi and mobile-agent technology". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.2. Publisher: IEEE, pp. 193–205.

Yang, Heetae, Wonji Lee, and Hwansoo Lee (2018). "IoT smart home adoption: The importance of proper level automation". In: *Journal of Sensors* 2018. Publisher: Hindawi.

Yosinski, Jason et al. (2015). "Understanding neural networks through deep visualization". In: *arXiv preprint arXiv:1506.06579*.

Zablocki, Éloi et al. (2021). "Explainability of vision-based autonomous driving systems: Review and challenges". In: *arXiv preprint arXiv:2101.05307*.

Zhang, Zhongheng (2016). "Introduction to machine learning: k-nearest neighbors". In: *Annals of translational medicine* 4.11. Publisher: AME Publications.

Zhou, Bin et al. (2016). "Smart home energy management systems: Concept, configurations, and scheduling strategies". In: *Renewable and Sustainable Energy Reviews* 61. Publisher: Elsevier, pp. 30–40.

Zimmermann, Gottfried, Tobias Ableitner, and Christophe Strobbe (2017). "User Needs and Wishes in Smart Homes: What Can Artificial Intelligence Contribute?" In: *2017 14th International Symposium on Pervasive Systems, Algorithms and Networks*. IEEE, pp. 449–453.

ECOLE
DOCTORALE

Titre : Une approche générique et adaptative de l'IA Explicable pour les systèmes autonomiques: le cas de la maison intelligente

Mots clés : IA Explicable, Maison Intelligente, Système Autonomique

Résumé : Les maisons intelligentes sont des systèmes cyber-physiques dans lesquels de nombreux composants intéragissent les uns avec les autres pour accomplir des objectifs de haut niveau comme le confort ou la sécurité de l'occupant. Ces systèmes autonomiques sont capables de s'adapter sans demander d'intervention de la part de l'utilisateur: ce fonctionnement autonomique est difficile à comprendre pour l'occupant. Ce manque d'explicabilité peut être un frein à l'adoption plus large de tels systèmes.

Depuis le milieu des années 2010, l'explicabilité des modèles complexes d'IA est devenue un sujet de recherche important. La difficulté à expliquer les systèmes autonomiques ne vient pas de la complexité des composants, mais plutôt de leur capacité d'adaptation qui peut entraîner des changements de configurations, de logique ou d'objectifs. Par ailleurs, l'hétérogénéité des dispositifs présents dans une maison intelligente complique la tâche.

Afin de répondre à ces difficultés, nous proposons d'ajouter à un système autonomique de maison intelligente un système explicatif dont le rôle sera d'observer les différents contrôleurs, capteurs et équipements présents pour générer des explications à l'occupant. Nous définissons six objectifs pour un tel système. 1) Produire des explications contrastives, c'est-à-dire qui visent les situations inattendues ou non voulues. 2) Produire des explications peu profondes, dont les éléments sont causalement proches. 3) Être transparent: exposer son raisonnement et quels composants sont impliqués dans le processus. 4) Être capable de réflexivité, d'exposer ses propres états et changement d'état comme explications à un phénomène. 5) Être générique, pouvoir s'adapter à des composants et des architectures de systèmes autonomiques variées. 6) Respecter la protection des données et favoriser les traitement local, au plus près du capteur.

Notre réflexion a abouti à un système explicatif dans lequel un composant central, nommé le "Spotlight" est guidé par l'algorithme D-CAS qui identifie trois éléments dans le processus: la détection de conflits par interprétation des observations, la propagation par abduction et la simulation de conséquences possibles. Ces trois étapes sont réalisées par des composants explicatifs locaux qui sont tour à tour interrogés par le Spotlight. Chaque composant local est relié à un capteur, actionneur ou contrôleur du système autonomique, et agit comme un expert dans le domaine associé. Cette organiasation permet l'ajout de nouveaux composants, intégrant leurs connaissances au sein du système global sans demander de reconfiguration. Nous illustrons ce fonctionnement en réalisant un prototype générant des explications sur des cas typiques.

Nous proposons que les composants explicatifs locaux soient des plateformes génériques pouvant être spécialisées par l'ajout de modules dont nous définissons les interfaces. Cette modularité permet d'intégrer des techniques diverses d'interprétation, d'abduction et de simulation. Notre système visant particulièrement des situations inhabituelles pour lesquelles les données peuvent être rares, les méthodes d'abduction basées sur les occurences passées sont inapplicables. Nous proposons une approche nouvelle : estimer la mémorabilité des événements afin d'utiliser les plus notables comme hypothèses pertinentes à un phénomène surprenant.

Notre approche de haut niveau de l'explicabilité a une visée générique, et pose les bases pour des systèmes intégrant des modules plus avancés, permettant de garantir l'explicabilité d'une maison intelligente, mais aussi d'autres systèmes cyber-physiques.

Title : A generic and adaptive approach to Explainable AI in autonomic systems: the case of the smart home.

Keywords : Explainable AI, Smart Home, Autonomic System

Abstract : Smart homes are Cyber-Physical Systems where various components cooperate to fulfill high-level goals such as user comfort or safety. These autonomic systems can adapt at runtime without requiring human intervention. This adaptation is hard to understand for the occupant, which can hinder the adoption of smart home systems.

Since the mid 2010s, explainable AI has been a topic of interest, aiming to open the black box of complex AI models. The difficulty to explain autonomic systems does not come from the intrinsic complexity of their components, but rather from their self-adaptation capability which leads changes of configuration, logic or goals at runtime. In addition, the diversity of smart home devices makes the task harder.

To tackle this challenge, we propose to add an explanatory system to the existing smart home autonomic system, whose task is to observe the various controllers and devices to generate explanations. We define six goals for such a system. 1) To generate contrastive explanations in unexpected or unwanted situations. 2) To generate a shallow reasoning, whose different elements are causaly closely related to each other. 3) To be transparent, i.e. to expose its entire reasoning and which components are involved. 4) To be self-aware, integrating its reflective knowledge into the explanation. 5) To be generic and able to adapt to diverse components and system architectures. 6) To preserve privacy and favor locality of reasoning.

Our proposed solution is an explanatory system in which a central component, name the “Spotlight”, implements an algorithm named D-CAS. This algorithm identifies three elements in an explanatory process: conflict detection via observation interpretation, conflict propagation via abductive inference and simulation of possible consequences. All three steps are performed locally, by Local Explanatory Components which are sequentially interrogated by the Spotlight. Each Local Component is paired to an autonomic device or controller and act as an expert in the related knowledge domain. This organization enables the addition of new components, integrating their knowledge into the general system without need for reconfiguration. We illustrate this architecture and algorithm in a proof-of-concept demonstrator that generates explanations in typical use cases.

We design Local Explanatory Components to be generic platforms that can be specialized by the addition of modules with predefined interfaces. This modularity enables the integration of various techniques for abduction, interpretation and simulation. Our system aims to handle unusual situations in which data may be scarce, making past occurrence-based abduction methods inoperable. We propose a novel approach: to estimate events memorability and use them as relevant hypotheses to a surprising phenomenon.

Our high-level approach to explainability aims to be generic and paves the way towards systems integrating more advanced modules, guaranteeing smart home explainability. The overall method can also be used for other Cyber-Physical Systems.

Institut Polytechnique de Paris
91120 Palaiseau, France