2023

Abductive Design of BDI Agent-based Digital Twins of Organizations

Ahmad Ali Meqbel Alelaimat

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Abductive Design of BDI Agent-based Digital Twins of Organizations

Ahmad Ali Meqbel Alelaimat

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy

Supervisor:
Prof. Aditya Ghose

Co-supervisor:
Prof. Hoa Khanh Dam

The University of Wollongong
School of Computing and Information Technology

July, 2023

This research has been conducted with the support of an Australian Government Research Training Program Scholarship.

Declaration

I, *Ahmad Ali Meqbel Alelaimat*, declare that this thesis is submitted in partial fulfilment of the requirements for the conferral of the degree *Doctor of Philosophy*, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Ahmad Ali Meqbel Alelaimat

July 28, 2023

Abstract

For a Digital Twin - a precise, virtual representation of a physical counterpart - of a human-like system to be faithful and complete, it must appeal to a notion of anthropomorphism (i.e., attributing human behaviour to non-human entities) to imitate (1) the externally visible behaviour and (2) the internal workings of that system. Although the Belief-Desire-Intention (BDI) paradigm was not developed for this purpose, it has been used successfully in human modeling applications. In this sense, we introduce in this thesis the notion of abductive design of BDI agent-based Digital Twins of organizations, which builds on two powerful reasoning disciplines: reverse engineering (to recreate the visible behaviour of the target system) and goal-driven eXplainable Artificial Intelligence (XAI) (for viewing the behaviour of the target system through the lens of BDI agents). Precisely speaking, the overall problem we are trying to address in this thesis is to "Find a BDI agent program that best explains (in the sense of formal abduction) the behaviour of a target system based on its past experiences". To do so, we propose three goal-driven XAI techniques: (1) abductive design of BDI agents, (2) leveraging imperfect explanations and (3) mining belief-based explanations. The resulting approach suggests that using goal-driven XAI to generate Digital Twins of organizations in the form of BDI agents can be effective, even in a setting with limited information about the target system's behaviour.

Dedication

To the memory of Professor Aditya Ghose. (1968-2023)

Acknowledgements

Academics often refer to a doctoral thesis as a solo endeavour. However, the acknowledgement below definitely tells the opposite.

At every stage of this research, the late Prof. Aditya Ghose was not just a patient and insightful supervisor, but also a very kind and generous advisor. Perhaps, much of my gratitude goes to Prof. Aditya Ghose for two reasons: (1) believing in someone with little background (after being a man-at-arms for many years) in AI and (2) introducing me to the field of intelligent agents and process analytics. My gratitude also goes to Prof. Hoa Khanh Dam for his constructive, helpful and valuable assistance.

For the past six years, my lovely wife - Ph. Elham Ahmad - was there, making sure that everything flowed smoothly. I acknowledge that this work would not have been possible without her dedicated support. For his continuous and unconditional support, I would like to thank my father - Prof. Ali Alelaimat. Undoubtedly, this journey would not have been started nor completed without the prayers of my lovely mother - Nawal Ahmad.

List of Publications

- Alelaimat, A., Ghose, A. and Dam, H.K., 2021. Abductive design of BDI agent-based digital twins of organizations. In PRIMA 2020: Principles and Practice of Multi-Agent Systems: 23rd International Conference, Nagoya, Japan, November 18–20, 2020, Proceedings 23 (pp. 377-385). Springer International Publishing. [Chapter 3]

- Alelaimat, A., Ghose, A. and Dam, H.K., 2023. Leveraging Imperfect Explanations for Plan Recognition Problems. To appear in Explainable, Transparent Autonomous Agents and Multi-Agent Systems: Fifth International Workshop, EXTRAAMAS 2023, London, UK. [Chapter 4]

- Alelaimat, A., Ghose, A. and Dam, H.K., 2023. Mining and Validating Belief-based Agent Explanations. To appear in Explainable, Transparent Autonomous Agents and Multi-Agent Systems: Fifth International Workshop, EXTRAAMAS 2023, London, UK. [Chapter 5]

- Alelaimat, A., Ghose, A. and Dam, H.K., 2023. XPlaM: A toolkit for automating the acquisition of BDI agent-based Digital Twins of organizations. Computers in Industry, 145, p.103805. [Chapter 6]

- Alelaimat, A., Santipuri, M., Gou, Y. and Ghose, A., 2018. Learning Planning Model for Semantic Process Compensation. In Service Research and Innovation: 5th and 6th Australasian Symposium, ASSRI 2015 and ASSRI 2017, Sydney, NSW, Australia, November 2–3, 2015, and October 19–20, 2017, Revised Selected Papers 5 (pp. 35-47). Springer International Publishing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital Twin technology - a precise, virtual representation of a physical device or system - has been the subject of considerable attention over the past decade. A relatively recent survey made by Gartner, Inc. on 200 firms has found that 48% of the survey sample are using or planning to use Digital Twins in the foreseeable future based in the USA [1]. Another recent survey conducted by Altair Engineering Inc. on 2000 firms has reported that 69% of the firms are now leveraging Digital Twin technology [2]. HVM Catapult research centre has reported that the Digital Twin market will grow to over $15 billion by 2023 in the field of manufacturing based in the United Kingdom [3]. Major companies already leverage Digital Twin technology. For example, NASA uses Digital Twins for continuous monitoring of its spaceflights [4]. General Electric employs Digital Twin technology to track the operations of electric utilities (e.g., wind turbines) [5]. Digital Twin platforms also enjoy popularity among technology corporations, such as Azure Digital Twin [6] and Oracle IoT Digital Twin [7]. While considerable attention has been paid to the development of Digital Twins for physical devices and systems, the question of developing twins for organizations has received relatively little attention.

A large body of literature in management anthropomorphises (i.e., attributing human behaviour to non-human entities) the organization, e.g., [8–10]. Indeed, real-life organizations are organic (i.e., social systems) to a large degree since they are, above all, an assembly of personalities. Relatively few works on Digital Twins (e.g., [11–13]) submit that for a Digital Twin of human-like systems to be complete and faithful, it must appeal to a notion of anthropomorphism to imitate (1)

the structured processes and (2) the internal workings of the target system. It makes sense, therefore, to concentrate on the beliefs of an organization and its goals and intentions (i.e., internal workings) when developing Digital Twins of organizations. A software agent with Belief-Desire-Intention (BDI) architecture [14] is arguably one of the most sophisticated agent architectures on offer, permitting the modelling of organizations using anthropomorphic handles. Our choice of BDI agent technology as the implementation vehicle thus enables some modicum of adaptation capability, context-sensitivity and autonomy in the resulting Digital Twins. While the target systems might not exhibit these attributes in an obvious fashion, the act of viewing them through the lens of BDI agents allows us to model, implement and analyse these using an anthropomorphic vocabulary. As the literature [15] suggests, this is an elegant means of dealing with systems with considerable underlying complexity. It is also generally submitted that there is no unified template for organizations. Hence, it is not our intention to try to argue that we can build complete and faithful Digital Twins of organizations. Rather, we aim to provide Digital Twin developers with some “first cut" description (also known as the “start with what you have" principle to build organizational Digital Twins [13]) of the behavioural aspects of an organization. However, even with “rough cuts", hand-crafting Digital Twins, which is the ongoing norm, comes with several obvious limitations. The time and effort involved in crafting a Digital Twin lead to bottlenecks.

Digital Twins, therefore, have to be “re-crafted” in response to every change in the target system or the operating context [16]. The setting in which we address this problem is very general and, consequently, very challenging. We look at automating the acquisition of Digital Twins of organizations without loss of generality to physical and cyber-physical systems. This, in turn, opens up important new classes of applications, such as competitor modeling, where our contribution to Digital Twin acquisition enables the building of effective twins, even in a setting where there is limited information available about competitor behaviour. A major assumption guiding the state-of-the-art Digital Twins is collaborative target systems, which allows developers to design Digital Twins under the assumption that the target system would disclose its characteristics at any given time to Digi-

tal Twin developers. Rather, we view the target system as neither cooperative nor unobtrusive with the observant (aka keyhole settings).

Our overall goal in this thesis is to address the above-cited challenges. To do so, we propose a number of goal-driven XAI techniques aiming to explain the behaviour of anthropomorphic systems and, consequently, recraft that behaviour through the lens of BDI agents.

1.1 Research Questions

As they are typically a collection of people, it is our perspective that organizations are "organic" to a large degree. Hence, our focus here is not only to build a Digital Twin of the externally visible behaviour of the organization but also of its internal workings, for which we choose the BDI agent technology as an implementation vehicle. During this thesis, we sought to address the general research question of how to design first cut Digital Twins of organizations based on BDI ontology. To that end, the first research question we seek to address is:

> **RQ1**: What is an end-to-end methodology for the development of BDI agent-based Digital Twins?

In competitive settings, where our research is situated, the only readily available inputs one may think of are (1) the externally visible behaviour of the target organization and (2) common patterns of organization behaviour. However, observed behaviours are often not from the set of common patterns [17]. For example, business environments typically require dynamic execution of plans where organizations must engage in behaviour that includes, for example, re-planning, plans reusing, plan repair, etc. Hence, the second research question we would like to address is:

> **RQ2**: How can potential plans be leveraged to explain new classes of observations?

Explaining the internal workings of anthropomorphic systems may involve some ambiguity (e.g, one decision having more than one possible explanation) and irrationality (e.g., different decisions having one explanation) in competitive settings.

Arguably, a sufficiently detailed explanation (i.e., one that justifies an action with extensive information about the internal reasoning of the target system) can contribute to an accurate justification of the decisions of anthropomorphic systems. Hence, the third question we seek to address is:

> **RQ3**: How to leverage the historical data associated with the business environment to provide detailed explanations about the organization's actions?

It is generally acknowledged that the design of Digital Twins involves considerable investment in terms of effort and time , which would be multiplied if there was an additional obligation to do so in a keyhole setting. Thus, Digital Twins have to be "re-crafted" in response to every change in the target system or the operating context [16]. Our fourth research question thus focuses on providing this kind of support.

> **RQ4**: Is it possible to build a toolkit to support programmers in developing first-cut Digital Twins of organizations and use it as a basis for evaluating our approach?

Note that addressing **RQ4** is strongly related to the three above-cited research questions. Particularly, answering **RQ4** is closely linked to the question of evaluating the generated Digital Twins, including the proposed answers to the above-cited research questions.

1.2 Research Main Contributions

During this thesis, we sought to develop a body of contributions that is able to "Find a BDI agent program that best explains the behaviour of a target system on the basis of its past experiences". To that end, we leverage two powerful reasoning strategies: reverse engineering (to re-create the visible behaviour of the target system) and goal-driven XAI (for viewing the behaviour of the target system through the lens of BDI agents). We shall refer to this notion of re-crafting as the abductive design of BDI agents.

Our overall contribution is to leverage the externally observable behaviour of the target system and then generate candidate BDI agent programs that best explain (in the sense of formal abduction) the observed behaviour. The candidate

agent programs are generated by searching through potentially large hypotheses spaces for possible plans, selection functions and beliefs. This framework is depicted in Figure 1.1.

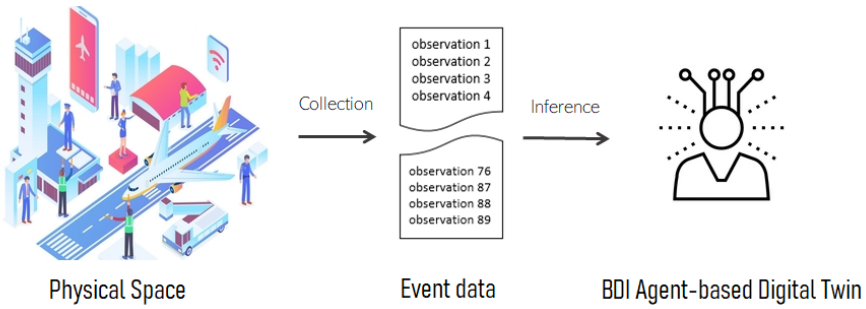


Figure 1.1: Abductive design of BDI agent-based Digital Twins

The key contributions of this thesis can be summarized as follows:

1. Our first contribution is the abductive design of BDI-based Digital Twins, a pragmatic approach to building Digital Twins of organizations through the lens of BDI agents. The overall approach is to leverage the externally observable behaviour of the target system and then generate candidate BDI agent programs that best explain (in the sense of formal abduction) the observed behaviour. Our abductive design gives a complete account of the BDI handles (e.g., plans, beliefs, desires and intentions), which is implemented through the following means (1) abductive plan recognition, (2) explaining by beliefs, and (3) explaining by selection strategies. Our evaluation of the abductive design of BDI-based Digital Twins using two synthetic yet realistic observation logs suggests the effectiveness of this contribution to addressing **RQ1**.

2. The second contribution of the thesis is leveraging (in the sense of plan editing) imperfect explanations, the task of modifying existing hypotheses (i.e., potential plans) to explain new classes of observations. We show that when the target organization operates in a domain model known to the observer, imperfect explanations can be a valuable guide to explain unknown plans that involve new classes of observations. Hence, this contribution can be seen as a post-processing stage for various plan library-based plan recognition techniques. To avoid arbitrary leveraging of hypotheses, we also intro-

duce a classification model that can determine the settings (e.g., noisy or explanatory) in which an unknown plan has been observed. The performance of the proposed approach using the Monroe Plan Corpus [19] demonstrates the applicability of this contribution to answering **RQ2**.

3. Our third contribution is mining belief-based agent explanations, a data-driven approach to mining and validating explanations (specifically belief-based explanations) of the actions of the target system. Our approach makes use of the historical data associated with the target system execution, which describes action execution events and external events (represented as beliefs). We have employed an association rule miner to discover regularities between external events and actions of the target system. Also, we developed a state update operator (i.e., an operator that defines how the specification of a belief state is updated as a consequence of the system's perception of the environment) to contextualise the explanation (i.e., provide users with detailed explanations) and validate the discovered explanations. Our evaluation suggests that mining belief-based explanations can answer **RQ3**.

4. Our fourth contribution is the XPlaM (eXplainable Plan Miner) toolkit, developed to support users in the design of Digital Twins of organizations without loss of generality to other anthropomorphic systems. XPlaM builds on the following two premises: (1) Digital Twins should be re-crafted in response to every change in the target system or the operating context [16] and (2) Hand-crafting Digital Twins, which is the current norm, comes with several obvious limitations (e.g., time and effort). The setting in which XPlaM has developed is very challenging. We look at the automatic acquisition of Digital Twins in competitive environments (i.e., with very little insight into exactly how the target system carries out its tasks). XPlaM provides three explanation techniques in the form of plugins, each representing the implementation of the above-cited contributions. XPlaM has been developed to answer **RQ4**.

1.3 Structure of this Thesis

This section provides a summary of this thesis and the general structure of its chapters. A brief overview of the thesis structure is depicted in Figure 1.2.

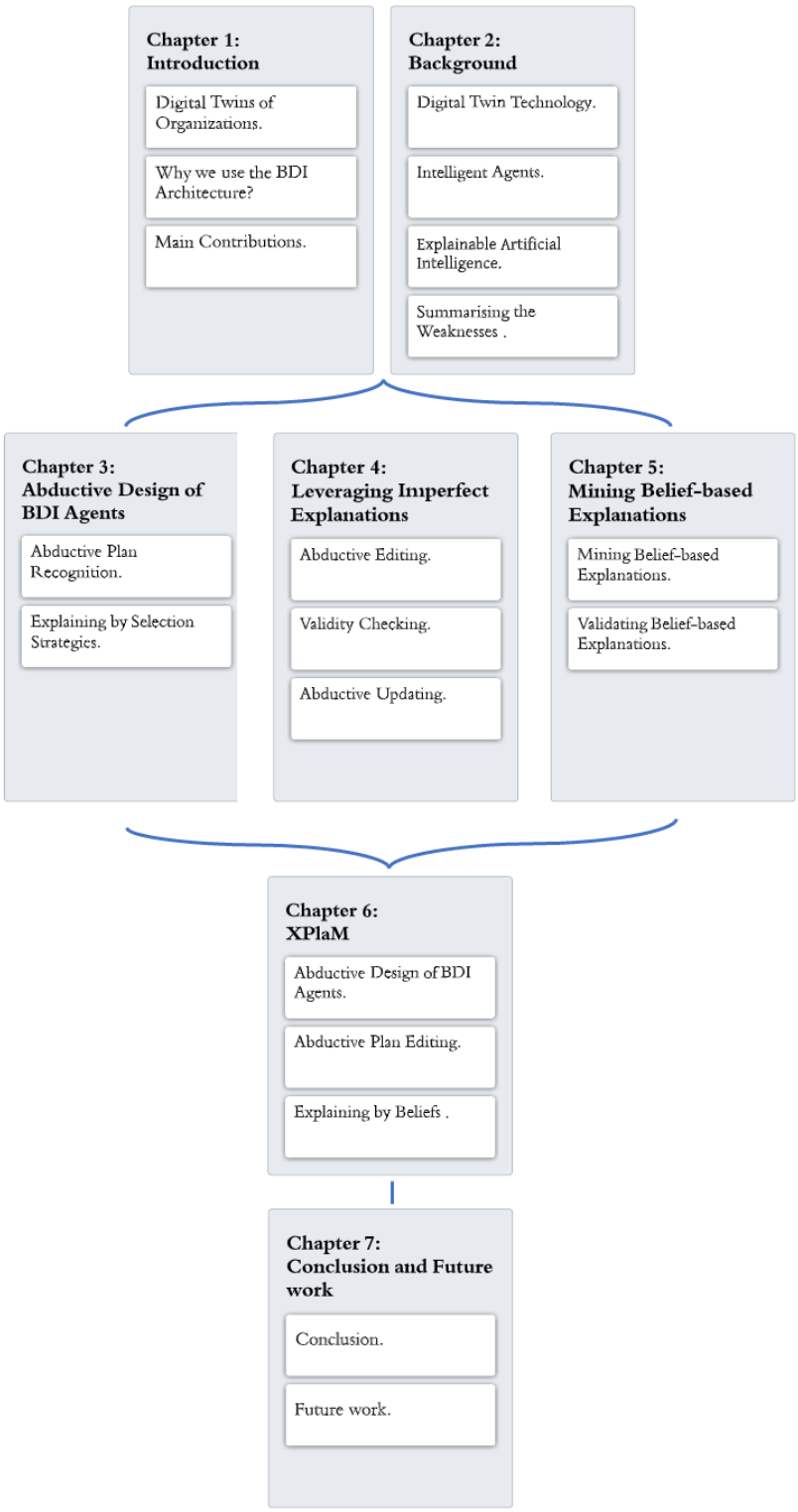


Figure 1.2: Structure of this thesis

Figure 1.2 can be detailed as follows:

- Chapter 2 reviews the state-of-the-art Digital Twin technology focusing on its related concepts, applications and simulation modeling techniques. The chapter also provides a background on intelligent agents (specifically BDI agents) along with an overview of one specific agent-based programming language (AgentSpeak(L) [20]). It also presents a brief introduction to XAI (specifically goal-driven XAI). Recall that this thesis lies at the intersection of these three research areas. Finally, this chapter concludes by outlining gaps in the literature

- Chapter 3 addresses the problem of the abductive design of Digital Twins based on the BDI agent ontology (specifically AgentSpeak(L) programming language [20]) by the following means (1) abductive plan recognition, (2) explaining by beliefs, and (3) explaining by selection strategies.

- Chapter 4 addresses the problem of leveraging imperfect explanations. This chapter focuses on improving the explanatory power of plan libraries by leveraging imperfect explanations and exploiting new classes of observations. Hence, it can be seen as a post-processing technique to the plan recognition mechanism presented in chapter 3.

- Chapter 5 introduces and addresses the problem of mining belief-based explanations. This chapter also shows how to obtain detailed explanations in situations when the observed behaviour cannot be explained rationally by other BDI handles. Hence, it can be seen as a complementary technique to explaining by selection strategies presented in chapter 3.

- Chapter 6 describes how the algorithms and the conceptual components presented in chapters 3, 4 and 5 have been implemented. The implemented tool has been named XPlaM ("eXplaniable Plan Miner). XPlaM provides five techniques in the form of plugins, three of which are related to the explanation techniques offered in the thesis, which are (1) Abductive design of BDI agents, (2) Abductive plan editing, and (3) Explaining by beliefs.

- Chapter 7 summarizes the contributions of this thesis and discusses the possible future work.

Chapter 2

Background

The main goal of this chapter is to (1) briefly introduce the landscape of Digital Twin technology, (2) give an overview of the computational model that we use as the implementation vehicle for Digital Twins development (i.e., the BDI model) and (3) provide some background on eXplainable Artificial Intelligence (XAI) and its role in re-crafting Digital Twins. Note that these three disciplines represent the main ingredients of this thesis. The first part of this chapter (Section 2.1) provides a semi-comprehensive review of Digital Twin technology, including its concept, applications and modeling techniques. The second part of this chapter (Section 2.2) gives an overview of intelligent agents, particularly the Belief-Desire-Intention (BDI) agents. Finally, XAI is briefly described in Section 2.3 before we summarise the weaknesses of the existing literature in Section 2.4. Note that for any work that is strongly relevant to the research questions of this thesis, we discuss them in the related work section of each chapter.

2.1 Digital Twin Technology

Digital Twins have largely become an integral part of the Fourth Industrial Revolution (Industry 4.0). A recent survey conducted by Altair Engineering Inc. has reported that 1393 out of 2000 firms are currently using Digital Twin technology to achieve various business goals [2]. The report includes industries like aerospace, automotive, financial services and healthcare, indicating that Digital Twin technology enjoys popularity among different industrial sectors. Moreover, the significant increase in publications indicates that there has also been serious work in

the academic field of Digital Twins. Figure 2.1 illustrates the rate of publication increase since 2013 [a].

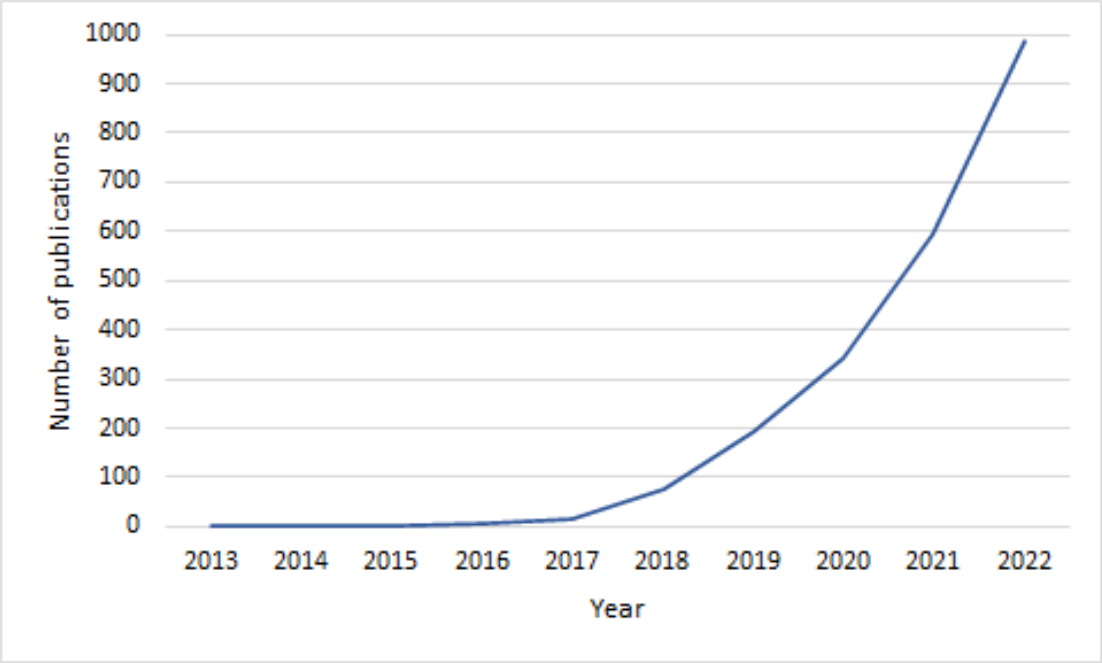


Figure 2.1: Number of publications on Digital Twins

A wide range of initiatives for developing Digital Twins has been proposed in the literature, including traffic control [**17**] and predicting driver intention [21], vehicle manufacturing systems [22], iron and steel product life cycle [23], health care information systems [24], and remote surgeries [25]. Nevertheless, many of these solutions concentrate on specific domain problems and produce unreusable, incompatible, and limited Digital Twin designs, leading to high duplication and confusion levels. It has been reported by HVM Catapult research centre [3] that 80% of its engineers have no common understanding of the Digital Twin. Another recent survey on 2000 firms conducted by Altair Engineering Inc. has reported that 50% of the survey sample has no common understanding of the Digital Twin technology. Another recent research by Schleich et al. [26] has emphasized this.

To address this challenge, we provide a brief review of the state-of-the-art Digital Twins, describing the development of Digital Twins and discussing Digital Twin-related concepts, applications and simulation modelling techniques. We refer to the following reviews for readers interested in an in-depth review of Digital

[a]https://dblp.uni-trier.de/

Twins. Negri et al. [27] reviewed the state-of-art Digital Twins between 2012 and 2016. The review investigated the Digital Twin technology origin, uses, simulation techniques and tools. Tao et al [28] reviewed the Digital Twin concept, history, and theoretical foundations between 2003 and 2018. Based on the reviewed literature, the authors suggested a list of best practices for Digital Twin development.

The remainder of this section is structured as follows. After the introduction, we represent a brief historical view of Digital Twin technology, in Section 2.1.1. Section 2.1.2 presents a comprehensive view of Digital Twin definitions and proposes an unbiased unified one. Section 2.1.3 describes Digital Twin applications. In Section 2.1.4, we describe and group the existing literature according to the most noticeable simulation modeling techniques.

2.1.1 Historical View

Digital Twin technology was first introduced in 2002 [29–31] but did not become an active research space until recently. Arguably, the reason for this recent interest may relate to the emerging identity of the Fourth Industrial Revolution. It is useful at this point to recall the first three industrial revolutions: mechanization, electricity and computer and information technology. The fourth industrial revolution (or simply Industry 4.0) has been characterized as the intertwining of advanced technologies and physical processes. According to the Consortium II [b] (as cited in [32]) Industry 4.0 is “the integration of complex physical machinery and devices with networked sensors and software, used to predict, control and plan for better business and social outcomes".

According to the available literature, Digital Twin technology has been proposed for the first time by Grotepass et al. [33] as “virtual representative, allows independent measuring and the integration of body data into Computer-Aided Design (CAD) data of the production lines and logistic processes at companies.", for creating anthropometric models. NASA (National Aeronautics and Space Administration) defined a Digital Twin as “an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin.

[b] Available from: http://www.iiconsortium.org/docs/IIC_FACT_SHEET.pdf

It is ultra-realistic and may consider one or more important and interdependent vehicle systems" [34].

2.1.2 Digital Twin Concept

Fundamentally, a Digital Twin provides a test bed that mimics the behaviour of its physical counterpart based on real-time data, which can form the basis for optimizations or handle potential deviations before they turn into real-world events. Although this concept of Digital Twin technology sounds straightforward enough, there is no general agreement on the definition of Digital Twins in the literature. Indeed, through our review, we found that there are numerous definitions of the Digital Twin concept. Table 2.1 represents a brief view of Digital Twin definitions that have been proposed in the literature.

It is possible to attribute such diversity in defining Digital Twin technology to two main reasons: (1) physical space nature, Digital Twins are not limited to physical devices and their controllers only. Indeed, a Digital Twin can be a counterpart of any IoT applications, such as smart manufacturing systems, vehicles, supply chains, and many other applications (see Section 2.1.3 for further details) and (2) level of abstraction. Defining Digital Twins at a certain level of abstraction determines the level of detail and complexity by which a Digital Twin is designed and viewed. As mentioned above, the concept of the Digital Twin has been proposed from different abstraction levels and perspectives. This, in turn, impedes the understanding of the concept. Thus, we propose an unbiased definition of a Digital Twin as follows

"*an independent living virtual representation of a physical space that drives innovative business outcomes.*"

An *independent representation* implies that a Digital Twin is not limited to replicating physical space behaviour but also simulating hypothetical situations. A Digital Twin can be a *virtual* counterpart of any *physical space*, including, but not limited to, products, manufacturing systems, factories, or even cities. A *living representation* means that a Digital Twin updates itself constantly using two types of data: operational and environmental data. In this sense, a Digital Twin represents

Table 2.1: Definitions of Digital Twin in literature

No	Ref	Definition of Digital Twin
1	[35]	“a hierarchical system of mathematical models, computational methods and software services, which provides near real-time synchronization between the state of the real-world process or system and its virtual copy."
2	[36]	“a one-to-one virtual replica of a technical asset."
3	[37]	“virtual models of physical artifacts are created in a digital way to simulate their behaviors in real-world environments."
4	[21]	“a corresponding virtual image for a physical asset at the ground. That is, the virtual entity has all the structural as well as behavioural properties as the corresponding physical element."
5	[38]	“an accurate simulation model."
6	[39]	“the use of holistic simulations to virtually mirror a physical system."
7	[40]	“simulate physical world objects by creating as-is virtual images in a cyberspace"
8	[41]	“the virtual representation of a physical object or system across its life-cycle."
9	[42]	“a digital profile of the historical and current behaviour of a physical object or process that helps optimize the business performance."
10	[43]	“ a virtual counterpart of a physical system that can be used for various purposes."
11	[24]	“a virtual and dynamic model in the virtual space that is completely uniform and consistent with its corresponding physical entity in the real space. It can simulate its physical counterpart’s characteristics, behaviour, life, and performance in a timely fashion."
12	[44]	“a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level."
13	[45]	“a virtual replica of physical artifacts."
14	[22]	“a current digital image of the real production plant."
15	[46]	“a complete virtual prototype of an entire system."
16	[47]	“a digital replica of the physical assets of an industrial plant which contains the structure and the dynamics of how the devices and the process operate."
17	[48]	“a cyber representation of a physical twin."

the latest state of the physical space. With the intention of *driving innovative business outcomes*, the Digital Twin manages, analyses and leverages operational and environmental data in order to provide business people and industrial practitioners with critical solutions.

2.1.3 Digital Twin Applications

This section presents the most common Digital Twin applications that we found in the literature. Through our review, we observed three main applications: smart products, smart manufacturing systems and smart factories.

Digital Twins of Products

As stated in [49], a smart product is an entity, such as a software product, service, or tangible object, designed with computational, data storing, communication and interaction capabilities. Schmidt et al. [50] assert that smart products result from intertwining digital and physical workflows across the product life-cycle. The authors suggest a number of smart product capabilities, including, but limited to, the ability to obtain workflow tasks, anticipate upcoming tasks and maintenance operations, perceiving and interacting with the surrounding environment.

An example of smart products related to Digital Twin technology is Virtual Vehicle (VV) models [21], for addressing smart vehicles. The VV model is a virtual state of both drivers and vehicles. It uses, for example, Global Positioning System (GPS) coordinates, current speed, the average speed of vehicles, and driver behaviour to model smart traffic management systems. Another example, Chen et al. [**17**] propose a framework for learning and sharing Digital behavioural Twins of connected cars, for which decision trees and K-Nearest Neighbors are used to learn driver behaviour, while discrete-time Markov Decision Process (MDP) is used to build driving context models and risk analysis. Luo et al. [46] propose a Digital Twin framework to realize self-sensing, prediction and maintenance of Computer Numerical Control Machine Tools (CNCMT).

Digital Twins of Manufacturing Systems

As reported by the National Institute of Standards and Technology (NIST), a smart manufacturing system, or simply smart manufacturing, is “fully integrated, collaborative manufacturing system that responds in real-time to meet changing demands and conditions in the factory, in the supply network and in customer needs".

Biesinger et al. [22] employ IoT, cyber-physical systems and 3D scans of vehicle production systems to create a Digital Twin of two real stations in a body-in-white production system. Shahriar et al. [40] present a cyber-physical cloud-based Digital Twin for physical machine operations, for which cloud environment is used to reduce the gap between Digital Twin and physical machines. Schluse et al. [36] propose Experimentable Digital Twin (EDT) in an act to simulate smart manufacturing systems based on model-based systems engineering and simulation technology. CPS Twinning [39] is a framework to construct Digital Twins of cyber-physical systems. The framework uses specifications identified throughout the system engineering phases. The main components of this framework are: a generator to transfer system specifications into a vertical environment; a vertical environment where the physical space is simulated and the cyber part is reconstructed; and a set of modules that interact with the Digital Twin, such as monitoring, testing, security and safety, and behaviour learning modules. Microservice-based architecture is used in [41] to build a flexible smart cyber-physical production system, in which the Digital Twin is a composition of these microservices. Stojanovic and Milenovic [45] defines a self-aware digital twin as a new generation of Digital Twins which it does not replicate the behaviour of production processes and physical assets only, but also it can reason about its own behaviour.

Digital Twins for Smart Factories

A smart Factory is an industrialization solution for flexible and adaptive production facilities, as described by [51]. Intuitively, it refers to software systems, mechanics, labours, resources, and industrial and non-industrial partners. For our purposes, then, we use the term “Factory" to refer to the shopfloor, which might include smart Manufacturing Systems and products. In this sense, we wish to dis-

tinguish smart factories from Industry 4.0. Utilizing advanced technologies, such as IoT, cyber-physical systems, cloud computing, etc., we call “Industry 4.0", while “smart Factory" we see as the result of employing these technologies on shopfloor entities, such as those mentioned above. Industry 4.0 is a manufacturing trend, while a smart Factory is a result.

To give an idea of smart Factories related to Digital Twins, CyberFactory#1 aims to develop key enabling capabilities for Digital factories and Factories of the Future. CyberFactory#1 includes, but is not limited to, supply chains, human behaviour, finances, goods, and products [52]. Another example is the Digital Twin-Driven smart Shopfloor (DTSF), which considers production processes, tooling, equipment, materials, quality, cost, human, and environmental data [53]. Qi and Tao [37] review the role of IoT, Big data, cloud computing, and machine learning in industry 4.0. The design uses Big Data analytics to gather sharper insights into the product life-cycle, such as customer demands, market preferences, and customer voices about product features and quality. Karakra et al. [24] propose a Digital Twin for hospital services based on discrete event simulation and health care information systems. The work aims to enable the assessment of existing healthcare systems and apply different change impact analyses without interpreting hospital activities. For the developed Digital Twin model to be reliable, hospital information systems and IoT devices are used to collect care and contextual data, respectively.

2.1.4 Two approaches to Design Digital Twins

In this section, we give a brief overview of the most common simulation modeling techniques to design Digital Twins: (1) discrete-event and (2) continuous simulation modeling.

Discrete-event Digital Twins

Banks [54] defines discrete-event simulation modeling as a representation of system components as discrete order of sequential events. Hence, the simulation model transits from one state to another based on the occurrence of an event. Another concept related to discrete-event simulation modeling is agent-based model-

ing and simulation, which enables to recreate and prediction of complex phenomena using autonomous software agents (see section 2.2 for more details). Agent-based simulation and modeling have been proposed in a wide range of applications, such as supply chains and logistics, social sciences, and business processes [55]. Machine learning can be a useful tool for agent-based modeling and simulation software. DeepMind, for example, uses deep reinforcement learning to teach software agents locomotion behaviours without any previous learning; that is, an agent has never shown locomotion behaviours, but it learns how to walk using a reward function [56].

Eckhart and Ekelhart [57] propose a Digital Twin of Industrial control systems (ICSs) based on Finite-State Machine (FSM). Formally, a program P of an ICS is defined by tuple (S, s_0, I, O, γ), where $S = (s_0, s_0, \ldots, s_n)$ is a finite (i.e., non-empty) set of states, $s_0 \in S$ is an initial state, I is a finite set of inputs, O is a finite set of outputs, and γ is a state-transition function: $\gamma : S \times I \rightarrow 2^S$. A Digital Twin $\hat{P}$ is an identical counterpart of the system ($P = \hat{P}$), thus $\gamma(s,i) = \hat{\gamma}(\hat{s},\hat{i}) \Leftrightarrow x_1 = \hat{x_1}$ provided that $(x = \hat{x}) \wedge (i = \hat{i})$. In this sense, replicating a stimulus (i.e., a state) in an input-output fashion requires capturing its trigger. Atorf and Roßmann [58] present a Digital Twin for traffic situation from 2016 in Florida, USA, whose objective is to simulate accurately all possible state trajectories. The Digital Twin relies on monitoring and recording drive state variable $s(i, j, t)$, where i is a property, such as joint angles, positions, and temperatures; j simulation variants; and t is time. On this basis, simulating relies on: state variable s(i, j, t), Time series $\tau(i, j)$, State vector $s(j, t)$, State trajectory $S(j)$. Isochrone map and variant ghost are used to analyse and visualise the physical space. Luo et al. [59] propose a Digital Twin for Computer Numerical Control Machine Tool (CNCMT). The Digital Twin gathers different data on manufacturing phases, such as temperature, pressure, velocity, position, and vibrating of the CNCMT, augmented with system knowledge. An inference engine is used to deduce new information. MWorks software is used to generate descriptive models of the physical space. Neural networks algorithm over the descriptive models to provide fault prediction and diagnosis.

Continuous Digital Twins

Continuous simulation modeling refers to simulation techniques that view the system as variables that change continuously with respect to time. For example, Lamb [60] uses time series plots and flow duration curves to build a continuous simulation of flood frequency estimation. Banks [54] believes continuous simulation is more popular among agricultural, chemical and electrical engineers.

Laaki et al. [25] propose a Digital Twin for remote medical operations. Its goal is to emulate accurately human hand movements during medical operations. To achieve this, the application relies on human hand movements data and a coordination system. The Digital Twin stores human hand movements data in a quaternion format (location, X-, Y-, Z- axis), so that a set of coordinate equations can model the axis-angle representation. Using a control interface and 4G connection, a robotic arm replicates human hand movement in real-life settings. Zhao et al. [61] introduce a Digital Twin for micro-punching systems, i.e., embossing flat sheet materials. In micro-punching physical space, the punching process relies mainly on composite platforms, linear motors, and piezoelectric ceramics that drive (X-, Z- axis), X-axis, and Z-axis, respectively. The Digital Twin obtains these coordinates through reiterating experiments of the punching process under different inputs. Dynamic equations, such as depth value and staggered interval equations, are used to mimic punching functionalities. Nikolakis et al. [62] describe a Digital Twin for a human-based manufacturing environment. Timestamped sensors, Controlled Natural Language (CNL), and multi-depth cameras, are used to gather data on the physical space. The Digital Twin relies on a data-driven motion synthesis approach to simulate human motions and objects picking, placing, and carrying. The authors suggest a Digital-to-analog conversion to provide the physical part with details, optimisation constraints, and potential modifications.

2.2 Intelligent Agents

For readers who are unfamiliar with the concept of “intelligent agent," we briefly present here the notion of an intelligent agent and its main characteristics. Next, we present the BDI architecture with one particular reference for programming

BDI agents. For an elaborate introduction to intelligent agents and the BDI architecture, the reader is referred to [63] and [14], respectively.

Perhaps the most general way the term “intelligent agent" is used is to denote any software entity that has the ability to perceive the environment in which it is situated through sensors and (2) act upon that environment through actuators [64]. This notion of intelligent agents is depicted in Figure 2.2.

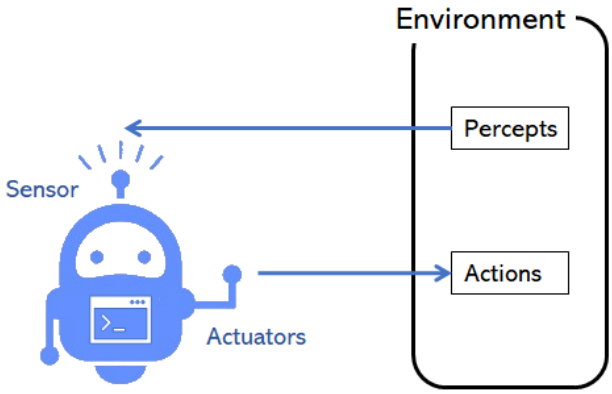


Figure 2.2: Agent-environment interactions

For many researchers, the term “intelligent agent" possess a stronger notion than perceiving and acting. An intelligent agent is a software entity that enjoys the following characteristics [65]:

- **Autonomy**: An autonomous agent is a software entity operating on behalf of an employer but without any direct intervention of that employer or other agents.
- **Reactivity**: A reactive agent has the ability to perceive the environment and, consequently, take an action that serves its goals in a timely fashion.
- **Sociability**: A social agent is a software entity that has the ability to interact with other entities (e.g., humans and agents) through common agent-based communication language.
- **Proactivity**: A proactive agent is one that do not only react to changes in the environment but also responds to changes before they turn into real-world events through the realization of its long-term goals.

- **Rationality**: A rational agent should select actions that are expected to serve the achievement of its goals and optimize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Other researchers tend to describe intelligent agents with an even stronger notion of characteristics, such as learning, flexibility, mobility, and context-sensitive [66, 67]. Another distinctive mark of intelligent agents is their cognitive architectures. A cognitive architecture depicts how the agent implements a certain theory of cognition. The BDI software model is a typical example of cognitive architecture. One might reasonably ask “Why not another software model?". Actually, the reader can find many alternative architectures (e.g., Soar [68], and ACT-R [69]) that have been used for similar applications. It should be noted that it is not our intention to argue that BDI agents are the best way to design Digital Twins of organizations. Rather, this thesis has been built on the premise that the BDI agent framework is already widely used, and its state-of-the-art exhibits a large number of extensions, which can be leveraged as bases for a more fine-grained design of Digital Twins.

2.2.1 BDI Agents

The BDI agent architecture was first raised by Georgeff and Rao [15], which is an architecture inspired by the theory of human practical reasoning of Michael Bratman [70] but which is significantly influenced by the need to harmonise between deliberative and reactive planning in goal-oriented autonomous systems. The origin of the BDI model goes back to folk psychology with a little variation. While folk psychology is commonly used to explain and predict human behaviour [71], the BDI model is used to generate behaviour. Perhaps the most eloquent and simplest description of this notion is the intentional stance of Dennett

> ”First, you decide to treat the object whose behaviour is to be predicted as a rational agent; then you figure out what beliefs that agent ought to have, given its place in the world and its purpose. Then you figure out what desires it ought to have, on the same considerations,

> and finally you predict that this rational agent will act to further its goals in the light of its beliefs. A little practical reasoning from the chosen set of beliefs and desires will in most instances yield a decision about what the agent ought to do; that is what you predict the agent will do." [72, p. 17]

Committing to a course of action (i.e., intention) is the centric notion of the BDI model. Once the agent is committed to bringing about a certain goal, it proceeds with a relevant course of action until, but not necessarily, that goal is accomplished. Among the exceptions are when the goal becomes unreachable, irrelevant or another opportunity comes to light. Arguably, the last exception is what grants BDI agents the sense of harmony between reactivity and deliberation.

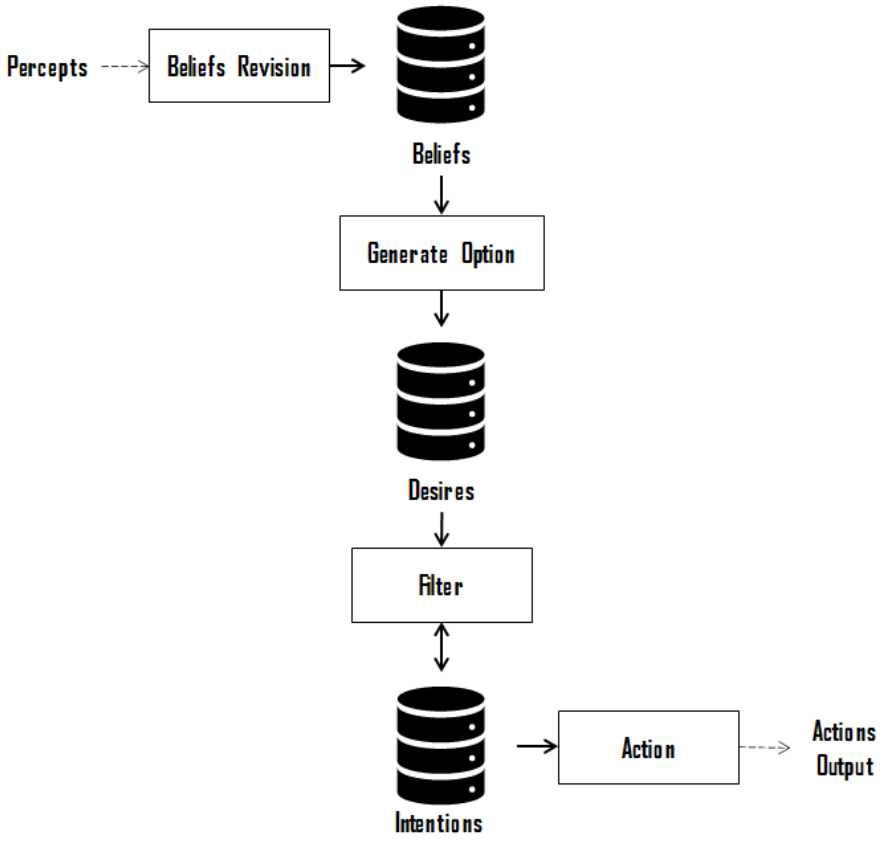


Figure 2.3: A high-level abstraction of BDI architecture

Figure 2.3 illustrates the main components of the BDI agent, which are:

1. The beliefs of the agent, which represent what the agent knows about the environment in which it is situated. New percepts of the world should be

submitted to a belief revision function, which in turn updates the current belief base with the new beliefs. Where sensor readings may not accurately reflect a perfect view of the environment, beliefs can be incomplete and inconsistent

2. The desires of the agent, which are states of affairs (i.e., goals) that the agent would like to bring about. The option generation function looks at the current beliefs and intentions of the agent in order to select which desires to pursue. Normally, the agent is equipped with one or more suitable plans to achieve its goals.

3. The intentions of the agent are plans that the agent is committed to executing in order to achieve its goals. The agent needs to filter unrealistic options (e.g., unreachable goals) and, consequently, determine its intentions. Finally, one intention has to be executed.

Implementing BDI agents requires also a plan library (a set of predefined operational procedures). Despite this discussion, the reader should note that there are slight differences between the original BDI architecture and its implementations. Where there are different agent programming languages (e.g., Jack [73] and 3APL [74]) for BDI agents, we use AgentSpeak(L) programming language [20] as basis to design Digital Twins. This section introduces a brief overview of the standard AgentSpeak(L). For an elaborate introduction to AgentSpeak(L), the reader is referred to [75] and [20]. An AgentSpeak(L) agent is a tuple $\langle E, B, P, I, A, S_E, S_O, S_I \rangle$, where:

- E is a set of events,
- B is a set of beliefs
- P is a set of plans,
- I is a set of intentions,
- A is a set of actions,
- S_E selects an event from the set E

- S_O selects an applicable plan from the set P, and
- S_I selects an intention from the set I

The basic reactive behaviour of an agent system programmed using AgentSpeak(L) involves the system responding to events by selecting an event to address from the set of all pending events (E), selecting a plan from the library (P), and stacking its program into I. A plan in the plan library is a rule of the form $\varepsilon : \nu \leftarrow \rho$, where the program ρ is a predefined strategy designed to handle the goal event ε whenever the context condition ν is believed to be true by the agent. A program ρ can include primitive actions and sub-goals that can be achieved by selecting other suitable sub-plans. A plan can be selected for addressing an event ε if it is relevant and applicable, i.e., designed with respect to the goal event ε, and the agent believes that its context ν is a logical consequence of his belief base, respectively. For our purposes, given an AgentSpeak(L) plan p, we use the functions triggering(p) to return its triggering event, context(p) to return its context conditions, and body(p) to return its program.

An AgentSpeak(L) agent is associated with various selection functions, which form the bases for customizing its internal behaviour (i.e., S_E, S_O, S_I). Selection strategies yield a way of customizing the BDI internal behaviour in terms of event selection (S_E), selection of plans (S_O), and generation and deliberation of goals (S_I). Note that S_E, S_O, S_I are user-defined, i.e., they can be customized by programmers [75].

2.2.2 Positioning BDI Agents Within the Scope

Now, we try to demonstrate how a software agent with a belief-desire-intention architecture can be appropriate to imitate the behaviour of organizations in an environment characterized by various requirements and limitations. Organizations, and a wide range of other physical spaces, exhibit a number of characterises, such as:

1. At any given time, there are different ways in which the business environment can evolve. Note that this could include data on orders, characteristics

of production inputs, also information on the state of the business environment, competitors, regulatory regime, etc. Also, there are different objectives that the organization is required to achieve (e.g., establishing brand awareness and recruiting candidates).

2. As the organization has to act on environment developments, it is required to select suitable activities and procedures that best achieve its objectives. This can include plans for workforce development, product and services, finances and expansion.

3. During the execution of selected options (or even during the selection process), organisational environment may change in an unpredictable and significant manner. This imposes the organization to deliberate - think carefully - about alternative options.

To clarify our claim that a BDI agent can be an adequate means to remodel an organization's behaviour in such settings, let us map the above characterises to the S_E, S_O and S_I, which represent the core of BDI agent reasoning cycle:

1. After perceiving the business environment, an event selection function (S_E) selects an event (e) from the set of all pending events (E), where each event can be assumed to have different importance for the agent. An event represents either change in the organization's goals or changes in the business environment.

2. As the agent has to react to the selected event e, an option selection function (S_O) selects a business plan (p) among other applicable options (O_e). A plan is applicable with respect to an event e if the agent believes that the current business context is a logical consequence of its beliefs.

3. Where the agent might have many intentions (e.g., business goals), an intention selection function (S_I) is, typically, agent-specific. That is, the choice of an intention (i) between other competing intentions (I) might be associated with goals urgency degrees.

A BDI agent is arguably one of the most sophisticated agent architectures on offer, permitting the modelling of organizations using anthropomorphic handles.

2.3 Explainable Artificial Intelligence

For readers who are unfamiliar with eXplainable Artificial Intelligence (XAI), we introduce here a very brief overview of XAI concept. Readers who are unfamiliar with the explainable agency or, indeed, who are familiar with a different notion of XAI (e.g., explainable machine learning) have to proceed to Section 2.3.1. Further details on explainable machine learning can be found in works such as [76] and [77].

Roughly speaking, AI models (whether statistical or cognitive) are black boxes to the user and, indeed, to their designers in the sense of their capability to answer why a certain prediction or decision has been made. Fundamentally, XAI allows humans to understand the predictions or decisions of AI models [76]. One might reasonably ask “Why my AI model needs to be explainable?" especially if it achieves high-performance measurements (e.g., accuracy). The answer is that XAI can help improve human-machine interaction (e.g., teaming and collaboration) by ensuring trustworthiness, transferability, confidence, etc. [76]. As we will show in this thesis, XAI also opens up important new classes of applications, such as competitor modelling in a setting where there is limited information available about competitor behaviour. It should be noted that not all machine learning algorithms are black boxes - some are interpretable (aka white-box) by design (i.e., they do not need an external XAI model), such as decision trees and logistic regression models [77]. However, white-box machine learning algorithms are not always able to achieve high performance.

To illustrate how explainable machine learning works, consider the following example.

Example 2.1. In our scenario, let us assume a startup that aims to understand the decision process of one of its competitors. A particular situation for the startup is how its competitor chooses courier companies to carry out its logistic services. Assume that the selection process depends on four features: years of experience, whether the courier services have been used before, the number of drivers, and whether the courier supports offshore deliveries (columns 2-5). Assume also that these features represent information that many of the industry players know

about. As illustrated in Table 2.2, raw 1-9 are labelled data points representing past decisions (column 6) of the competitor. Whereas, raw 10 is not labelled data point representing a future selection of the competitor.

Table 2.2: Example of an explainable machine learning problem

No.	Experience	Employed?	No. of Couriers	Offshore?	Hired
1	10	Y	35	N	Y
2	9	N	41	N	N
3	15	Y	51	N	Y
4	16	N	60	Y	N
5	9	N	30	N	N
6	17	Y	66	Y	Y
7	21	Y	131	Y	Y
8	13	N	38	Y	N
9	10	N	30	N	N
10	5	N	20	N	?

Assume that we have a machine learning model that was trained to predict whether the tenth local courier company would be hired (Y) or not (N). The learning model can predict this problem as long as the above-cited features are available. Given the above-cited features of the tenth courier, most machine learning algorithms prediction will be (N) with 1.00 accuracy. Although such predictions are beneficial in many applications, the startup still has no clue on which bases its competitor will make such a decision.

Continuing with Example 1, the startup can obtain more explainable predictions by the following two strategies: (1) using a white-box predictive algorithm (e.g., decision trees) and (2) using an external XAI model. The former strategy can be problematic since some learning problems requires sophisticated black-box algorithms (e.g., deep learning algorithms), which are extremely hard to explain. Let us consider, therefore, the latter strategy. Among other XAI techniques, the feature importance score would tell the startup the significance of each feature to the prediction. In this case, it would tell the startup that the competitor concentrates on the feature Employed and overlooks other features.

Perhaps, it is now clearer that XAI was developed as a means of understanding

the predictions and decisions of machine learning algorithms. That is to say, XAI was not designed to build Digital Twins, but it has considerable potential in this application area. At the core of the original proposal of Digital Twin technology was the idea that a Digital Twin should be re-crafted in response to every change in the target system or the operating context [16]. Once Digital Twin developers understand the target system and its operational context (provided by XAI), they do not need to start from scratch.

2.3.1 Explainable Agency

Explainable Agency refers to intelligent agents' ability to explain their actions, and goals [78]. Although the explainable agency can intersect with explainable machine learning (recall that agents can be learning software entities [66]), we shall limit our discussion to goal-driven XAI in this section.

It is generally acceptable to assume that people tend to explain observed behaviour by resorting to their own anthropomorphic vocabularies (e.g., beliefs, emotions, goals and norms). Where most of the well-known agent architectures are inspired by folk psychology, explainable agents could be the most appropriate XAI model to provide structured explanations to humans. Researchers who have used folk psychology-inspired architectures for this purpose have identified three tasks for the agent to be explainable: (i) generating why an action was taken or a goal was carried out, (ii) representing the explanation to the end-user, and (iii) evaluating explanation usefulness. This thesis focuses on the former two tasks, which use the BDI ontology as the implementation vehicle of the first task and AgentSpeak(L) programming language as a means of communication for the second task. Note that existing agent models are not explainable by design [78]. That is to say, similar to machine learning algorithms, an agent model requires an external XAI model to satisfy the above three tasks. However, unlike machine learning algorithms, agent models are open software architecture that allows adding explanation generation modules [78].

Most of the existing work on explainable BDI agents (e.g., [79] and [80]) views action selection in the BDI framework as a result of the agent's current beliefs and

goals[c], which can be represented using a Goal Hierarchy Tree (GHT). A GHT is a tree structure representing a high-level abstraction of agent reasoning. An agent's top-level goal is placed at the root of the GHT. A link from a goal to one or more sub-goals (square nodes) means that these sub-goals must be carried out as part of achieving that top-level goal. Tree leaves (shaded square nodes) represent actions that the agent can execute. For the agent to execute an action, certain beliefs (rounded nodes) placed directly above the action must be true. For illustration purposes, we consider a part of the leading firefighter agent [79] as an example.

Example 2.2. As shown in Figure 2.4, the leading FireFighter Agent (or simply FFAgent) has the goal of leading the firefighter team. FFAgent prepares to handle a fire incident whenever it acquires the belief fire alarm. If the equipment has not yet been collected, then the FFAgent collects the equipment, gets into the fire engine and calls the operator.

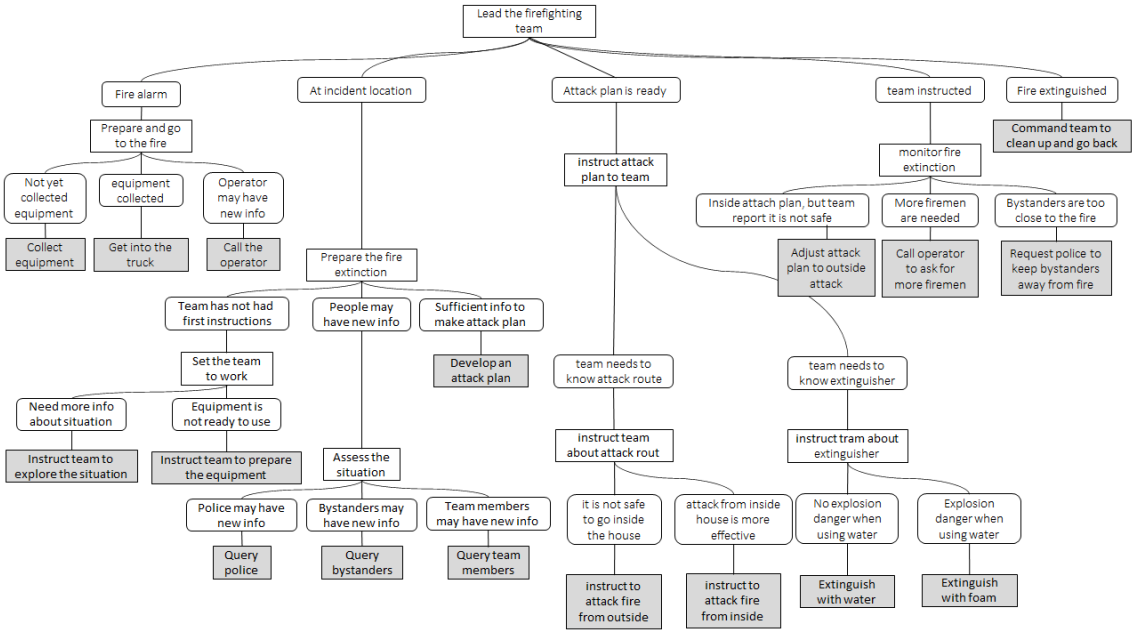


Figure 2.4: An excerpt of GHT of FFAgent

Figure 2.4 illustrates the structure of the goal hierarchy tree to which the goals and beliefs of the FFAgent are updated. One may think of triggering action as the result of the agent's current goals ad beliefs. At every reasoning cycle, if there

[c]We argue that this notion of action selection is exclusive. Indeed, a practical BDI agent triggers an action based on all its handles (e.g., current beliefs, goal, plan and the event, option and intention selection functions). However, explaining action selection by beliefs and/or goals for human-agent interaction purposes can be useful.

is no applicable action, the agent waits until its beliefs change. If multiple actions are applicable, the agent chooses an action based on a particular selection strategy (see section 3.6 for more details). What existing explanation algorithms do is selecting the beliefs and goals that are directly above the selected action in the tree to design explanation patterns. For example, in the goal hierarchy depicted in Figure 2.4, the action Collect equipment can be explained by the belief Not yet collected equipment and the goal Prepare and go to the fire.

Designing agent explanations is not as straightforward as the account above suggests. There are preferences involved in terms of choosing between different types of explanations (e.g., belief-based, goal-based, and belief and goal-based explanations). Determining how to represent generated explanations is related to the task (ii). For example, Harbers et al. [79] described four algorithms to design explainable BDI agents: one using parent goals, one using top-level goals, one using enabling beliefs, and one using the following action or goal in the execution sequence. They found goal-based explanations were slightly preferable to belief-based expansions to explain procedural actions (i.e., a sequence of actions and sub-goals) based on users' evaluation. Nevertheless, belief-based explanations were preferable in explaining conditional and single actions. Similar explanation algorithms were proposed by Kaptein et al. [80], but to investigate the difference in preference of adults and children for goal-based and belief-based explanations. They found that both adults and children prefer goal-based explanations.

2.4 Summarising the Weaknesses

After introducing the main ingredients of this thesis, we summarize the weaknesses of using the current endeavours for developing Digital Twins of organizations and explainable agencies attempting to build on the strengths of the BDI ontology and XAI. We identified six critical shortcomings, namely.

1. While considerable attention has been paid to developing Digital Twins for physical devices and systems (as described in section 2.1.3), the question of developing twins for organizations has received relatively little attention. Digital Twins of organizations can provide value to the organization and

other industry players.

2. Much of the reviewed literature on Digital Twins of anthropomorphic counterparts focuses only on how physical space behaviour can be remodeled (in silico) to output the same external behaviour (as described in section 2.1.4). Recall that real-life organizations are anthropomorphic entities [8–10]. Therefore, it is also of great importance to concentrate on their inner workings. For example, how an organization prioritizes its goals?

3. Although there is a large body of work on explainable agency concentrated on making the internal state of human-like agents and robots more understandable to humans (as described in section 2.3), it does not provide an approach to generate an explicit representation of the agent model. Rather, it generates explanations that look like “because I believe that {beliefs}" and “because I want to {goals}".

4. Much of the literature on explainable BDI agents does not provide a complete account of the reasoning capacity of the BDI model (i.e., beliefs, desires, intentions and plans). To illustrate this, assume a reward function defined to give a positive reward to a BDI agent each time it selects a suitable action. One might reasonably ask “What exactly should I reward?" the S_E, S_O or the S_I. Predicated on the original proposal of the BDI model, all these components shape the action selection mechanism.

5. A significant assumption made by the reviewed literature was the collaborative settings. Collaborative settings allow developers to design Digital Twins, assuming that honest physical counterparts would tell their characteristics at any given time. Clearly, this does not work in many settings, where no more than common patterns, external events, and the external behaviour of the target system are available.

6. Much of the work done on explanation generation in the field of goal-driven XAI has traditionally assumed the availability of explanation generation modules, reliable observations, and deterministic execution of plans. However, in real-life settings, explanation generation modules are not readily avail-

able, unreliable observations are frequently encountered, and plans are non-deterministic.

Chapter 3

Abductive Design of BDI Based Digital Twins

Digital Twin technology ideally manifests the same behaviour (in silico) as its physical counterpart. While considerable attention has been paid to the development of Digital Twins for physical devices/systems, the question of developing Twins for organizations has received relatively little attention. The setting in which we address this problem is very general and, consequently, very challenging. We look at the automatic acquisition of Digital Twins of organizations. To that end, this work builds on the following two premises: (1) that Digital Twins of organizations can provide value, and (2) that Belief-Desire-Intention (BDI) agents are a particularly effective means for representing Digital Twins. The overall approach is to leverage the externally observable behaviour of the target system and then generate candidate BDI agent programs that best explain (in the sense of formal abduction) the observed behaviour. The candidate agent programs are generated by searching through potentially large hypotheses spaces for possible plans, selection functions and beliefs. The resulting approach suggests that using abduction to generate Digital Twins of organizations in the form of BDI agents can be effective.

3.1 Introduction

Fundamentally, a Digital Twin provides a test-bed that mimics the behaviour of its physical counterpart based on real-time data, which can form the basis for op-

timizations or handle potential deviations before they turn into real-world events. A wide range of initiatives for developing Digital Twins have been proposed in the literature, including, but not limited to, vehicle manufacturing systems, product life cycle and health care information systems [81]. Recently, work has emerged addressing the problem of developing Digital Twins of organizations [13].

Our objective in this chapter is twofold. First, we look at the automatic acquisition of Digital Twins. "Hand-crafting" Digital Twins, which is the current norm, comes with several obvious limitations. The time and effort involved in crafting a Digital Twin lead to bottlenecks. Digital Twins have to be "re-crafted" in response to every change in the target system or the operating context [16]. Second, while much of the literature on Digital Twins addresses target systems that are physical devices and their controllers, we propose an approach that can generate effective Digital Twins of organizations without loss of generality to physical and cyber-physical systems. This opens up important new classes of applications, such as competitor modelling, where our approach to Digital Twin acquisition enables the building of effective twins even in a setting where there is limited information available about competitor behaviour.

This chapter presents an approach to automate the acquisition of autonomous, context-aware and adaptive Digital Twins [16] by defining techniques for acquiring agent programs in the BDI architecture. The starting point to our approach is to leverage the externally observable behaviour of the target organization and then generate candidate agent programs that best explain (in the sense of formal abduction) the observed behaviour. The candidate agent programs are generated by searching through potentially large spaces for possible plans, selection functions and beliefs.

We structure this chapter as follows. Section 3.2 presents our running example with pointers to different common patterns of organization behaviour. Section 3.3 introduces some required preliminaries, i.e., AgentSpeak(L) programming language. Section 3.4 describes the application of abductive plan recognition to explain the behaviour of organizations. We describe our approach for belief-based action explanation in Section 3.5. Section 3.6 describes our approach to explaining the behaviour of organizations in terms of goal generation and deliberation

and applicable plan selection strategies. Related work is discussed in Section 3.7 before we conclude and outline future work in Section 3.8.

3.2 Running Example

Startup competitor analysis can be a practical motivating application for using the abductive design of BDI based Digital Twins. As a running example, we consider a startup that aims to build a Digital Twin of one of its competitors, with a view to predicting how the competitor will behave in certain situations. We shall refer to the latter company as the Adversary Co. or simply ACo. A particularly stressful situation for the startup is how ACo takes its products through their life cycle (these situations might present competitive opportunities for the startup in our story). As part of its competitor analysis, the startup also leverages a forecasting model similar to the one taken by Jennings et al. [82]. A key advantage of the forecasting model is that it enables the startup to estimate the stage of products based on different factors such as demand states, product age, and changes in technology.

Example 3.1. In our scenario, ACo designs industrial IoT products to stay on the market for an average lifetime of 4 years. For ACo to manage products during their lifetime, it adopts a similar product life cycle to the one developed by Levitt et al. [83], in that the product typically goes through five recognizable stages: release, active, mature, limited, and obsolete. We identify five common patterns that can arise during the product life cycle.

1. A new product is released to the market before there is a real demand for it and often before a proven competition. Before launching the product, the organization should conduct pre-release testing (i.e., beta test). Also, it is committed to developing and maintaining feedback channels during this stage.

2. Due to sales volume and competition growth, the organization moves the product into the active stage. During this stage, the product is in volume

production and sold according to the price-lists for the active stage. Also, the organization will continue to improve and add new features to the product.

3. As market saturation and sales volume peaks are reached, the organization moves the product into the mature stage. During this stage, the organization offers limited technical support, reduces production and distribution, and does limited product improvement.

4. Due to components unavailability or sales volume declines, the organization moves the product into the limited stage. Last Time Buy (LTB) will be scheduled during this stage, and the organization will make no further product improvements or testing. As a rule, mature stage prices are no longer valid, i.e., higher unit prices may be applied.

5. Due to latent demand or product ageing, the organization moves the product into the obsolete stage. During this stage, the organization will not accept orders for the product, and no technical support will be offered. Nevertheless, the organization is willing to provide a limited amount of compatible spare parts for the product.

As in Example 3.1, each transition in the product stages can be seen as a result of changes in the business environment (i.e., external events), which triggers the execution of a product life cycle strategy. Note that the product does not need to go through these five stages. For example, a product can be classified as limited soon after being active because of component unavailability, e.g., semiconductor shortage.

Example 3.2. After identifying ACo as a direct competitor, let us assume that the startup decided to concentrate its competitive analysis on five products: (1) IoT Device Gateway model sgx51, (2) Wireless IoT Gateway model wgt67, (3) Time GPS Tracker model tpt09, (4) IoT Smart Home Kit model swd01, and (5) Temperature & Humidity Sensor model ths04. We describe the history of the past behaviour of ACo for managing these five products below. While Table 3.1 illustrates the behaviour log for managing these five products

Table 3.1: An example of ACo external behaviour

Timestamp	Action	Timestamp	Action
t_3	advertise(sgx51,socialMedia)	t_{29}	rise_price(wgt67)
t_5	launch(sgx51)	t_{33}	advertise(swd01,email_list)
t_8	create_feedback(sgx51)	t_{35}	reduce_price(swd01)
t_{14}	advertise(wgt67,email_list)	t_{37}	increase_production(swd01)
t_{16}	advertise(tpt09,email_list)	t_{38}	add_features(swd01)
t_{18}	LTB(wgt67)	t_{40}	improve(swd01)
t_{20}	reduce_support(tpt09)	t_{44}	advertise(ths04,email_list)
t_{22}	discontinue_updating(wgt67)	t_{46}	stop_distributing(ths04)
t_{24}	reduce_production(tpt09)	t_{48}	cease_production(ths04)
t_{25}	discontinue_testing(wgt67)	t_{50}	discontinue_support(ths04)
t_{27}	reduce_updating(tpt09)	t_{52}	provide_parts(ths04)

Table 3.2 represents an event log that stores information on the state of the business environment during the execution of an excerpt of the behaviour log in Table 3.1 (we omit details on changes in the business environment during the rest of the behaviour log, which would be not necessary for the objectives of our running example). We use an underlying commercial vocabulary (i.e., state description language) to describe the changes in the business environment. For example, at time t_{30}, the first observation indicates that sensor model ths04 has been in the market for 4 years.

Table 3.2: An excerpt example of changes in ACo business environment

Timestamp	Event
t_{30}	age(ths04) = 4Y0M0D
t_{31}	high_demand(swd01)
t_{32}	competitors(swd01)
t_{34}	promoted(swd01,active)
t_{36}	price_reduced(swd01)
t_{39}	features_added(swd01)

As the product goes through its life cycle stages, a wide variety of information on the product characteristics and the state of the business environment can be represented through events. We are interested in recording two types of events: (1) events that record the behaviour of the organization (e.g., tasks and activities recorded in Table 3.1) and (2) events that record state changes in the business environment (e.g., changes in the state of the business environment recorded in Table 3.2). Obtaining these two types of events can be achieved by instrumenting the business environment with off-the-shelf market monitors. However, we do

not claim to be able to obtain a complete description of the state of the business environment in competitive settings.

Now, we can relate the problem of this work to our running example. Fundamentally, a Digital Twin should re-create the externally visible behaviour of ACo, as captured in its behaviour logs (e.g., Table 3.1). Recreating the behaviour depicted in Example 3.2 is far from trivial and involves a number of strategies for goal deliberation and plan selection. For example, the startup might ask: How does ACo react to competing product management strategies? Why did ACo select a particular plan among other options? What must have been known for ACo to perform a particular task over another? A more fine-grained definition of a Digital Twin, therefore, is one that not only faithfully recreates the externally visible behaviour of ACo but also faithfully re-creates its internal workings. We address the problem of designing such Digital Twins based on the BDI agent ontology (specifically AgentSpeak(L) programs [20]) by the following means (1) abductive plan recognition, (2) explaining by beliefs, and (3) explaining by selection strategies.

3.3 Programming BDI Agents

A software agent system programmed using BDI programming languages commonly consists of a belief base (what the agent knows about the environment in which it is situated), a set of events (desires that the agent would like to bring about), a plan library (a set of predefined operational procedures of the domain), and a base of intentions (plans that the agent is committed to executing). This chapter builds on the well-studied AgentSpeak(L) language for programming BDI agents as a demonstration platform.

Now we apply the alphabet of AgentSpeak(L) language introduced in this section to Example 3.1. Figure 3.1 illustrates an excerpt of the product life cycle management plan library written on the basis of AgentSpeak(L) programming language.

```
/* initial beliefs and rules */
lifetime(x) = .date(04,00,00).
release(x):- new(x) & competitive(x).
active(x):- high_demand(x) & not competitive(x).
mature(x):- market_saturation(x) & overfull_demand(x).
limited(x):- declining_demand(x) | not available_component(x).
obsolete(x):- age(x) > lifetime(x) | latent_demand(x).

/* plans */

@p1  +release(x): beta_test(x)
                <- !promote(x);
                launch(x);
                create_feedback(x).
@p2  +active(x): not stocked(x)
                <- !promote(x);
                reduce_price(x);
                increase_production(x);
                add_features(x);
                improve(x).
@P3  +mature(x): stocked(x)
                <- !promote(x);
                reduce_support(x);
                reduce_production(x);
                reduce_updating(x).
@p4  +limited(x): stocked(x)
                <- !promote(x);
                LTB(x);
                discontinue_updating(x);
                discontinue_testing(x);
                rise_price(x).
@p5  +obsolete(x): not stocked(x)
                <- !promote(x);
                stop_distributing(x);
                cease_production(x);
                discontinue_support(x);
                provide_parts(x).
@p6 +defective(x): distributed(x) & not obsolete(x)
                <- stop_distributing(x);
                !promote(x);
                withdraw(x);
                modify_production(x).
@p7  +!promote(x): not promoted(x) <- advertise(x,email_list).                .
@p8  +!promote(x): not promoted(x) <- advertise(x,socialMedia).
```

Figure 3.1: An excerpt of product life cycle management plan library

The main task of the plan library is to manage a product as it goes through its life cycle. As illustrated in Figure 3.1, the belief base contains an initial belief regarding the lifetime of the product and rules that allow us to conclude the stage of the product based on its characteristics, information on the state of the business environment, competitors, etc. Belief addition events trigger each depicted top-level plan due to the perception of the business environment, which in our scenario are outputs of the product life cycle forecasting model. Note that these plans have a common sub-goal !promote(x), for which there are two applicable options to achieve it (p7 and p8) whenever the organization does not believe that the product has been promoted yet.

3.4 Abductive Plan Recognition

We describe in this section the application of abductive plan recognition as a hypothesis assembly problem[a] to support Digital Twin developers in explaining the external behaviour of organizations. However, what our mechanism tries to infer is slightly different from the notion of classical abductive plan recognition, where it seeks to find a set of plans whose execution generates precisely the observed action sequence. To our application area, abductive plan recognition is designed to take

1. A set of potential ordered plans, denoted by H, which describes common patterns of organization behaviour, and

2. A behaviour log, denoted by D, of the target organization that we want to explain.

Assuming:

1. That the observed behaviour is rational (i.e., there are no selection problems).

2. That all behaviour of interest we might extract from the available data will be drawn from H.

[a] logical abduction is also of interest, and we think that it is an applicable extension of the settings presented in this chapter, but is beyond the scope of this chapter

Explaining the external behaviour of an organization using H computes the following:

1. An initial high-level event, and
2. A composite hypothesis of plans and sub-plans ($\mathsf{H}' \subseteq \mathsf{H}$) from the plan library

such that the execution of the plans and sub-plans thus identified generates precisely the observed action sequence given the initial high-level event thus identified (we use the terms “hypothesis" and “plan " interchangeably in this chapter). Particularly, we seek to find plans and sub-plans that explain the observed behaviour (i.e., organization external behaviour). We say that H' explains a given organization external behaviour if its execution generates precisely that behaviour. Finally, to ensure the practical use of the inferred model, we need to assume the completeness of H'.

Example 3.3. To illustrate this idea, consider the behaviour log given in Table 3.1, and let H be given in our running example specifications in Figure 3.1. For this simple example, it might be straightforward to infer that the top-level plans p1,p2,p3,p4, and p5 explain the behaviour represented in Table 3.1.

As discussed above, organizations are driven by a wide range of values that shape their behaviour, and they cannot be recognized using domain knowledge. Recall that one of the main objectives of this work is to recreate any organization behaviour with as little as possible previous knowledge of the underlying behaviour of the organization. Intending to address this issue, we define the following algorithm.

Algorithm 3.1 (Abductive plan recognition algorithm). Let D be an observed sequence of tasks and activities and H be a set of potential explanations written on the basis of AgentSpeak(L). A composite hypothesis $\mathsf{H}' \subseteq \mathsf{H}$ that explains D can be computed as follows

Algorithm 3.1 starts with parsing the triggering part of hypotheses that exist in H. What the function $\mathsf{top}(\mathsf{E})$ does is simply to select the first event (e) in E at

Algorithm 3.1. Abductive Plan Recognition Algorithm

```
1:  D = {d1,...,dn}
2:  H = {h1,...,hm}
3:  E = ∅
4:  H' = ∅
5:  For h in H: E = E ∪ [triggering(h)]
6:  stack = ∅
7:  while(E ≠ ∅):
8:    e = top(E)
9:    Re = unify_event(H)
10:   stack.push(Re)
11:   E = E - {e}
12:   For ht in stack:
13:     For am in body(ht) and dn in D:
14:         if(am = dn):
15:             explain(ht) = [dn]
16:             H' = H' ∪ {ht}
17:         if(parent(ht) ≠ nil):
18:             explain(parent(ht)) = [dn]
19:         if(am = !g(t)):
20:             e = !g(t)
21:             Re = unify_event(H)
22:             Foreach h in Re: parent(h) = ht
23:             stack.push(Re)
24:     Endfor
25:   Endfor
26:   stack.clear()
27: Endwhile
```

each parse step (line 8). Afterwards, it uses the function unify_event to retrieve all relevant hypotheses R_e with respect to the event e (line 9) and then removes it from E (line 11). A nested parsing runs over R_e (lines 12 and 13). At each parse step, the body part of each relevant hypothesis is compared with D elements (line 14). We use the function parent(h) to return the plan for which the hypothesis h has been triggered. Depending on the current hypothesis body construct, the algorithm determines if the potential explanation should be added to H' (line 16) or parsing hypothesis sub-plans (line 19). The algorithm terminates when $E = \varnothing$ (line 7).

It is useful at this point to disclose some insightful strength points and weaknesses we observed through the implementation of Algorithm 3.1. Our algorithm offers valuable advantage for: (1) unlike other workflow mining algorithms (e.g., alpha algorithm [84]), it works efficiently on both sorted and unsorted observa-

tion logs, (2) as we describe in chapter 6, in the worst-case scenario, where the target hypothesis is at the end of H or not present at all, Algorithm 3.1 will have to scan the entire dataset. This results in an average and worst-case time complexity of O(n), where n is the number of hypotheses in H, and (3) even when all the relevant observations are not available (due to competitive settings) it can still provide plausible explanations. However, Algorithm 3.1 can be (1) subjectivity, where the selection of the best explanation relies on the programmer's background knowledge, (2) it does not guarantee that the explanation is true or correct, and (3) it may lead to premature conclusions if it overlooks other plausible explanations that were not initially considered.

Finally, we should disclose a number of relaxing assumptions that were made to focus on the core issue of this work. First, although we aim to automate the acquisition of agent-based Digital Twins, we need manual input to map between the vocabulary used in the plan library and the vocabulary used in the event logs used. Second, in other settings, target plans are not necessary from the set of potential plans. Hence, other notions of plan recognition would be more suitable, e.g., discovering plans by executing action models to best explain the observed actions, such as in [85]. Third, it is important to note that we do not claim to be capable of inferring a complete and correct composite of hypotheses but providing a description of organization behaviour. Developers can then use this description for developing a more complete and correct Digital Twin.

3.5 Explaining by Beliefs

We seek in this section to provide explanations in terms of the beliefs responsible for organization actions, such that explanations about an action would possibly describe the knowledge or the context of the organization about its business environment. Explaining by beliefs can be summarized as follows: Given as inputs

1. A behaviour log of the target organization,

2. Information that other industry players would expect the target organization to be aware of, and

3. The set of plans whose execution generated input (1),

compute the beliefs that must have been known to the organization in order to execute every action referred to in the behaviour log. Let us consider the following example to highlight this problem.

Example 3.4. Continuing with Example 3.2, let us assume that the startup observed the events shown in Table 3.3 (a) while moving IoT Smart Home Kit model swd01 into the active stage, as shown in Table 3.3 (b).

Table 3.3: An example of ACo external behaviour (excerpt 1)

(a)

Timestamp	Event
t_{30}	age(ths04) = 4Y0M0D
t_{31}	high_demand(swd01)
t_{32}	competitors(swd01)
t_{34}	promoted(swd01,active)
t_{36}	price_reduced(swd01)
t_{39}	features_added(swd01)

(b)

Timestamp	Action
t_{33}	advertise(swd01,email_list)
t_{35}	reduce_price(swd01)
t_{37}	increase_production(swd01)
t_{38}	add_features(swd01)
t_{40}	improve(swd01)
t_{44}	advertise(ths04,email_list)

We expect that explaining by beliefs can help Digital Twin developers to answer, for example, the following question: What must have been known for ACo to reduce the price of the product swd01?

3.5.1 Explanation Design

Recall that the task of explaining by beliefs is to generate the beliefs because of which a certain action was executed. To that end, we build on the approach of [79] to design explainable BDI agents. A BDI agent triggers an action with respect to its goals and beliefs, which can be represented in terms of a Goal Hierarchy Tree (GHT), such as in [79], [80] and [86]. In the following, we describe the structure of GHTs, which are based on hierarchical task analysis, an approach utilized in cognitive psychology for complex human tasks specification. We then use GHTs later in this section, where we describe how we can design explanations by beliefs.

Definition 3.1 (Goal Hierarchy Tree). A Goal Hierarchy Tree (GHT) is a tree structure representing a high-level abstraction of an agent's reasoning. In any GHT, the agent's main goal is placed at the root of the tree. A link from the top-

level goal to one or more sub-goals (square nodes) means that these sub-goals must be achieved as part of the top-level goal. Tree leaves (shaded square nodes) represent actions that the agent can execute. For the agent to execute an action, certain beliefs (rounded nodes) placed directly above the action must be true.

Example 3.5. Continuing with Example 3.4, consider the GHT illustrated in Figure 3.2. The top-level node represents the main goal of ACo (i.e., moving product swd01 to the active stage) at time t_{33}. A link from the belief node to its children nodes means that ACo must have known high_demand(swd01) and competitors(swd01) when it performed, for example, the action increae_production(swd01).

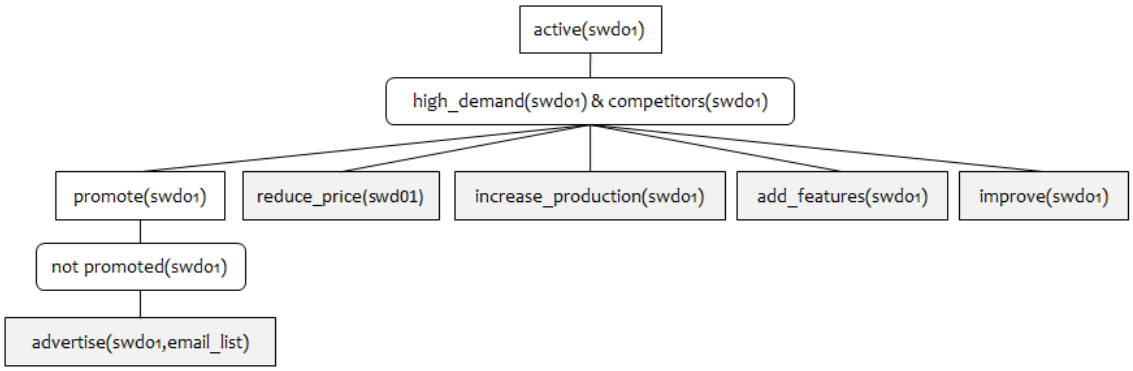


Figure 3.2: A GHT of ACo

Figure 3.2 illustrates the structure of the GHT to which the goals and beliefs of the organization are updated. One may think of triggering action as the result of the organization's current goals and beliefs. If there is no applicable action, the organization waits until its beliefs change. If multiple actions are applicable, then the organization chooses an action based on a particular selection strategy (which will be discussed in detail in Section 3.6).

When a developer asks to explain an action in terms of beliefs, a GHT is constructed, to which the goals and beliefs of the organization are updated. Next, an explanation generation returns the beliefs that are directly above the selected action to design an explanation by beliefs. Continuing with Example 3.5, explaining by beliefs generates an explanation that looks like "given that the ACo performed the action reduce_price(swd01), it must have known high_demand(swd01) and competitors(swd01)".

We should point out a number of relaxing assumptions that were made to focus

on viewing the target system through the lens of BDI agents. We assumed that GHTs can be readily available to the observer, deterministic execution of plans and reliable observations. However, these are, to a certain extend, hard assumptions in real-life settings. To that end, we propose a data-driven approach to mining and validating belief-based explanations in chapter 5.

3.6 Explaining by Selection Strategies

Our guiding intuition behind explaining by selection strategies is that practical strategies to S_E, S_O, and S_I used to customize the rational behaviour of BDI agent systems can also explain the rational behaviour of organizations. We concentrate in this chapter on explaining by S_E, S_O, and S_I strategies (other selection functions are also of interest, and we believe that an extension of the mechanism presented here can address these but are outside the scope of the present chapter). To that end, we leverage existing strategies to AgentSpeak(L) selection functions used in practical implementations for programming BDI agents, including First-in-First-out (FIFO), Round Robin (RR), and User-Defined (UD) selection strategies. Table 3.4 outlines the potential explanations for explaining by selection strategies, which are described later in this section in greater detail.

Table 3.4: Main selection strategies for customizing the BDI reasoning cycle

Strategy	Explanation	Description
A1	FIFO S_E	Selects the first event in the list of pending events.
A2	FIFO S_O	Selects the first applicable plan from the set of applicable plans.
A3	UD S_O	Selects one applicable plan based on user-defined conditions.
A4	FIFO S_I	Executes each intention to completion before starting another one.
A5	RR S_I	Executes a fixed number of steps of each intention in turn.

Explaining by selection strategies involves some mechanisms that take as inputs (1) A behaviour log of the target organization, (2) an event log, (3) plans and sub-plans whose execution generated these logs, (4) selection strategies that serve as potential explanations, and find the best selection strategies for explaining the observed behaviour. Recall that input (3) can be abductively recognized

given (1) and a set of potential plans, as discussed in Section 3.4. We view explaining by selection strategies as an inclusive process, which accounts for S_E, S_O, and S_I selection strategies without distinction. Initially, users might prefer to explain an observed behaviour using some BDI ontology. For example, [79] has shown that explaining by enabling beliefs and parent goals are preferable to users. Nevertheless, users may overlook some of the desired explanations that may present competitive opportunities in our settings. Thus, explaining by all AgentSpeak(L) selection functions is essential. To illustrate this with our running example, we consider three cases[b] in which selection strategies can explain the observed behaviour.

Case I: FIFO S_E, FIFO S_O, and FIFO S_I

Case I is the most straightforward rational behaviour that selection strategies can explain, where it can be explained merely by FIFO S_E, FIFO S_O, and FIFO S_I. We expect FIFO S_E to be helpful when the revision of beliefs is unobservable (due to the competitive settings of this work). FIFO S_O can deliver useful explanations when the set of applicable plans is always singleton (i.e., S_O has only one applicable plan). FIFO S_I allows us to explain the non-interleaving behaviour of the organization. To clarify this case, let us consider the following example.

Example 3.6. Due to the risk of product information overload and lack of consistency, organizations often tend to minimize the interactions between product management or marketing plans. To illustrate this, consider the behaviour of moving product swd01 to the active stage and product ths04 to the obsolete stage, as depicted in Table 3.5 and let the event log be given in our running example.

Table 3.5: An example of ACo external behaviour (excerpt 2)

Timestamp	Action
t_{33}	advertise(swd01,email_list)
t_{35}	reduce_price(swd01)
t_{37}	increase_production(swd01)
t_{38}	add_features(swd01)
t_{40}	improve(swd01)

Timestamp	Action
t_{44}	advertise(ths04,email_list)
t_{46}	stop_distributing(ths04)
t_{48}	cease_production(ths04)
t_{50}	discontinue_support(ths04)
t_{52}	provide_parts(ths04)

[b]Many other cases are possible also, however they are redundant.

A1: Explaining by FIFO S_E. FIFO S_E selects which event to be handled based on event temporal ordering (selects the first event in the list of pending events). Note that it can be irrational to explain event selection based on behaviour logs due to the independency of S_O and S_I selection strategies. For instance, in Example 3.6, ACo selected to move the product swd01 into the active stage, although perceiving that sensor model ths04 has been obsolete due to product age occurring at an earlier stage.

A2: Explaining by FIFO S_O. FIFO S_O selects an applicable plan based on the order in which the set of applicable plans appears in the plan library (selects the first applicable plan in the set of applicable plans). For instance, in Example 3.6, ACo chose email marketing (plan p7) to promote changes in its products life cycle status (i.e., handling the sub-goal $+!\mathsf{promote}(\mathsf{x})$), according to the order in which the applicable plans (plans p7 and p8) appear in Figure 3.1.

A4: Explaining by FIFO S_I. FIFO S_I executes one intention to completion before selecting another intention. For example, the behaviour shown in Example 3.6 represents two intentions: one to achieve the goal of moving the product swd01 into the active stage and another to move the product ths04 into the obsolete stage. FIFO S_I can explain the behaviour depicted in Example 3.6, since ACo selected to carry out moving $+!\mathsf{active}(\mathsf{swd01})$ to completion before committing to achieve $+!\mathsf{obsolete}(\mathsf{ths04})$.

Case II: FIFO S_E, FIFO S_O, and RR S_I

The interleaving of plan steps mainly characterizes case II. The only difference between Case II and simple rational behaviour is that intention execution is interleaved with others. In contrast, in Case I, the goal is achieved, or the next step in the intention cannot be executed. Hence, the behaviour in Case II is explained by FIFO S_E, FIFO S_O, and RR S_I. We expect RR S_I to explain the interleaving behaviour of plans steps. To illustrate this case, let us consider the following example.

Example 3.7. Assume that managing mature and limited products possess the

same importance for ACo. Where it is possible for mature and limited products to emerge regularly, ACo can commit to managing either before committing to managing the other. Nevertheless, doing so may result in trapping short-term tasks behind long-term tasks, which, in turn, requires the organization to ensure fairness between competing product life cycle strategies. For example, consider the behaviour of moving product wgt67 to the mature stage and product tpt09 to the limited stage, as depicted in Table 3.6 and let the event log be given in our running example. **A1: Explaining by FIFO S_E.** Explains the organization behaviour

Table 3.6: An example of ACo external behaviour (excerpt 3)

Timestamp	Action
t_{14}	advertise(wgt67,email_list)
t_{16}	advertise(tpt09,email_list)
t_{18}	LTB(wgt67)
t_{20}	reduce_support(tpt09)
t_{22}	discontinue_updating(wgt67)

Timestamp	Action
t_{24}	reduce_production(tpt09)
t_{25}	discontinue_testing(wgt67)
t_{27}	reduce_updating(tpt09)
t_{29}	rise_price(wgt67)
t_{33}	advertise(swd01,email_list)

according to the guidance given in explanation A1 (Section 3.6.1).

A2: Explaining by FIFO S_O. Explains the organization behaviour according to the guidance given in explanation A2 (Section 3.6.1).

A5: Explaining by RR S_I. RR S_I executes a certain number of steps of each intention in turn. More precisely, RR S_I allows the agent to be committed to concurrently executing a set of plans while keeping the option of further commitments to other plans open. For instance, in Example 3.7, the two product life cycle management plans p3 and p4 have been executed concurrently (one step in turn). This, in turn, ensures fairness between competing product life cycle strategies.

Case III: FIFO S_E, UD S_O and RR S_I

So far, we have assumed that predefined selection strategies (e.g., FIFO and RR) can explain the rational behaviour of an organization. Nevertheless, in real-life settings, organizations are driven by a wide range of values that shape their selection strategies. For Case III, we consider the problem of what is required to

be done when predefined selection strategies are unable to maintain rational selection. We offer developers an easy yet practical guide to manually override the first-cut selection functions from available data. Our discussion concentrates on settings with plan selections, but the approach easily extends to other AgentSpeak(L) selection functions.

Example 3.8. Promoting media releases can be communicated through a variety of means, such as television advertising, online and printed newspapers, ACo's website, and social media (e.g., Facebook and Twitter). Figure 3.3 represents different plans for notifying customers of the changes in product life cycle status.

```
@p7  +!promote(x): not promoted(x) <- advertise(x,socialMedia).
@p8  +!promote(x): not promoted(x) <- advertise(x,email_list).
```

Figure 3.3: An example on a set of applicable plans

It is clear that the sub-goal +!promote(x) has two applicable options to achieve it (p7 and p8) whenever ACo does not believe that the product has been promoted yet. Now, let us consider the behaviour shown below in table 3.7 and let the event log be given in our running example.

Table 3.7: An example of ACo external behaviour (excerpt 4)

Timestamp	Action	Timestamp	Action
t_3	advertise(sgx51,socialMedia)	t_{20}	reduce_support(tpt09)
t_5	launch(sgx51)	t_{22}	discontinue_updating(wgt67)
t_8	create_feedback(sgx51)	t_{24}	reduce_production(tpt09)
t_{14}	advertise(wgt67,email_list)	t_{25}	discontinue_testing(wgt67)
t_{16}	advertise(tpt09,email_list)	t_{27}	reduce_updating(tpt09)
t_{18}	LTB(wgt67)	t_{29}	rise_price(wgt67)

A1: Explaining by FIFO S_E. Explains the organization behaviour according to the guidance given in explanation A1 (Section 3.6.1).

A3: Explaining by UD S_O. UD S_O selects an applicable plan based on handcrafted conditions. We expect that explaining by UD S_O is useful, in particular

when FIFO S_O fails to explain the observed behaviour rationally. For instance, in Example 3.8, the goal event $+!\mathsf{promote}(x)$ has two relevant plans $\{\mathsf{p7}, \mathsf{p8}\}$, and they can be selected whenever ACo does not believe that the product has been promoted. We say that FIFO S_O is an irrational selection strategy if $S_O = \mathsf{p7}$ at some instances and $S_O = \mathsf{p8}$ at other instances, which has occurred at timestamp t_3 and timestamp t_{14}, respectively. A straightforward way to solve this is to seek more beliefs (as described in Section 3.4) to explain each selection and try again to override the S_O method accordingly. For example, one may override the S_O method to make the selection based on the current stage of the product.

A5: Explaining by RR S_I. Explains the organization behaviour according to the guidance given in explanation A5 (Section 3.6.2).

3.6.1 Abductive Design of BDI Selection Functions

After introducing explaining by selection strategies, we shall now describe how these explanations are made use of. The basic idea behind the abductive design of AgentSpeak(L) selection functions is that those selection strategies that can explain an observed behaviour can also be used as the basis for customizing AgentSpeak(L) selection functions. As such, the execution of the plans and sub-plans thus identified by, for example, our abductive plan recognition mechanism generates precisely the observed behaviour given the selection functions thus customized. We term the customization of AgentSpeak(L) selection functions based on observation as the abductive design of AgentSpeak(L) selection functions.

Algorithm 3.2 (Abductive design of AgentSpeak(L) program). From the results of the abductive plan recognition (i.e., H') and the available selection strategies (i.e., potential explanations), we can leverage the AgentSpeak(L) interpreter of Rao [20] to customize S_E, S_O and S_I abductively as follows.

Algorithm 3.2 presents an abstract interpreter for AgentSpeak(L) given the composite of hypotheses H' and the selection strategies found using the guidelines in this section.

Algorithm 3.2.Abductive design of AgentSpeak(L) program

```
1: H′ = {h1, ..., hn}
2: While E ≠ ∅:
3:   IF explain(A1) = E:
4:     SE(E) = top(E) // implements FIFO SE
5:   E = E - top(E)
6:   Re = unify_event(H′) // retrieves all relevant plans
7:   Oe = check_context(Re) // determines applicable plans
8:   IF explain(A2) = Oe:
9:     SO(Oe) = top(Oe) // implements FIFO SO
10:  IF explain(A3) = Oe:
11:    SO(Oe) = UD(Oe) // implements UD SO
12:  I = Construct_intention_stacks(H′)
13:  For each i in I:
14:    IF explain(A4) = i:
15:      SI(I) = top(I) // implements FIFO SI
16:    IF explain(A5) = i:
17:      SI(I) = RR(I) // implements RR SI
18: Endwhile
```

Finally, we need to point out two key abstractions that were made to concentrate on the core problem of explaining by selection strategies. First, we abstract from theoretical approaches to intention selection, e.g., coverage-based [87] and summary information-based intention selection [88]. Clearly, theoretical approaches to intention selection are relevant for explaining by selection strategy. Nevertheless, for the practical results of this work, we can abstract from this. For similar reasons, we abstract from the theoretical approaches to applicable plan selection, e.g., cost-based [89] and best outcomes based [90] plan selection.

3.7 Related Work

Our work is related to multiple research areas: XAI (used as explainable agency in this chapter), learning agents, and process mining. In this section, we present different approaches in the literature as alternatives to support the design of Digital Twins of organizations. Afterwards, we put our work into context.

3.7.1 Digital Twins of Anthropomorphic Systems

Although a large body of literature exists on developing Digital Twins for physical devices and systems, few works focus on developing Digital Twins for anthropo-

morphic entities (e.g., organizations), where re-crafting the internal behaviour of the physical space is relevant.

Focusing on patient flows, [91] proposed a Digital Twin for hospital services. For the Digital Twin to be reliable, hospital information systems and IoT devices are employed to collect care and contextual data, respectively. These data represent the incorporation of dynamic and static resources and predefined care flows. The hospital Digital Twin employs a discrete event simulation software package (FlexSim) to assess the health care current state and observe its dynamic behaviour. [58] presented a Digital Twin for automated driving based on traffic situations from 2016 in Florida, USA, whose objective is to accurately simulate all possible driving trajectories. The Digital Twin relies on monitoring and recording drive state variables. Isochrone map and variant ghost are used to analyze and visualize the physical space. [62] described a Digital Twin for a human-based manufacturing environment. Timestamped sensors, Controlled Natural Language (CNL), and multi-depth cameras are used to gather data on the physical space. The Digital Twin relies on a data-driven motion synthesis approach to simulate human motions and objects picking, placing, and carrying. The authors suggest a digital-to-analogue conversion to provide the physical part with details, optimization constraints, and potential modifications. [25] proposed a Digital Twin for remote medical operations. Its goal is to emulate accurately human hand movements during medical operations. The application relies on human hand movements data and a coordination system to achieve this. The Digital Twin stores human hand movement data in a quaternion format so that a set of coordinate equations can model the axis-angle representation. Using a control interface and 4G connection, a robotic arm is used to replicate human hand movement in real-life settings. [92] proposed a framework for learning and sharing Digital behavioural Twins of connected cars, for which decision trees and K-Nearest Neighbors are used to learn driver behaviour, while discrete-time Markov Decision Process (MDP) is used to build driving context models and risk analysis.

3.7.2 Explainable Agency

Our work is related to explainable agency, which seeks to make the internal state of intelligent agents and robots more understandable [78] to humans. [93] proposed an abductive approach to derive the mental states of BDI agents in terms of beliefs and goals based on their observed actions. [79] described four algorithms to design explainable BDI agents: one using parent goals, one using top-level goals, one using enabling beliefs, and one using the next action or goal in the execution sequence. They found goal-based explanations were preferable to belief-based expansions to explain procedural actions (i.e., a sequence of actions and sub-goals) based on user evaluation. [80] applied belief-based and goal-based explanation algorithms over GHTs to investigate the difference in preference of adults and children for goal-based and belief-based explanations. They found that goal-based explanations are preferable to both adults and children than belief-based explanations. [93] proposed an abductive approach to infer the mental states of BDI agents in terms of beliefs and goals. They described three explanatory strategies under three perceptive presumptions: complete, late, and partial observations. Sindlar et al. extended this work to an explanation approach considering three organizational principles: roles, norms, and scenes in [94]. Related, but in a different domain, is the work presented by [95] also leveraged the reasoning capacity of the BDI model to deduce students' emotions from the observable behaviour of students.

3.7.3 Learning Agents

Our work is also related to learning agents, particularly learning BDI agents. [96] applied inductive logical decision trees to enable BDI agents to learn plan executability (i.e., success and failure of plan execution). [97] proposed two approaches: aggressive concurrent and bottom-up learning to learn the success probability of plans based on historical execution experiences. A probabilistic plan selection function can then select the most appropriate plan among other applicable ones in the given context. [98] introduced a Jason-based framework for integrating Reinforcement Learning (RL) into BDI agents. The Jason-RL framework supports the design of BDI agents, in which some of the plans are programmed

by developers, and others can be learned during the development/engineering stage. Closely related to our work is the Case-Based Reasoning (CBR) BDI system of [99]. A CBR-BDI architecture enables the agent to learn from past cases stored in a case memory. Unlike other works, if there are no similar cases in the case memory, a CBR-BDI agent resorts to concept hierarchy (keywords representing areas of expertise) to explore information on the WWW.

3.7.4 Process Mining

Process mining has been the subject of considerable attention in business process management. Much of this attention has focused on discovering and analyzing what the organization is doing based on its event data [100]. [84] introduced a formal algorithm (called the alpha algorithm) to rediscover workflow models in terms of Petri nets based on event logs. Many authors subsequently expanded this work to analyze various aspects of business processes. [101] applied decision trees to analyse choices in business processes (i.e., decision point analysis) based on discovered process models (e.g., models discovered using a process mining algorithm called the alpha algorithm) and data attributes. [102] have also used a process mining algorithm and machine learning techniques to discover process models with decision points. [103] proposed an approach to learning the prioritization orders of resources that can be used to discover how resources select and prioritize their work items. Their approach supports three disciplines of queuing: FIFO, LIFO, and Priority. [104] introduced an extensible software framework to extract changes in resource behaviour over time using time series analysis. [105] applied decision trees and queuing heuristics to discover resource scheduling protocols in service processes.

Additional related work that applied process mining techniques to automate the recognition of agent-based models is presented in [106], [107], [108], and [109]. Fundamentally, these works employ process mining algorithms, such as the alpha algorithm, to generate workflow nets by which agent models are inferred.

3.8 Chapter summary

In this chapter, we have proposed a novel approach to re-crafting Digital Twins of organizations. A key novelty of our method is the fusion of two powerful research disciplines: reverse engineering (to re-create the visible behaviour of the target system) and explainable agency (for viewing the target system's behaviour through the lens of BDI agents). The abductive design of Digital Twins of organizations was formulated as follows: "Find a BDI agent program that best explains the behaviour of a target system based on its observation logs." We have described three explanation techniques to model a Digital Twin as a BDI agent: (i) Abductive plan recognition, (ii) Explaining by beliefs, and (iii) Explaining by selection strategies. To do so, unlike the current norm in the explainable agency literature, we have provided a complete account of the anthropomorphic handles (e.g., beliefs, desires and intentions). Also, we have leveraged a number of practical strategies for goal generation and deliberation and applicable plan selection (e.g., Round-robin scheduling) to explain the rational behaviour of the target system. This, in turn, allowed for using off-the-shelf agent-oriented platforms (e.g., Jason [75]) to re-crafting observed behaviour.

At this point, we are improving algorithm (1) to deal with logs obtained from complex real-life organizations, where incompleteness of knowledge and non-determinism might be present, and algorithm (2) to model hypothetical situations. Experimental results and technical details are discussed in chapter 6.

Chapter 4

Leveraging Imperfect Explanations

OPEN environments (e.g., business environments) typically require dynamic execution of plans where organizations must engage in situations that include, for example, re-planning, plan repair, plan reusing, etc. Hence, real-life Plan Recognition (PR) applications are required to deal with different classes of observations, such as exogenous actions, switching between activities, and missing observations. Many approaches to PR consider the above-cited classes of observations, but none have dealt with them as deliberated events. Actually, using existing PR approaches to explain such classes of observations may generate only so-called imperfect explanations (plans that partially explain a sequence of observations). Unlike previous applications to PR, we view imperfect explanations and new classes of observations as bases for PR problems. Our overall approach is to leverage (in the sense of plan editing) imperfect explanations by exploiting new classes of observations. We use the notation of capabilities in the well-known Belief-Desire-Intention (BDI) agents programming as an ideal platform to discuss our work.

Following that rational behaviour of an organization can be theoretically viewed through the lens of BDI agents, we hereafter use the terms “organization" and “agent system" interchangeably.

4.1 Introduction

It is generally accepted that an agent system with practical extensions can carry out tasks that would not otherwise be achievable by its basic reactive system. Of-

ten when the environment is highly dynamic and/or the task is complicated for the basic reactive behaviour of the agent system, developers resort to extending or adding new modules to the agent system. A typical example of extending the basic reactive behaviour of agent models can be found far and wide in the state-of-the-art Belief-Desire-Intention (BDI) paradigm [14], including, but not limited to, extending the architecture with self-awareness [110], automated planning [111] and reconfigurability [112].

Much of the work done on PR involves abductive reasoning (e.g., [113], [114], and [115]), which seeks to abductively infer plans by mapping the observed actions to a plan library. A major drawback to Abductive Plan Recognition (APR) is that target plans are usually not from the plan library (this also applies largely to anthropomorphic systems, since they are able to repair and reuse existing plan designs). This drawback can be due to several reasons, some of which are related to extending the target system with different modules, such as the works presented in [110–112]. Actually, appealing to APR applications to explain such observed actions would only generate imperfect explanations. An imperfect explanation is one that partially explains a sequence of actions. Another notion of PR is discovering plans by executing action models to best explain the observed actions. Nevertheless, this can take a great deal of time in complex problems. Our approach is a third way which does not align itself with either notion. Still, it can potentially improve the explanatory power of plan libraries without the need for action model execution.

In this chapter, we address the problem of *leveraging imperfect explanations*, the task of modifying existing hypotheses to explain an observed sequence of actions. We show that when the target system operates in a domain model known to the observer, imperfect explanations can be a valuable guide to explain unknown plans that involve new classes of observations. Hence, our approach can be seen as a post-processing stage for various single-agent plan library-based PR techniques. To avoid arbitrary modification of hypotheses, we also introduce a classification model that can determine the settings (e.g., noisy or explanatory) in which an unknown plan has been observed.

The remainder of this chapter is structured as follows. Related work is dis-

cussed in Section 4.2 with some pointers to different PR domains. Section 4.3 presents a motivation example with pointers to different scenarios. In Section 4.4, we introduce some preliminaries on the notion of capabilities and BDI agents programming, which are the two main ingredients of this work. We formalize the problem of leveraging imperfect explanations in Section 4.5. Our approach to leveraging imperfect explanations for PR problems is described in Section 4.6 before we conclude and outline future work in Section 4.7.

4.2 Related Work

Real-life PR systems are required to deal with domains in which new classes of observations are frequently encountered. Roughly speaking, there are three noticeable domains regarding new classes of observations in PR problems: (1) Exploratory domains - the observed behaviour is a subject of exploratory and discovery learning (e.g., mistakes, exogenous and repeating activities), (2) Noisy domains - the observed behaviour is characterized by imperfect observability (e.g., extraneous, mislabeled and missing activities), and (3) Open domains - the observed behaviour is characterized by new, deliberated action events (e.g., reconfigured plans). Table 4.1 summarizes how new classes of observations are viewed according to these three PR domains.

Table 4.1: How different PR domains view edits in plan execution.

Domain	Exploratory	Noisy	Open
Classes of observations	Match	Match	Match
	Exogenous	Extraneous	Inserted
	Mistake	Mislabeled	Replaced
	Mistake	Missing	Deleted

We concentrate our review on how existing PR approaches viewed and handled new classes of observations. We then use these classes as classification criteria in later sections where we describe our approach for leveraging imperfect explanations.

We first describe works that assume exploratory domains. With the intention of inferring students' plans, Mirsky et al. [115] proposed a heuristic plan recognition

algorithm (called CRADLE) that incrementally prunes the set of possible explanations by reasoning about new observations and by updating plan arguments, in which explanations stay consistent with new observations. Uzan et al. [116] introduced an off-line PR algorithm (called PRISM) to recognize students' plans by traversing the plan tree in a way that is consistent with the temporal order of students' activities. Amir et al. [117] proposed an algorithm (called BUILDPLAN) based on recursive grammar to heuristically generate students' problem-solving strategies.

Many prior approaches to PR focused on dealing with noisy domains. Massardi et al. [118] classified noise in PR problems into three types: missing observations, mislabeled observations and extraneous actions and proposed a particle filter algorithm to provide robust-to-noise solutions to PR problems. Sohrabi et al. [119] transformed the PR problem into an AI planning problem that allows noisy and missing observations. Ramírez and Geffner [85] proposed a probabilistic plan recognition approach, in which a classical planner is used to produce plans for a given goal G and compare these plans to the observed behaviour O in noisy settings. The probability distribution $P(G|O)$ can be then computed by how the produced plans are close to the observed behaviour. Sukthankar and Sycara [120] proposed an approach for pruning and ranking hypotheses using temporal ordering constraints and agent resource dependencies.

PR systems are also required to deal with open domains, where the observed plans are usually not from the used plan library. Avrahami-Zilberbrand et al. [121] described two processes for anomalous and suspicious activity recognition: one using symbolic behaviour recognizer (SBR) and one using utility-based plan recognizer (UPR), respectively. Mainly, SBR filters inconsistent and ranking hypotheses, while UPR allows the observer to incorporate his preferences as a utility function. Zhuo et al. [17] also addressed the problem of PR in open domains using two approaches: one using an expectation–maximization algorithm and one using deep neural networks. A notable difference from other approaches is that the work of f Zhuo et al. [17] is able to discover unobserved actions by constantly sampling actions and optimizing the probability of hypotheses.

Our work is not competing to, but complementing most of the previous works

on PR, where it can be seen as a post-processing task for various single-agent plan library-based PR techniques. More precisely, leveraging imperfect explanations for PR problems can be viewed as an activity that occurs just before ruling out imperfect explanations or considering an observation as exploratory or noisy. In contrast with the literature, we focus on improving the explanatory power of plan libraries by leveraging imperfect explanations and exploiting new classes of observations.

4.3 Running Example

As a running example, we consider the Request For Quote (RFQ) process developed to buy specific goods or services. The reader can view the task RFQ as a goal that is delegated to an intelligent agent by the target organization, such as in [122].

Example 4.1. As shown in Figure 4.1, the RFQ starts with the agent advertising the need for some required goods (gds). After receiving all the quotes (qtd) from prospective vendors, the agent selects a vendor to supply the goods. Once the goods are delivered, the agent sends the payment. Also, the agent is also equipped with capabilities related to the FRQ domain problem.

```
@p14 +!request(qtd,x): not stocked(x)
                <- !promote(rfq);
                select(vendor);
                receive(gds);
                pay(vendor).

@p8  +!promote(x): not promoted(x) <- adver(x).          .
```

```
{not promoted(x),prepared(rfq)} adver(x) {promoted(x)}    .
{valid(vendor)} select(vendor) {publised(qtd)}
{publised(qtd)} receive(gds) {stocked(x)}
{received(gds),matched(gds,qtd)} pay(vendor) {sent(payment)}
{not competitive(qtd)} negotiate(vendor) {competitive(qtd)}
{avaliable(truck) } pick_up(x) {stocked(x)}
```

Figure 4.1: RFQ plan and some known capabilities

RFQ is supposed to be a straightforward business process. However, in business environments, agents must handle exceptions and exploit opportunities. We argue that existing APR applications are inadequate in such settings. To illustrate this, let us consider the following example.

Example 4.2. Consider the following scenarios that may arise in the RFQ:

i Before receiving the goods, the agent decided to negotiate the quote further with the vendor.

ii Instead of waiting for the delivery to arrive, the agent decided to pick up the goods from the vendor's warehouse.

iii The agent selected a vendor without prior advertising for RFQ, and even before receiving quotes from other prospective vendors.

Although simple, Example 4.2 is far from trivial. First of all, it is not difficult to recognize that the basic reactive behaviour of the BDI agent system (described in chapter 2) cannot produce the behaviours depicted in these scenarios on its own, since it does not have those plans in its plan library. Arguably, there is at least an extension to the system that enabled such edits in the agent behaviour. Indeed, the state-of-the-art BDI agent framework exhibits a large number of extensions to the reactive behaviour of the BDI agent system. For example, the behaviour illustrated in scenario (i) involves an insertion edit (where the agent added negotiation into the plan). It is possible for the agent to add an action(s) to a plan by incorporating automated planning into its system, such as in [111] and [123]. Scenario (ii) involves a substitution edit (where the agent replaced waiting for the delivery to arrive with collecting the goods from the vendor's warehouse). It is possible for the agent to replace a capability with another one by leveraging an extension such as reconfigurability [112]. Finally, scenario (iii) represents a deletion edit (where the agent passed the advertising of RFQ), which is doable if the agent is equipped with a task aborting mechanism, such as the one presented in [124].

4.4 Preliminaries

This Section reviews the prior research used in the remainder of this chapter. First, we clarify the link between the notion of capability and BDI programming, and then we describe the representation we use for capabilities and plans.

4.4.1 BDI Programming and Capabilities

An agent system with BDI architecture [14] commonly consists of a belief base (what the agent knows about the environment), a set of events (desires that the agent would like to bring about), a plan library (a set of predefined operational procedures), and a base of intentions (plans that the agent is committed to executing).

It is useful at this point to revisit AgentSpeak(L) programming language to clarify the link between the notion of capability and BDI programming. Fundamentally, the reactive behaviour of BDI agent systems includes the agent system handling external and internal events by selecting an event to address from the set of pending events, selecting a suitable plan from the plan library, and stacking its program into the intention base. A plan in the plan library is a rule of the form $\varepsilon : \nu \leftarrow \rho$, where the program ρ is a predefined strategy to handle the event ε whenever the context condition ν is believed to be true by the agent. A plan can be selected for handling an event ε if it is relevant and applicable, i.e., designed with respect to the event ε, and the agent believes that the context of the plan ν is a logical consequence of its belief base, respectively. A program ρ often can be presented as a set of actions that result in changes in the environment state. For the purpose of this work, we ignore other elements (e.g., trigger events or guards) in the plan body. Note that using BDI programming languages such as Jason and 2APL, information on action pre- and post-conditions can only be defined in a separate file (called simulated environment), making this information invisible to the agent.

To reason about actions and their specifications, we need to access information about the preconditions and postconditions of all available actions. We shall refer to *capability* as an explicit specification of action preconditions and postconditions. A capability has been understood in intelligent agent studies as having at least one

way to achieve some state of affairs, where it can be used only if its preconditions are believed to be true [110, 125, 126]. For the purposes of this work, we shall concentrate mostly on the plan library. We do not, therefore, discuss other issues related to integrating the notion of capabilities into the BDI paradigm. For a detailed introduction to integrating the notion of capabilities into the BDI system, the reader is referred to [110, 125].

4.4.2 Capability and Plan Representation

Our representation of agent capabilities is closely related to [110, 125], but significantly influenced by the action theory found in classical automated planning, such as what has been presented in situation calculus [127] and STRIPS reasoning [128]. Following this representation, we use a language of propositional logic $\mathcal{L}$ over a finite set of literals $\mathsf{L} = \{\mathsf{l}_1, \ldots, \mathsf{l}_n\}$ to represent the set of states of the environment S in which the agent is situated, such that each state of the environment $\mathsf{s} \in \mathsf{S}$ is a subset of L, i.e., $\mathsf{l}_i \in \mathsf{s}$ defines that the propositional literal l_i holds at the state s. As mentioned before, a capability specification describes an action that the agent can carry out along with its pre- and post-conditions. Notationally, capability specification is triple subsets of L, which can be written as a rule of the form $\{\mathsf{pre(c)}\}\mathsf{c}\{\mathsf{post(c)}\}$, where

- $\{\mathsf{pre(c)}\}$ is a set of predicates whose satisfiability determines the applicability of the capability,
- c is the capability, and
- $\{\mathsf{post(c)}\}$ is a set of predicates that materialize with respect to the execution of the capability.

It is not hard to see that, by sequentially grouping capabilities that are dedicated to bringing about some state of affairs, the sequence $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ can be seen as an operational procedure to resolve that state. Consider the plan p, we use

- $\{\mathsf{pre(p)}\}$ as the conditions under which the plan is applicable,
- $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ as the plan body, and

– $\{\mathsf{post}(\mathsf{p})\}$ as the conditions associated with the end event of the plan,

As such, we use the notation $\mathsf{p} = \{\mathsf{pre}(\mathsf{p})\}\mathsf{C}\{\mathsf{post}(\mathsf{p})\}$ to represent plan p specifications.

For the purpose of reasoning about the execution of agent capabilities, we work with a simple representation of the possible ways the plan can be executed, which we term normative plan traces. A normative plan trace is one such that (1) the specification of capabilities completely determines the transition on states in S, i.e., if $\mathsf{s} \in \mathsf{S}$ and c is applicable to s, then it produces another state $\mathsf{s}' \in \mathsf{S}$, (2) the capabilities are guaranteed to execute sequentially, e.g., knowing that capability c_2 immediately follows capability c_1, then c_2 cannot be executed until $\mathsf{post}(\mathsf{c}_1)$ holds, and (3) a plan execution can not be interleaved with other plans.

4.5 Problem Formulation

A plan library H is a set of plans, each of which contains a sequence of capabilities $\langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ as its body, where each c_i, $1 \leq i \leq n$, is the capability name and a list of typed parameters. We assume the presence of a capability library, denoted by $\mathscr{A}$, comprises the set of all available capabilities specifications related to the domain problem. An observation of an *unknown plan* $\overline{\mathsf{p}}$ is denoted by $\mathsf{O} = \langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle$, where $\mathsf{o}_i \in \mathscr{A} \cup \{\varnothing\}$, i.e., the observation o_i is either a capability in $\mathscr{A}$ or an empty capability $\varnothing$ that has not been observed. Note that the plan $\overline{\mathsf{p}}$ is not necessarily in H, and thus mapping O to the H may generate only imperfect explanations.

When reasoning about new classes of observation, one can classify an observed capability by four types: (1) match, when the capability is correctly observed, (2) insertion edit, when the observed capability is added to a normative plan trace, (3) deletion edit when the observed capability is dropped from a normative plan trace, (4) substitution edit, when a capability that is to be executed as part of a normative plan trace is replaced with another one. We propose to describe these three edits in plan execution using operations as follows.

Definition 4.1 (Abductive edit operation, sequence). Let p with $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ as its body be an imperfect explanation for the observation $\mathsf{O} = \langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle$. An

abductive edit operation is the insertion, deletion, or substitution of capabilities in C according to the observations in O. An abductive insertion of an observed capability o_i is denoted by $(\varnothing \rightarrow o_i)$, deletion of c_i is denoted by $(c_i \rightarrow \varnothing)$ and substitution of c_i with o_i is denoted by $(c_i \rightarrow o_i)$. An abductive edit sequence $AES = \langle ae_1, \ldots, ae_n \rangle$ is a sequence of abductive edit operations. An AES derivation from C to O is a sequence of sequences $C_0, \ldots, C_n$, such that $C_0 = C$ and $C_n = O$ and for all $1 \leqslant i \leqslant n$, $C_{i-1} \rightarrow C_i$ via ae_i.

Definition 4.2 (Extended plan library). Given H and AES, an extended plan library is a couple $EPL = (H, AES)$, where

1. H is a plan library, and
2. AES is a sequence of abductive edit operations.

Definition 4.3 (APR problem). Considering our settings, the APR problem is then can be defined by a 4-tuple $APR = (EPL, O, \text{explain}, \mathscr{A})$, where:

1. EPL is an extended plan library,
2. O is an observed trace of capabilities,
3. explain is a map from plans and sub-plans of H to subset of O, and
4. $\mathscr{A}$ is a library of capability specifications.

As such, the solution to APR is to discover an unknown plan $\overline{p}$, which is a plan with an edited sequence of capabilities as its body that best explains O given EPL and $\mathscr{A}$. Again, this can be challenging since the plan $\overline{p}$ is not necessarily in H, and thus mapping O to the H may generate only imperfect explanations.

4.6 Approach

Our approach to leveraging imperfect explanations consists of four phases, as shown in Figure 4.2. These phases are:

1. **Classification of unknown plans**. To avoid arbitrary leveraging of hypotheses (i.e., we do not want to build on noisy or exploratory observations), we

introduce a classification model that can determine the settings (e.g., noisy or explanatory) in which an unknown plan has been observed.

2. **Abductive editing**. Appealing to the notion of optimal edit distance, we propose a simple procedure to "modify" an imperfect explanation according to the observed unknown plan. This step is doable *iff* it turns out that the observed edits are not noisy nor explanatory.

3. **Validity checking**. Abductive editing is not as straightforward as the above count suggests, where we must check the consistency of the edited plan and whether the new sequence of capabilities can still handle the original target event.

4. **Abductive updating**. To improve the explanatory power of plan libraries by leveraging imperfect explanations and exploiting new classes of observations, we propose an algorithm to merge existing hypotheses (i.e., imperfect explanations) with unknown plans. The algorithm improves the explanatory power of the plan library while maintaining its earliest construct.

We describe each of these steps in greater detail in the following sub-sections.

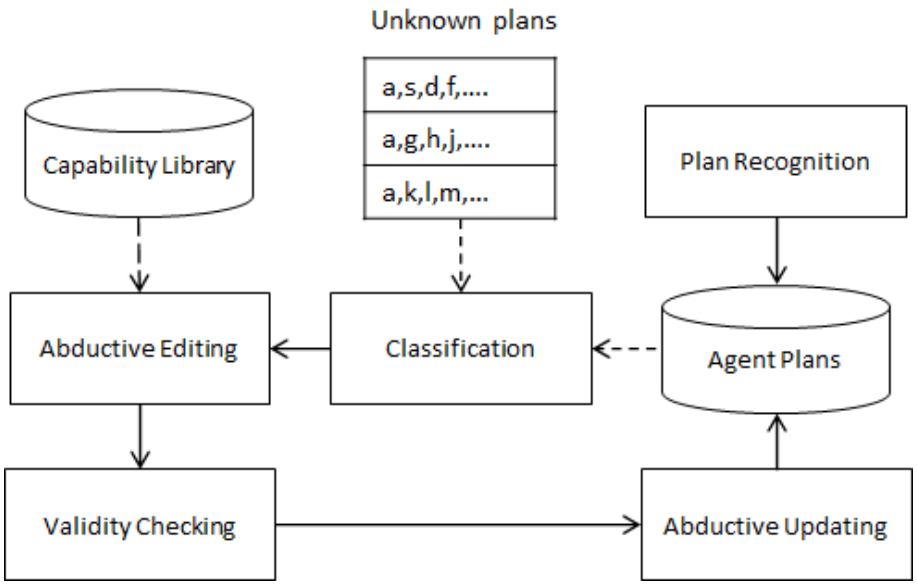


Figure 4.2: Overview of leveraging imperfect explanation approach

As illustrated in Figure 4.2, leveraging imperfect explanations takes as inputs

1. An unknown plan involves new classes of observations that we want to explain using an imperfect explanation,

2. An approximation of the plan (i.e., an imperfect explanation) that has been used to generate input (1), and

3. A set of all available capabilities specifications related to the domain problem.

While inputs (1) and (2) are used in the classification and updating phases, inputs (2) and (3) are used in the editing and validation phases.

4.6.1 Classification

Given an unknown plan, before any decision can be made concerning leveraging imperfect explanations, it is first necessary to determine the characteristic of the environment in which the unknown plan has been carried out (i.e., we do not want to build on noisy or exploratory observations). To that end, we use decision tree learning to classify unknown plans. Following the classification described in Section 4.2, the following taxonomy for classification is proposed:

EE Exploratory environment - observed behaviour is a subject of exploratory or discovery learning. Much of the work done on PR for exploratory domains considers trial-and-error, activity repeating and interleaving as features of exploratory behaviours [115–117].

NE Noisy environment - observed behaviour is characterized by imperfect observability. Previous studies (e.g., [119] and [118]) reported noisy observations as those that cannot be explained by the actions of any plan for any given goal (computing all possible plans for a given goal is fully described in [114]).

OE Open environment - observed behaviour is characterized by by new, deliberated action events. Many studies on intelligent agents (e.g., [112, 129, 130]) consider rational changes in plan execution (see Section 4.5.3 for how rational changes are validated) as a feature of engaging the agent with open environments.

A given unknown plan is classified into one of the classes {**EE, NE, OE**}, each representing the settings in which the unknown plan has been executed. An unknown plan is assigned to a class membership based on its characteristic features by comparing it to its imperfect explanation (i.e., approximation of the unknown plan). For example, unknown plans containing actions that any plan for any given goal cannot explain are labelled as **NE**, while unknown plans with rational edits compared to their imperfect explanations are labelled as **OE**. Historical instances are labelled manually while the test data is not labelled, so the decision tree can classify whether the unknown plan is a result of **EE**, **NE**, or **OE**.

According to state-of-the-art PR and intelligent agents [112, 114–119, 129, 130], we initially extracted a number of features related to **EE**, **NE**, and **OE**.

1. **Unreliable action**: This binary feature represents whether an unknown plan contains action(s) that any plan for any given goal cannot explain.

2. **Trial-and-error**: This binary feature represents whether an unknown plan contains multiple attempts to achieve a desirable effect using different activities.

3. **Action repeating**: This binary feature represents whether an unknown plan contains multiple attempts to achieve a desirable outcome using the same activity with different parameters.

4. **Activity interleaving**: This binary feature represents whether an unknown plan contains the execution of an activity while waiting for the results of the current activity.

5. **Rational editing**: This binary feature represents whether an unknown plan contains rational edit(s) compared to its normative plan traces.

Classification takes place as a supervised multi-class classification making use of the features described above. Figure 4.3 represents a decision tree trained to discriminate **OE** from **EE** and **NE**. For each PR problem, the classification task compares the unknown plan and its imperfect explanation by checking for the above-cited features. This decision tree can be understood as follows: "Rational

edits predict **OE**, whereas unreliable actions predict **NE**. Activity interleaving predicts **EE**, as do action repeating and trial-and-error."

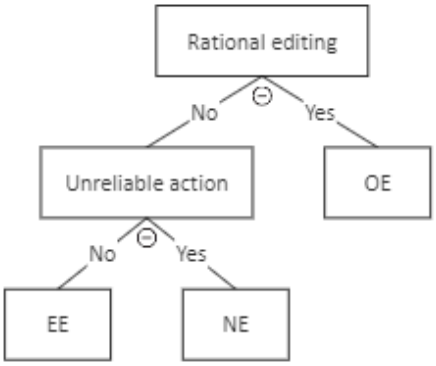


Figure 4.3: A Decision tree trained to predict the domain of unknown plans.

In this work, we have seen one possible way of classifying changes in plan execution by attending to the settings in which they have been carried out (i.e., noisy, explanatory and open). However, in fact, many other features can be adopted to classify these changes hence a better understanding of what the target agent is actually doing and why. For example, these changes can be divided into mistakes and exploratory activities in exploratory domains. Another example, noisy observations can be classified as sensor failure or programming errors (e.g., inappropriate belief revision). However, such classifications are outside the scope of the present work since this is, in general, an intractable problem.

Example 4.3. Continuing with our running example and using the decision tree described above, the sequence below was classified as **EE**. This is due to containing the feature of action repeating, which strongly correlates to **EE**.

```
adver(x),adver(x),adver(x),select(vendor),receive(gds),pay(vendor)
```

Whiles, the scenario below was classified as a **NE** because it could not be explained by any plan of the given goal.

```
adver(x),improve(x),receive(gds),pay(vendor)
```

Classification of unknown plans is applied before leveraging imperfect explanations to avoid useless wait (i.e., we do not want to build on noisy or exploratory

observations). If an unknown plan is classified as **OE**, the plan will be taken as input.

4.6.2 Abductive Editing

Our guiding intuition here is that a plan that serves as an imperfect explanation for an observed behaviour could be edited (modified) according to that observation, thus improving the explanatory power of the plan library. We realize a computational solution to leveraging imperfect explanations by appealing to the optimal edit distance between an unknown plan and its imperfect explanation and using its corresponding edit sequence.

Let plan p with body $C = \langle c_1, \ldots, c_n \rangle$ be an imperfect explanation of the observations $O = \langle o_1, \ldots, o_m \rangle$, with the former having length n and the latter length m. Recall that turning the plan body C into O requires a sequence of edit operations. Each operation can be weighted by a cost function, denoted by $w(ae)$. For example, one can set the cost function to return 0 when the capability is correctly observed and return one otherwise.

With a cost function in hand, the abductive plan edit distance between C and O is given by a matrix d of size $n \times m$, defined by the recurrence

$$d[i,0] = i$$
$$d[0,j] = j$$

$$d[i,j] = \begin{cases} d[\text{i-1,j-1}] & \text{(correctly observed)} \\ \min \begin{cases} d[\text{i-1,j}] + w(\varnothing, o_i) & \text{(insertion edit)} \\ d[\text{i,j-1}] + w(c_i, \varnothing) & \text{(deletion edit)} \\ d[\text{i-1,j-1}] + w(c_i, o_i) & \text{(substitution edit)} \end{cases} \end{cases}$$

After filling the matrix, the value in the bottom-right cell of the d, or $d[m,n]$, will represent the minimum cost to turn the plan body C into the observation sequence O, and this cost is the abductive plan edit distance. For the corresponding AES to be computed, we need to trace back the choices that led to the minimum edit cost

in the above recurrence. Hence, turning the imperfect explanation p into an unknown plan $\bar{p}$ that best explains the observations in O can be seen as applying an AES that corresponds to the abductive plan edit distance of p.

Example 4.4. Continuing with our running example, assume the imperfect explanations h1 with a body as shown below:

```
adver(x),select(vendor),receive(gds),pay(vendor)
```

An observation sequence O from scenario (ii) as shown below:

```
adver(x),select(vendor),pick_up(gds),pay(vendor)
```

And let $w(ae) = 0$ when the capability is correctly observed and $w(ae) = 1$ otherwise. Based on the above inputs, the abductive edit operations needed to turn the body part of plan p14 into O are:

- $receive(gds) \rightarrow pick\_up(gds)$,

Example 4.4 illustrates how the imperfect explanation p14 can be modified to explain an unknown plan involves a replacement edit. Although the required edit in Example 4.4 sounds rational, it may be invalid in other scenarios. Hence, two important questions are yet to be discussed: how to ensure (1) that the edited plan is consistent and (2) at the end of its execution, the goal is achieved. We will address these two questions in the following sub-section.

4.6.3 Validity Checking

When an edit is made to a plan, it brings into question whether the plan being modified rests valid. We assume that plans are drawn based on the specifications in $\mathscr{A}$. The task of plan validity check involves: (1) checking whether the execution of an edited plan will reach the defined goal, (2) whether the preconditions of each step of the plan execution will be satisfied.

We build on the approach of monitoring plan validity proposed by [131]. Our theory of edited plan validity uses the accumulative effects of [132] and ensures consistency during the process of plan editing, given the capability library $\mathscr{A}$. To

monitor an edited plan validity, two plans are generated for every PR problem - one with the minimum edit cost, i.e., the imperfect explanation, and one that explains O completely, i.e., $\mathsf{explain}(\overline{p}) = O$. Assuming an idealised execution environment, plan validity can be defined as follows.

Definition 4.4 (Valid Plan). Consider the capability library $\mathscr{A}$ and a plan specification $p = \{\mathsf{pre}(p)\}C\{\mathsf{post}(p)\}$ for plan p with body $C = \langle c_1, \ldots, c_n \rangle$, we say that the plan p is valid in the state s if

1. $\mathscr{A} \models (\langle \mathsf{pre}(c_1), \ldots, \mathsf{pre}(c_n) \rangle, s)$, and
2. $\mathsf{accum}(p) \models \mathsf{post}(p)$.

As such, with respect to the library $\mathscr{A}$ and the current state of the world s, the preconditions of each capability in the plan body of p will be satisfied (here $\models$ means logical entailment), and the final effect scenario associated with the end state event of the plan p execution entails its post-condition specifications.

An important detail of this definition is that a single edit can impact the plan's consistency. Furthermore, it can change the final effect scenario associated with the end state of the plan. We now identify a valid edited plan.

Definition 4.5 (Valid edited plan). Consider the plan $p = \{\mathsf{pre}(p)\}C\{\mathsf{post}(p)\}$ as an imperfect explanation of O. Let $\overline{p}$ be an edited plan that has identical pre- and post-conditions to the imperfect explanation p except that $\overline{p}$ has an edited sequence of capabilities $\overline{C}$. We say that the edited plan $\overline{p}$ preserves the validity of p if $\overline{p} = \{\mathsf{pre}(p)\}\overline{C}\{\mathsf{post}(p)\}$ is valid.

For example, it does not matter by which means the goods arrive to the agent (e.g., pick up or delivery) for $\overline{p}$ to preserve $h14$ validity as long as the goal of requesting for a quote is achieved under the same context conditions.

4.6.4 Abductive Updating

Now, we consider the problem of what needs to be done to improve the explanatory power of the plan library when an edited plan is found valid according to the checks described in the previous section. An easy solution is to create a new plan

that has identical triggering and context parts to the imperfect explanation, except that $\overline{p}$ has an edited body $\overline{C}$ that explains O, i.e., $\text{explain}(\overline{p}) = O$. However, this may increase the complexity of determining applicable plans at run-time. More interestingly, we offer a semi-automated solution for merging an edited plan with its imperfect explanation.

Procedure 4.1 (Abductive Updating) Consider the imperfect explanation p and its edited plan $\overline{p}$ and let $AES = \langle ae_1, \ldots, ae_n \rangle$ be a derivation from p to $\overline{p}$.

Procedure 4.1. Abductive Updating

```
1:  For each ae_i in AES:
2:  Replace c_i with subgoal sg_i
3:  Create two sub-plans p1 and p2, and let
4:  triggering(p1) = triggering(p2) = sg_i
5:  switch(ae_i):
6:    case(c_i → o_i)
7:      context(p1) = pre(c_i)
8:      context(p2) = pre(o_i)
9:      body(p1) = c_i
10:     body(p2) = o_i
11:   case(c_i → ∅)
12:     context(p1) = pre(c_i)
13:     context(p2) = not pre(c_i)
14:     body(p1) = c_i
15:     body(p2) = true
16:   case(∅ → o_i)
17:     context(p1) = not pre(o_i)
18:     context(p2) = pre(o_i)
19:     body(p1) = true
20:     body(p1) = o_i
```

Procedure 4.1 facilitates the merging between an imperfect explanation, i.e., one with the minimum edit cost, and its edited plan, i.e., one that explains O completely. Fundamentally, for each ae_i in AES, the procedure replaces the corresponding capability with a sub-goal sg_i (line 2), for which two sub-plans p_1 and p_2 are created (line 3-4). Our guiding intuition behind creating two sub-plans is to improve the explanatory power of the plan library while maintaining its construction.

Recall that there are three ways in which a plan can be edited: deletion, insertion, and substitution. Hence, there are three ways in which sub-plans can be created (line 5). For example, let us consider the substitution edit (line 6): In this

case, plan p_1 takes the corresponding capability c_i as its body and $pre(c_i)$ as its context part (lines 7 and 9). While p_1 takes the corresponding observation o_i as its body and $pre(o_i)$ as its context part (lines 8 and 10). However, there are situations where valid edited plans could possibly be merged with their imperfect explanations, but they should not be merged. For example, avoiding negative interactions between goals. For readers interested in how we can deal with the feasibility of plans merging, we refer to [133].

4.7 Chapter summary

Much of the work done on APR requires a plan library to infer the top-level plans of the observed system. Nevertheless, in open environment settings, target plans are usually not from the plan library due to reusing plans, replanning and agent self-awareness, etc. To that end, this work builds on a more sophisticated notion of APR, which seeks to improve the explanatory power of plan libraries by way of leveraging imperfect explanations and exploiting new classes of observations.

In this chapter, we proposed a classification of unknown plans based on the characteristics of the domains in which they have been carried out. As far as we know, this has been absent in PR research. Furthermore, we presented a theory based on capabilities and plans and introduced the notion of abductive plan editing. Finally, we described how imperfect explanations could be updated with new classes of observations in a rational fashion.

A number of extensions of this work are of interest, including applications of plan library reconfigurability [112], plan editing in online settings, and dealing with incomplete action models (i.e., learning unknown activities). Experimental results and technical details are discussed in chapter 6.

Chapter 5

Explanation Mining

Explanation generation is the task of justifying the decisions of a target system after observing its behaviour. Much of the previous explanation generation approaches can theoretically do so but assuming the availability of explanation generation modules, deterministic execution of plans and reliable observations. However, in real-life settings, explanation generation modules are not readily available, plans are non-deterministic, and unreliable observations are frequently encountered. We seek in this chapter to address these challenges by proposing a data-driven approach to mining and validating explanations (and specifically belief-based explanations) of the past executed actions of a target system. Our approach leverages the historical data associated with the system execution, which describes action execution events and external events (represented as beliefs).

Following that rational behaviour an organization can be theoretically viewed through the lens of BDI agents; we hereafter use the terms “organization" and “agent system" interchangeably.

5.1 Introduction

Explainable agents have been the subject of considerable attention in recent literature. Much of this attention involves folk psychology [78], which seeks to explain the actions of an agent by citing its mental state (e.g., the beliefs of the agent, and its goals and intentions). Roughly speaking, when an explainee requests explanations about a particular action, two common explanation styles might be adopted:

(1) a goal-based explanation and (2) a belief-based explanation [79, 80, 86]. This chapter focuses on the latter style[a] in the context of the well-known Belief-Desire-Intention (BDI) paradigm [14]. Fundamentally, belief-based explanations help answer the following question: *What must have been known for the target system to perform a particular action over another?* Arguably, a sufficiently detailed explanation (e.g., one that justifies an action with extensive information about the reasoning of the target system) will require no additional information to answer this question. Nevertheless, in specific settings (e.g., explaining by UD S_0 in chapter 3 and time-constrained environments), explanations are more useful when they are relatively unfaithful [134, 135]. Explaining by beliefs can also help solve a range of problems, such as encouraging behaviour change [86], enhancing human-agent teaming [136], and transparency [137].

Although there is a large and growing body of work on explanation generation in the field of explainable agency [78], much of this work has traditionally assumed the availability of explanation generation modules (e.g., the target system is explainable by design), reliable observations and deterministic execution of plans. However, in real-life settings, explanation generation modules are not readily available, unreliable observations are frequently encountered, and plans are non-deterministic. We seek in this chapter to address these challenges by proposing techniques that mine the historical data associated with the system execution to generate belief-based explanations of the past actions of that system. We shall refer to this problem as *mining belief-based explanations*.

Our proposal relies on audit logging (or instrumenting the business environment with off-the-shelf market monitors). Many existing MAS frameworks (e.g., JACK framework [73]) support logging different aspects of agent behaviour. Of these, we are interested in two particular aspects: (1) a behaviour log that records the creation and completion of the past executed actions, and (2) an event log that describes the belief set activity of the target system during the recording of (1). One such implementation of audit logging is the tracing and logging tools for JACK Intelligent Agents [73]. Our approach to mining belief-based explanation

[a] goal-based explanations are also of great value to our general research question, and we confirm that an extension of the techniques presented in this chapter can address these, but are outside the scope of the present work

involves two steps:

1. We leverage these two audit logs in chronological order to generate one sequence database taken as input by a sequential pattern miner. The intuition behind this is identifying commonly occurring patterns of action execution events preceded by sequences of external events (i.e., beliefs). Here, we intend to mine the enabling beliefs of each action referred to in the behaviour log.

2. We define a validation technique that leverages a state update operator (i.e., an operator that defines how the specification of a belief state is updated as a consequence of the agent's perception of the environment), agent's past experiences provided by the above-cited sequence database to compute the soundness and completeness of the mined belief-based explanations.

Mining and validating belief-based explanations can be outlined as follows: Given as inputs

1. A behaviour log of the past executed actions of the target system that we want to explain,

2. An event log of the past external events or information that other industry players would expect the target system to be aware of,

3. A plan or plans whose execution generate these logs (recall that these can be abductively recognized as described in chapter 3), and

4. A state update operator.

Compute: the belief-based explanations of every action referred to in the behaviour log. While inputs (1) and (2) are used for mining belief-based explanations, inputs (3) and (4) are used for validating the mined explanations. As we show later in this chapter, inputs (3) and (4) can also be used to generate detailed belief-based explanations.

The remainder of this chapter is organized as follows. Section 5.2 introduces our running example and some required preliminaries. Section 5.3 describes our approach to updating belief-based explanations, which sits at the core of this work.

Section 5.4 describes our approach to mining belief-based explanations. In Section 5.5, we describe how the mined explanations can be validated. Related work is discussed in Section 5.6 before we conclude and outline future work in Section 5.7.

5.2 Preliminaries and Running Example

Agents with BDI architecture are designed to imitate human practical reasoning using beliefs, desires and intentions. The beliefs represent what the agent knows about the environment in which it is situated, the desires are goals (i.e., objectives) that the agent would like to bring about, and the intentions are plans that the agent is committed to executing. With these anthropomorphic handles, the BDI agent derives its actions and, consequently, can explain them. As the literature (e.g., [138]) suggests, this is an elegant means to explain systems with considerable underlying complexity.

Running Example

Although explanations must faithfully reflect the underlying reasoning of the target system (i.e., including its beliefs, goals, plans and intentions), explaining by beliefs can provide value in ways other types of explanations cannot. To illustrate this, we study two scenarios for handling the situation of engine failure on take-off in jet-engine aircraft, which are: (1) accelerate-stop and (2) accelerate-go, as described in [139]. A secondary intent behind this example is to illustrate that our contributions apply to organizations without loss of generality to other anthropomorphic systems.

Example 5.1. As shown in Figure 5.1, the pilot agent has two plans to handle Engine Failure on Take Off (EFTO) in large jet-engine aircraft: (`p1`) accelerate-stop and (`p2`) accelerate-go. The plan `p1` involves reducing thrust, applying speed breaks and notifying the tower of the emergency. The plan `p2` begins with ensuring full power is applied (including mixture, throttle, landing gear and flaps), followed by liftoff, and then notifying the tower of the emergency and the intended landing. To handle an EFTO successfully, two critical speeds must be calculated

before each take-off, namely v1 (the speed at which the pilot can abort the take-off safely) and v2 (the speed at which the pilot can take-off safely).

```
@p1 +efto(aircraft):  v1 < speed < v2
                    <- idle(throttle);
                    deploy(brakes);
                   .send(tower,tell,stop(accelerate)).
@p2 +efto(aircraft):  v1 < speed < v2
                   <- increase(mixture);
                   increase(throttle);
                   take_up(flap);
                   pull(yoke);
                   take_up(gear);
                   .send(tower,tell,go(accelerate));
                   .send(tower,tell,return(landing)).
```

Figure 5.1: AgentSpeak(L) plans for handling EFTO

A particularly stressful situation for the pilot agent is when EFTO occurs between v1 and v2 (i.e., the aircraft is going too fast to accelerate-stop but too low to accelerate-go). For this particular situation, the two plans (p1 and p2) are applicable.

Another example illustrating the value of belief-based explanations has been seen earlier in this thesis (chapter 3), when we discussed **Explaining by UD** S_0. It is useful at this point to recall UD S_0 to clarify the importance of mining belief-based explanations. UD S_0 selects an applicable plan based on hand-crafted conditions. We expect that explaining by UD S_0 is useful, in particular when FIFO S_0 fails to explain the observed behaviour rationally. A straightforward way to solve this is to seek more beliefs to explain each selection and try again to override the S_0 method accordingly. Hence, we argue in this chapter that belief-based explanations (and precisely detailed ones) are also helpful to design an accurate **Explaining by UD** S_0.

Explaining by beliefs

Existing explanation generation techniques can be summarized as follows: A BDI agent triggers an action with respect to its goals and beliefs, which can be rep-

resented in terms of a Goal Hierarchy Tree (GHT) [79, 80, 86]. A GHT is a tree structure representing a high-level abstraction of an agent's reasoning. Figure 5.2 illustrates the GHT of of the pilot agent.

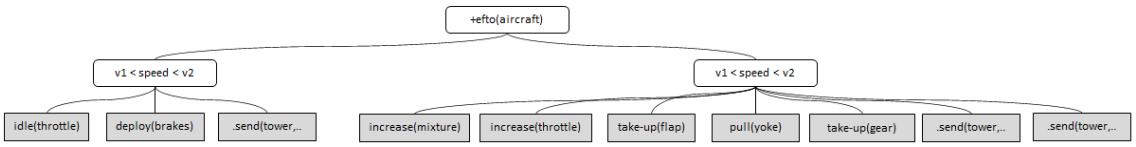


Figure 5.2: A GHT the pilot agent

At the root of the tree, the agent's main goal is placed. A link from the top-level goal to one or more sub-goals means that these sub-goals must be achieved as part of the top-level goal. Tree leaves represent actions that the agent can execute. For the agent to execute an action, certain beliefs placed directly above the action must be true.

What existing explanation algorithms do is select the beliefs and goals that are directly above the selected action node in the GHT to design explanation patterns. We argue that such explanation-generation techniques can involve some ambiguity (e.g, one decision having more than one possible explanation) and irrationality (e.g., different decisions having one explanation) in competitive settings. Furthermore, this is not how BDI agents are practically reason, and it cannot be easy to update the GHT associated with the agent reasoning in sophisticated environments. To illustrate this, consider the following scenarios.

Example 5.2. Assume that the pilot decided to accelerate-stop at one instance and accelerate-go at another instance in the past. Now, consider the following queries that may arise by an aviation candidate:

- *Why did the pilot agent pull the throttle lever to idle immediately after the EFTO?*
- *Why did the pilot agent move the mixture knob to rich after the EFTO?*

Following the current norm of explanation generation, the two queries in Example 5.2 can be readily answered using the goal efto(aircraft) and the belief that v1 $<$ speed $<$ v2. Another way to say that the same goal and same belief explain

both queries. It is clear that these explanations are inaccurate - they do not accurately describe how the pilot agent came to its decision to accelerate-stop at the first instance nor to accelerate-go at the latter instance.

Audit Logs

During a system execution, a wide variety of data on changes in the agent's mental attitude and environment can be represented in the form of audit logs. Collecting such data can be implemented using audit logging tools such as Mind Inspector in Jason platform [75] and Design Tracing Tool (DTT) in JACK platform [73]. We are interested in two modes of audit logging: (1) behaviour logs and (2) event logs. A behaviour log describes the historical execution of plans as sequences of events where each event refers to some action. A behaviour log can be represented as a set of instances, each of which is a set of triples $\langle \mathsf{p\_label}, t_i, a_i \rangle$, where the value of t_i refers to the starting time of the action a_i, which has been executed as part of a plan labelled p_label. An excerpt of the behaviour log associated with the plans in our running example is recorded in table 5.1.

Table 5.1: A behaviour log of the pilot agent.

plan	timestamp	action
p1	t75	idle(throttle)
p1	t77	deploy(brakes)
p1	t80	send(tower, msg)
p2	t1027	increase(mixture)
p2	t1029	increase(throttle)
p2	t1031	take_up(flap)
p2	t1033	pull(yoke)
p2	t1035	take_up(gear)
p2	t1037	send(tower, msg)
p2	t1038	send(tower, msg)

An event log records the history of the external events perceived by the target system. It consists of a set of pairs $\langle t_i, q_i \rangle$, where t_i value indicates the time when the agent added the event q_i to its belief base. Note that event logs [b] record new

[b] One can leverage JACK capability methods to make belief set activities available at agent level [140]. This manipulation allows, in turn, to store enabling beliefs based on the user-defined data structure.

beliefs as they are added to the belief base but do not record persistent beliefs. Determining which beliefs hold at a certain point of system execution, therefore, requires updating machinery (e.g., the state update operator described in Section 5.3). Table 5.2 illustrates an excerpt of the event log describing external events perceived by the pilot agent during the execution of the plans in Figure 5.1.

Table 5.2: An event log of the pilot agent

timestamp	beliefs
t70	runway(dry)
t71	wind(cross)
t72	efto
t73	v1 = 129
t73	v2 = 145
t73	flaps = 15
t74	speed = 135
t76	decelerate(thrust)
t78	steady(aircraft)
t1022	runway(wet)
t1023	wind(head)

timestamp	beliefs
t1024	efto
t1025	v1 = 156
t1025	v2 = 166
t1025	flaps = 15
t1026	speed = 161
t1028	escalate(fuel flow)
t1030	accelerate(thrust)
t1032	flaps = 0
t1034	liftoff(aircraft)
t1036	up(gear)
t1040	liftoff(aircraft)

Usually, it is more convenient to represent beliefs in first-order sentences to maintain consistency and constraints. To avoid handling different groundings of the variables as distinct beliefs, we need to neglect the precise grounding of valuables. Nevertheless, there are settings where we need exact instantiations of the variables.

5.3 Updating Belief-based Explanations

An updated belief-based explanation of any given step in a plan execution is a consistent set of persistent (cumulative) beliefs that explains the execution of that plan if it was executed up to that step. Recall that plans can be abductively recognized given a sequence of actions (e.g., the observations in Table 5.1) and a set of potential plans (e.g., the plans in Figure 5.1), as discussed in Section 3.4.

Updating belief-based explanations is helpful for at least three reasons. First, it could be used to contextualise explanations (i.e., providing users with detailed

explanations of each step in a plan execution). Another way to say that is updated belief-based explanations can help answer the following question for any step of plan execution: *What must have been known in detail for the agent to perform a particular action over another?* As the second reason, it could also be used to validate the mined explanations, which we describe in detail in later sections. Finally, updated belief-based explanations can be a valuable guide to implement **Explaining by UD** S_O.

At each action step in a plan execution, we derive the updated belief-based explanation of an action by combining the enabling beliefs of the preceding steps with the enabling beliefs of the step we are at. We assume that each action in a plan is associated with enabling beliefs (i.e., no provisional execution of actions) written as conjunctive normal-form sentences using a state description language that might involve propositional variables (i.e., variables that can be true or false) and non-Boolean variables (i.e., new value assignments). We allow the updated belief-based explanations to be non-deterministic for two reasons (1) in any plan with OR branching, one might arrive at an action through multiple trajectories, and (2) much of the existing state update operators resolve inconsistencies in multiple different ways. Among the two well-known state update operators in the literature - the Possible Worlds Approach (PWA) [141], and the Possible Models Approach (PMA) [142] - our work relies on the PWA[c]. More precisely, we use the state update operator $\oplus$ as defined below, assuming the presence of a background knowledge base KB.

Definition 5.1 ($\oplus$ Operator). For the two belief states s_i, s_j, and the knowledge base KB, the state update operator $\oplus$ can be defined as follows:

$$s_i \oplus s_j = \{s_j \cup s_i' \,|\, (s_i \wedge s_i' \cup s_j \cup KB \not\models \bot) \wedge (\nexists\, s_i'' \text{ such that } s_i' \subset s_i'' \subseteq s_i \wedge s_i'' \cup s_j \cup KB \not\models \bot)\},$$

in which if $s_j \cup s_i$ is consistent, then the resulting updated explanation is $s_j \cup s_i$. Otherwise, we need to define $s_i' \subseteq s_i$ such that $s_j \cup s_i'$ is consistent and there is no exists s_i'' such that $s_i' \subset s_i'' \subseteq s_i$ and $s_j \cup s_i''$ is consistent. Note that we might need to

[c]One reason for not using PMA is that it may be unable to satisfy the preservation criterion in stochastic settings, as proven in [143].

refer to a general version of the state update operator, i.e., if $S = \{s_1, \ldots, s_n\}$ is a finite set of belief states, then $S \oplus s = \{s_i \oplus s \,|\, s_i \in S\}$. Note that we might need to refer to a general version of the state update operator, i.e., if $S = \{s_1, \ldots, s_n\}$ is a finite set of belief states, then $S \oplus s = \{s_i \oplus s \,|\, s_i \in S\}$. Note also that the output of the state update operator is not always a unique state specification. Actually, the output might be a set of non-deterministic possible belief-based explanations. For the purpose of illustrating why this might be the case, we consider the following example.

Example 5.3. Consider the following knowledge base

$$KB = r \Rightarrow \neg(d \wedge q)$$

representing a rule for the pilot agent, where the propositional letter r can be read as there is an EFTO, the letter d as the thrust is accelerating, and the letter q as the aircraft is ascending. Now, let $(d \wedge q)$ hold in some previous belief state, and r came to be held in the belief state where we are at. Applying $\oplus$, the generated two alternative scenarios describing the updated belief states are

1. $\{d \wedge r\}$ and
2. $\{q \wedge r\}$

which is to say, the rule in KB expresses that whenever the pilot agent believes that there is an EFTO, then it is believed that either the thrust is accelerating or the aircraft is ascending (i.e., the thrust cannot accelerate unless descending after engine failure).

With the intention of obtaining complete detailed belief-based explanations of the agent behaviour, we need to apply the state update operator over each pair of actions in a plan body repeatedly, with the previous updated belief-based explanations associated with the former action as the first argument and the current enabling beliefs associated with the later action as the second argument. However, before we can do so, we need some guidance from plan developers who are at least somewhat familiar with the plan design. This guidance comes in the

form of providing a description of the enabling beliefs of each action (in the absence of an accurate mining model). Recall that a plan body can contain actions and sub-goals, where sub-goals can be combined by either AND (e.g., "achieve sub-goal sg1 and then achieve sub-goal sg2") or OR (e.g., "achieve sub-goal sg1 or achieve sub-goal sg2") operations. On this basis, we defined three procedures to "annotate" actions with belief-based explanations: (1) contiguous actions, (2) AND sub-goals and (3) OR sub-goals as follows:

- **Contiguous Actions**: Let $\langle a_i, a_j \rangle$ be an ordered pair of actions (i.e., the action a_j directly follows a_i), and assume that $b_i = \{p_{i1}, \ldots, p_{in}\}$ and $b_j = \{p_{j1}, \ldots, p_{jn}\}$ be their enabling beliefs, respectively. Recall that a belief set can be seen also as a conjunctive normal-form sentence (i.e., a set of clauses). On condition that the result of $b_i \cup b_j$ is consistent, then the resulting updated belief-based explanation is $b_i \cup b_j$. Otherwise ($b_i \cup b_j$ is inconsistent), then we need to define a synthetic belief state b'_i such that $b'_i = \{p_{ik} \mid p_{ik} \in b_i \text{ and } \{p_{ik}\} \cup b_i \text{ is consistent}\}$. Hence, the resulting updated belief-based explanation would be $b'_i \cup b_j$. Another way to say is that the updated explanation of the contiguous actions would contain the enabling beliefs of the later action a_j plus some of the enabling beliefs of the former action a_i provided that the combination is consistent. Precisely speaking, we need to remove those belief clauses in the updated explanation of the former action that contradict the enabling beliefs of the later action. We assume that the remaining belief clauses are "undones" (e.g., enabling beliefs overridden by the execution of the later action). We shall use $\mathsf{update}(b_i, b_j)$ to refer to the result of an updated belief-based explanation of two contiguous actions a_i and a_j with enabling beliefs b_i and b_j, respectively.

It is important to note that belief-based explanations are updated only within plans (i.e., not within a library of plans). Additional to "annotating" each action in a plan body with its enabling beliefs, we need to compute the updated belief-based explanation (denoted hereafter by β_a) of each action a. An updated explanation β_a can be defined as a set of alternative belief scenarios $\beta_a = \{bs_{a1}, \ldots, bs_{an}\}$, where each bs_{ai} denotes a belief scenario (we use the term "scenario" to indicate the non-deterministic nature of plan execution). As described later in this section, belief

scenarios are introduced by AND sub-goals and OR sub-goals. Hence, we need to make special provisions for updating belief-based explanations across AND sub-goals and OR sub-goals. Now, we describe a simple procedure to be followed in the instance of 2 sub-goals (which can be generalized in a straightforward fashion to n sub-goals):

- **AND sub-goals**: Let $\langle a_1, a_2 \rangle$ be two actions directly precede an AND sub-goal (!sg). Assume that their updated explanations be $\beta_1 = \{bs_{11}, \ldots, bs_{1n}\}$ and $\beta_2 = \{bs_{21}, \ldots, bs_{2n}\}$, respectively. Let b be the enabling beliefs and β the updated explanation of an action a that directly follows the AND sub-goals !sg. We define $\beta = \{update(bs_{1i}, b) \cup update(bs_{2j}, b) \mid bs_{1i} \in \beta_1 \text{ and } bs_{2i} \in \beta_2\}$. For the purpose of this work, we assume that plan designs are not erroneous. Hence, we do not consider cases such that bs_{1i} and bs_{2j} being inconsistent.

- **OR sub-goals**: Let $\langle a_1, a_2 \rangle$ be two actions directly precede an OR sub-goal (!sg). Assume that their updated explanations be $\beta_1 = \{bs_{11}, \ldots, bs_{1n}\}$ and $\beta_2 = \{bs_{21}, \ldots, bs_{2n}\}$, respectively. Let b be the enabling beliefs and β the updated explanation of an action a that directly follows the OR sub-goal !sg. We define $\beta = \{update(bs_{1i}, b) \mid bs_i \in \beta_1 \text{ or } bs_i \in \beta_2\}$.

We noticed that the above-cited procedures might be erroneous when applied to loops. Although loops might be uncommon in a declarative programming style (e.g., AgentSpeak(L) programming language), they still can be useful for building recurrence behaviour. We also noticed that some belief scenarios can be infeasible in some cases. Hence, we do not claim to be able to infer complete belief-based explanations of all the actions referred to in a given plan.

5.4 Mining Belief-based Explanations

Mining belief-based explanations starts with transforming the observations in the audit logs into observation sequences, each of which involves the execution of an action and the manifestations of some external events. Note that logging tools are usually designed to record one mode of observation per log. That is to say, action execution and external events manifestation are recorded in separate logs. To

that end, we need first to start a correlation between the two logs to obtain an observation log that serves as a sequential database, which will be mined to extract belief-based explanations using a sequential rule miner. We define this correlation as follows.

Definition 5.2 (Observation sequence, and log). Let $\{A = a_0, \ldots, a_n\}$ be an actions space, $B = \{b_0, \ldots, b_n\}$ be a belief states space. An observation instance is an alternating sequence of the form $b_0, a_0, \ldots, b_n, a_n$. D_{All} is an observation log, such that $D_{All} \in 2^T$, where T is the set of all observation instances.

Mining belief-based explanations relies on the two following premises: (1) that the external events observed in the event log immediately before executing an action can be the enabling beliefs of that action, and (2) that the persistent beliefs observed a long time before the execution of an action are typically not the enabling beliefs of that action, but may be of that action plus some others. Hence, we use the basic relation "direct successor" [144] over the actions and the external events in the D_{All} as follows:

Definition 5.3 (Direct successor). Let D_{All} be an observation log over T. Let $b, a, b' \in T$.

1. b is a direct predecessor state, denoted by $b >_D a$, *iff* $\langle b, a \rangle$ is a subsequence of a given observation instance of T, and

2. b' is a direct successor state, denoted by $a >_D b'$, *iff* $\langle a, b' \rangle$ is a subsequence of a given observation instance of T.

Relation $>_D$ describes which belief states directly following/preceding a given action. Direct predecessor relation over D_{All} would offer learning entries of the form $\langle b, a \rangle$, where a is an action applicable at the belief state b. For our purposes, we do not distinguish between $\langle q, p, a \rangle$ and $\langle p, q, a \rangle$, because we are only interested in relating actions to their enabling beliefs but not in the relation amongst enabling beliefs. Against this background, we create D_{Direct}, which is a sequence of the following form

$$\langle\langle\langle b_{11}\rangle, .., \langle b_{1n}\rangle, a_1\rangle\rangle, .., \langle\langle\langle b_{i1}\rangle, .., \langle b_{im}\rangle, a_i\rangle\rangle, .., \langle\langle\langle b_{p1}\rangle, .., \langle b_{pk}\rangle, a_p\rangle\rangle\rangle$$

where each $\langle a_{i-1}, a_i \rangle$ represents an ordered pair of actions, and each $\langle b_{i1} \rangle$ represents the observed external events before the execution of action a_i and after action a_{i-1} execution. An exception is required for the first recorded action in D_{All}, as there is no preceding action. In this case, we use the timestamp of the initial high-level event as the start time of the system execution. Given D_{Direct}, we view the problem of mining belief-based explanations as finding all the sequences $\langle b, a \rangle$ that satisfy some predefined measures of interestingness, assuming unique activity execution (i.e., there is no concurrent execution of actions).

Again, we are interested in discovering all the external events that are observed always, or most of the time, directly before the execution of each action referred to in the behaviour log. Association rule learning can be an effective means for discovering regularities (i.e., potential associations) between external events and executed actions. With regard to our application area, an association rule is an implication expression that looks like "the target system must have been known $\langle b \rangle$ to execute the action a". Remark that for an external event to be considered as an enabling belief of an action it has to satisfy some predefined measures of interestingness, which will be discussed later in this section.

Let $L = \{l_1, \ldots, l_n\}$ be a set of distinct distinct literals. Let D_{Direct} be a set of sequences, called a sequence database, such that each sequence consists of a set of literals. Given two itemsets X and Y, the association rule $X \Rightarrow Y$ is an implication states that if the elements in X occur, then it will be followed by the elements in Y. For an association rule to be an interesting one, it must satisfy two measures of interestingness: minimum support (*min supp*) and minimum confidence *min conf* thresholds, which can be computed as follows

- An association rule $X \Rightarrow Y$ holds in D_{Direct} with a certain support (how frequently the rule appears in D_{Direct}) provided by

  $$Support(X \Rightarrow Y) = P(X \cap Y)$$

- An association rule $X \Rightarrow Y$ holds in D_{Direct} with a certain confidence (the accuracy of the rule) provided by

  $$Confidence(X \Rightarrow Y) = Support(X \cup Y)/Support(X)$$

- For an association rule $X \Rightarrow Y$ to be interesting it has to have a support greater than, or equal to, *min supp* and a confidence greater than, or equal to, *min conf*, which are commonly depicted as

  $\mathsf{Support}(X \Rightarrow Y) \geqslant \mathit{min\ supp}$
  $\mathsf{Confidence}(X \Rightarrow Y) \geqslant \mathit{min\ conf}$

We use the well-known CMRules algorithm [145] to discover this form of rules. Fundamentally, CMRules algorithm makes multiple passes over $\mathsf{D}_{\mathsf{Direct}}$ to discover interesting rules. The algorithm starts with searching for all association rules, then it goes through $\mathsf{D}_{\mathsf{Direct}}$ again calculating the support and confidence of each discovered rule and eliminating those rules that do not satisfy the given *min supp* and *min conf*. The rest of the discovered rules are considered in our context as belief-based explanations.

Example 5.4. Continuing with our running example, table 5.3 shows the results of applying the CMRules algorithm to $\mathsf{D}_{\mathsf{All}}$ obtained from table 5.1 and table 5.2.

Table 5.3: Mined belief-based explanations

plan	action	belief-based explanations
p1	idle(throttle)	runway(dry) $\wedge$ wind(cross) $\wedge$ efto $\wedge$ flaps $=$ 15 $\wedge$ v1 <speed <v2
p1	deploy(brakes)	decelerate(thrust)
p1	send(tower, msg)	steady(aircraft)
p2	increase(mixture)	runway(wet) $\wedge$ wind(head) $\wedge$ efto $\wedge$ flaps $=$ 15 $\wedge$ v1 <speed <v2
p2	increase(throttle)	escalate(fuel_flow)
p2	take_up(flap)	accelerate(thrust)
p2	pull(yoke)	liftoff(aircraft)
p2	take_up(gear)	flaps $=$ 0
p2	send(tower, msg)	up(gear)

CMRules algorithm can discover all the interesting association rules from $\mathsf{D}_{\mathsf{Direct}}$. Nevertheless, additional post-processing is required, where we rule out any association rule that its consequent label is not a single action name or its antecedent label is not beliefs.

5.5 Validating the Mined Explanations

Our guiding intuition here is that the state update operator and the available data used to generate updated belief-based explanations can also be leveraged to validate the mined explanations. Our validation approach involves some mechanisms that take as inputs (1) A $\mathsf{D}_{\mathsf{All}}$, (2) A state update operator, and compute the soundness and completeness of the mind explanations. We shall keep assuming unique action execution through this section.

First, we need inputs (1) and (2) to generate a sequence (denoted hereafter as $\mathsf{D}_{\mathsf{Updated}}$) that associates each action in $\mathsf{D}_{\mathsf{All}}$ to the set of updated beliefs of all actions executed up to that point in the plan execution. Each object in $\mathsf{D}_{\mathsf{Updated}}$ is a pair of the form $\langle \beta_i, a_i \rangle$, where β is the set of updated beliefs of all actions executed up to a_i. $\mathsf{D}_{\mathsf{Updated}}$ can be simply obtained using $\oplus$ over $\mathsf{D}_{\mathsf{All}}$ assuming the presence of a KB defined in the same language as that in which the beliefs are described. table 5.4 illustrates the results of applying the operator $\oplus$ to input $\mathsf{D}_{\mathsf{All}}$. It should be noted that a single action in $\mathsf{D}_{\mathsf{Updated}}$ can be associated with a set of sets of beliefs, due to the non-determinism nature of the $\oplus$ operator. To validate the mined explanations, it is useful to establish:

- **Soundness**. A sound belief-based explanation is one that is mined correctly (i.e., observed in $\mathsf{D}_{\mathsf{All}}$). Another way to say that a detailed belief-based explanation to a given point in the plan execution should contain the mined belief-based explanation updated using $\oplus$ at that point in the plan execution. Formally, for each plan execution sequence in $\mathsf{D}_{\mathsf{Updated}}$ and each action a_i the following condition must hold: $\beta_i \cup \mathsf{KB} \models b$ for some $b \in b_{a_1}, \ldots, b_{a_i}$, where b_{a_i} is the mined belief-based explanation of action a_i .

- **Completeness**. A complete belief-based explanation requires that all the enabling beliefs of a given action are mined. Actually, this can be viewed as a reversal of the above-cited entailment relation (i.e., $b \cup \mathsf{KB} \models \beta_i$)

Table 5.4: Updated belief-based explanations of plan p1

plan	action	belief-based explanations
p1	idle(throttle)	runway(dry) $\wedge$ wind(cross) $\wedge$ efto $\wedge$ flaps = 15 $\wedge$ v1 <speed <v2
p1	deploy(brakes)	decelerate(thrust) $\wedge$ runway(dry) $\wedge$ wind(cross) $\wedge$ efto $\wedge$ flaps = 15 $\wedge$ v1 <speed <v2
p1	send(tower, msg)	decelerate(thrust) $\wedge$ steady(aircraft) $\wedge$ runway(dry) $\wedge$ wind(cross) $\wedge$ efto flaps = 15 $\wedge$ v1 <speed <v2

Where it is possible for the mined explanations to be unsound or incomplete, further post-processing may be required for more reliable results. We overtake this problem by seeking more observations and/or re-mining with lower support and confidence thresholds.

Example 5.5. Continuing with Example 5.2 and Example 5.4, the scenarios:

- *Why did the pilot agent pull the throttle lever to idle immediately after the EFTO?*,
- *Why did the pilot agent move the mixture knob to rich after the EFTO?*

Can be answered as depicted in Table 5.5 below

Table 5.5: An example of belief-based explanations

Action	Belief-based explanation
idle(throttle)	runway(dry) $\wedge$ wind(cross) $\wedge$ efto $\wedge$ flaps = 15 $\wedge$ v1 <speed <v2
increase(throttle)	runway(wet) $\wedge$ wind(head) $\wedge$ efto $\wedge$ flaps = 15 $\wedge$ v1 <speed <v2

Explaining by UD S_O using the above-cited mechanism is not as straightforward as the account above suggests. There is some post-processing required in terms of choosing between beliefs that achieve a sensible selection strategy. Continuing with Example 5.5, one can customize S_O to select between accelerate-stop and accelerate-go based on wind direction (e.g., cross or head), runway condition (e.g., dry or wet), or based on both context conditions.

5.6 Related Work

Although a large and growing body of work exists on developing explainable agents and robots [78], few works focus on generating explanations for intelligent agents.

Harbers et al. [79] described four algorithms to design explainable BDI agents: one using parent goals, one using top-level goals, one using enabling beliefs, and one using the following action or goal in the execution sequence. They found goal-based explanations were slightly preferable to belief-based expansions to explain procedural actions (i.e., a sequence of actions and sub-goals) based on users' evaluation. Nevertheless, belief-based explanations were preferable in explaining conditional and single actions. Similar explanation algorithms were proposed by Kaptein et al. [80], but to investigate the difference in preference of adults and children for goal-based and belief-based explanations. They found that both adults and children preferred goal-based explanations. Related, but in a different ontology, is the work presented in Kaptein et al. [146], in that the agent explains its action in terms of its beliefs, goals and emotions.

Sindlar et al. [93] proposed an abductive approach to infer the mental states of BDI agents in terms of beliefs and goals. To that end, they described three explanatory strategies under three perceptory presumptions: complete, late, and partial observations. Sindlar et al. extend their work to an explanation approach that takes into account three organizational principles: roles, norms, and scenes in [94]. The extended work proposes an approach to how the observed behaviour of game players can be explained and predicted in terms of the mental state of virtual characters. Related, but in a different domain, is the chapter presented in [147]. Sequeira and Gervasio propose a framework for explainable reinforcement learning that extracts relevant aspects of the RL agent interaction with its environment (i.e., interestingness elements) in [147]. They suggested four dimensions of analysis: frequency, execution certainty, transition-value, and sequence analysis to extract the interestingness elements, which are used to highlight the behaviour of the agent in terms of short video clips.

All the previous approaches presented in this chapter can theoretically generate explanations of agents' actions, but assuming reliable observations, availabil-

ity of an explanation generation module and deterministic execution of plans. On the other hand, we leverage the past execution experiences of the target agent, which allows performing various techniques to handle unreliable observations (e.g., measures of interestingness). Much of the work done on agent explanation generation shares the common judgment that relatively short explanations are more useful to explainees. Nevertheless, as shown in our running example, detailed explanations are critical in some cases (e.g., explaining BDI plan selection) as shown in our running example. Finally, and in contrast with the literature, our mechanism allows explanations to be non-deterministic.

5.7 Chapter Summary

In this chapter, we addressed the problem of agent explanation mining (and specifically belief-based explanations) in the context of the well-known BDI paradigm. This problem was formulated as follows: "Given the past execution experiences of an agent and an update operator, generate the belief-based explanation of each action referred to in the agent's past execution experiences". We presented an update operator that is able to generate detailed explanations. Through examples, we also showed that detailed explanations could be useful in explaining BDI plan selection. We have tackled the problem of explanation generation in non-deterministic settings. At this point, we are trying to extend the application of the proposed approach to other BDI handles. Experimental results and technical details are discussed in chapter 6.

Chapter 6

The XPlaM Toolkit

This chapter presents XPlaM (eXplainable Plan Miner) Toolkit guide, developed to support programmers in the design of Digital Twins of organizations without loss of generality to other anthropomorphic systems. XPlaM manual describes how the algorithms and the conceptual components presented in chapters 3, 4 and 5 have been implemented. Fundamentally, XPlaM builds on the following two premises: (1) Digital Twins should be re-crafted in response to every change in the target system or the operating context [16] and (2) Handcrafting Digital Twins, which is the current norm, comes with several obvious limitations (e.g., time and effort). Re-crafting a Digital Twin is the process of understanding and consequently imitating an anthropomorphic system through, in our case, abductive reasoning with little knowledge into its historical behaviour. The setting in which XPlaM has developed is very challenging, where we look at the automatic acquisition of Digital Twins. To that end, XPlaM leverages:

- Abductive reasoning to infer explanations from the historical behaviour of the target system,
- BDI ontology (e.g., beliefs and desires) as the implementation vehicle for explanations generation, and
- AgentSpeak(L) programming language as a means of representing the generated explanations.

XPlaM provides a number of explanation techniques that comply with the operational semantics of Jason [75]. For now, XPlaM provides five techniques in the

form of plugins, three of which are related to the explanation techniques offered in the previous three chapters and two pre- and post-processing plugins that will be discussed later in this chapter.

Technical Details

XPlaM is implemented on top of Visual Studio 2019 using C# programming language. Similar to the idea of the well-known process mining workbench ProM [101], XPlaM has been developed as an extensible framework to support different goal-driven XAI techniques (each of which comes in the form of a plugin action). XPlaM users can navigate through three main user interfaces: (1) **Workspace** view, which facilitates managing imported resources, (2) **Action** view, which shows all the plugins that can be taken over the imported resources, and (3) **View** view which shows an overview of a selected resource or the results of applying a plugin over that resource. Figure 6.1 represents the user interface of the XPlaM toolkit.

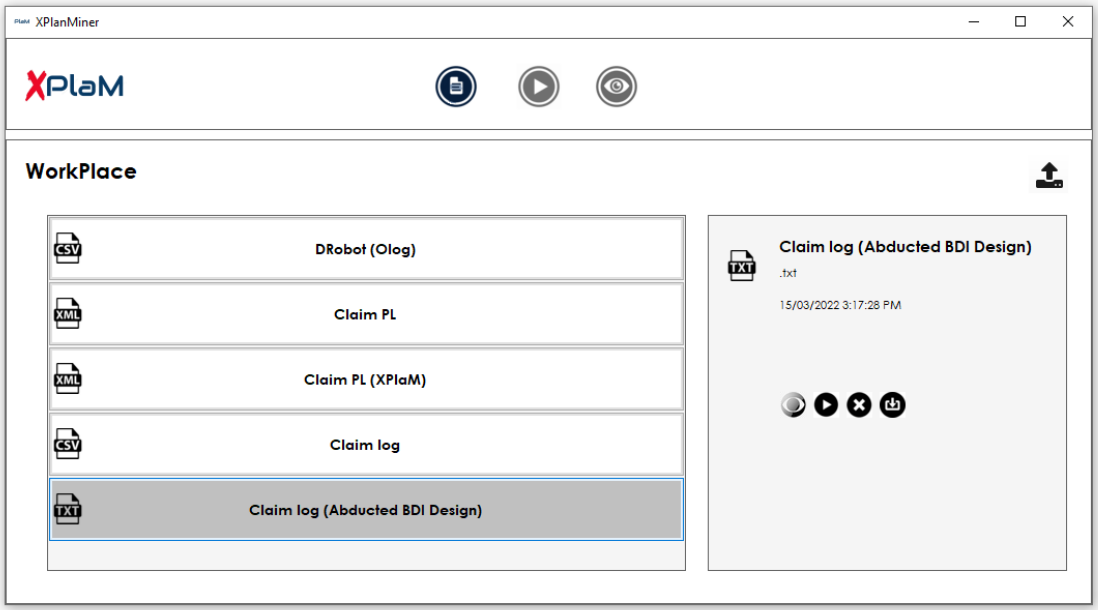


Figure 6.1: A snapshot of XPlaM main interface

XPlaM leverages Jason implementation[a] of the alphabet of AgentSpeak(L) language. Note that, for the practical results of this work, XPlaM recognises only the most common operational semantics of Jason. XPlaM also leverages the NanoByte

[a]https://github.com/jason-lang/jason

SAT Solver[b] to implement the state update operator described in chapters 4 and 5. The developed operator supports both propositional variables (i.e., variables that can be true or false) and non-Boolean variables (i.e., new value assignments). For testing, experiments have been implemented on an Intel Core i5-6200 CPU (with 8GB RAM). Note that each XPlaM plugin requires different inputs (data files) to operate on. XPlaM can deal with three data extensions: AgentSpeak(L) plan libraries with a .xml and a .txt extensions, logs with a .csv extension, and knowledge bases written in Conjunctive Normal Form (CNF) with a .txt extension. The source code for XPlaM has been published online at https://github.com/DSL-UOW/XPlaM.

What questions is XPlaM used for?

We expect XPlaM toolkit to help answer a number of questions that users may have about designing Digital Twins of organizations. To illustrate this, consider again the ACo example from chapter 3. Table 6.1 outlines some of the questions that the startup may have about ACo.

Table 6.1: What questions is XPlaM used for?

Question	Section
What are ACo's top-level plans based on its observed actions?	6.2
How ACo prioritizes its goals?	6.2
How does ACo deliberate between competing options?	6.2
How to leverage existing hypotheses to explain unknown plans?	6.3
What must have been known for ACo to perform a particular action?	6.4

The following sections show how to use XPlaM to answer such questions.

Converting Plugins

To use XPlaM, the user is required to upload AgentSpeak(L) plans in the form of .xml format. The choice of .xml format as a hierarchical structure of AgentSpeak(L) plans enables our parsing mechanism to resolve a plan into its component based on the operational semantics of Jason [75] without the need for a fully-fledged

[b]https://github.com/nano-byte/sat-solver

interpreter. Furthermore, .xml structural representation allows more readability for developers and other future works. To do so, XPlaM provides the plugin Convert AgentSpeak(L) to xml. This plugin complies with the operational semantics of Jason [75]. To illustrate the functionality of this plugin, consider the following example.

Example 6.1. Consider the product release plan from chapter 3.

```
@p1 +release(x): beta_test(x)
                 <- !promote(x);
                    launch(x);
                    create_feedback_channel(x).
```

Applying Convert AgentSpeak(L) to xml plugin over the above AgentSpeak(L) plan generates the following .xml file

```
<Plans>
  <Plan lable="p1">
    <Triggering>+release(x)</Triggering>
    <Context>beta_test(x)</Context>
    <PlanBody>
      <BodyLiteral Type="achieve">
        <Name>!promote</Name>
        <Parameters>x</Parameters>
      </BodyLiteral>
      <BodyLiteral Type="action">
        <Name>launch</Name>
        <Parameters>x</Parameters>
      </BodyLiteral>
      <BodyLiteral Type="action">
        <Name>create_feedback_channel</Name>
        <Parameters>x</Parameters>
      </BodyLiteral>
    </PlanBody>
  </Plan>
</Plans>
```

In addition, XPlaM supports converting .xml to AgentSpeak(L) with a .txt extension. To do so, the user needs to select the plugin Convert AgentSpeak(L) to text.

chapter organization

Table 6.2 illustrates how the remainder of this chapter is organized.

Table 6.2: XPlaM main plugins and corresponding chapters

Plugin	Section	Corresponding chapter
Abducive Design of BDI Agents	6.1	chapter 3
Abductive Plan Editing	6.2	chapter 4
Explaining by Belief	6.3	chapter 5

This section uses the startup competitor analysis example (described in chapter 3) as a running example.

6.1 Plugin I: Abductive Design of BDI Agents

The XPlaM Abductive Design of BDI Agents plugin provides the ability to understand, through abductive reasoning, how an anthropomorphic system (i.e., an organization in this case) carries out its tasks with very little insight into exactly how it does so. As the name suggests, Abductive Design of BDI Agents plugin does not provide mere explanations of target system behaviour but also generates an executable BDI agent model on the basis of the generated explanations. Executing the agent model can then imitate the behaviour of the target system in terms of plans, beliefs, goals generation and deliberation. This, in turn, allows performing various types of analysis of the internal workings of the system being twinned. It is important to note that we do not claim to be capable of inferring a complete and correct BDI agent-based Digital Twin but providing a description of organization behaviour. Developers can then use this description to develop a more complete and correct Digital Twin. As depicted in Table 6.3, Abductive Design of BDI Agents requires two inputs: (1) A collection of AgentSpeak(L) plans in a .xml extension, and (2) A sequence of actions in a .csv extension.

Table 6.3: Abductive Design of BDI Agents inputs

Data file	Description	Extension
Collection of plans	A set of potential hypotheses written using AgentSpeak(L) programming language.	.xml
A behaviour log	Action sequence that we want to explain.	.csv

Note that AgentSpeak(L) plans in .txt format can be converted to .xml (and vice versa) using XPlaM converting plugins. Nevertheless, it is important to note that the user is still required to map between the vocabulary used in input (1) and the vocabulary used in the behaviour logs used. This is not an impractical issue because, in many settings, developers must first agree on the grammar and the syntax used for parsing to avoid linguistic controversy.

Running Abductive Design of BDI Agents plugin over inputs (1) and (2) generates three outputs, as shown in Table 6.4.

Table 6.4: Abductive Design of BDI Agents outputs

Data file	Description	Extension
Plan Library	A set of plans and sub-plans whose execution generates precisely the observed actions.	.asl
Model of Abduction	Mapping from available plans to subsets of the observed actions.	.txt
Customized Agent Class	Three overrided methods describing event, option and selection functions.	.java

As such, the execution of the plans and sub-plans thus identified (in .asl format) generates precisely input (2) given the Agent Class thus customized (in .java format). For more explainability, the plugin also displays the model of abduction - a mapping between hypotheses (i.e., plans) and observations (i.e., observed actions). To create a BDI agent-based Digital Twin out of these outputs, the user needs to copy the plan library and Agent Class into his Jason project. For readers interested in Jason project development, we refer to [75].

The remainder of this Section is organized as follows. Section 6.1.1 illustrates the implementation of abductive plan recognition. Section 6.1.2 describes our implementation of the abductive design of BDI selection functions. Section 6.1.3 reports on the experimental evaluation of the plugin.

6.1.1 Abductive Plan Recognition

The starting point for the Abductive Design of BDI Agents plugin is to infer the top-level plans that best explain an observed sequence of actions. We view this prob-

lem as a hypothesis assembly task [148]. However, what our mechanism tries to infer is slightly different from the notion of classical abductive plan recognition, where it seeks to find a set of plans whose execution generates precisely the observed action sequence.

First of all, XPlaM traverses the uploaded plans checking for improper use of Jason's operational semantics. If there is a syntax error preventing XPlaM from interpreting the uploaded plans, a pop-up window will appear informing the user that XPlaM cannot parse the .xml file successfully.

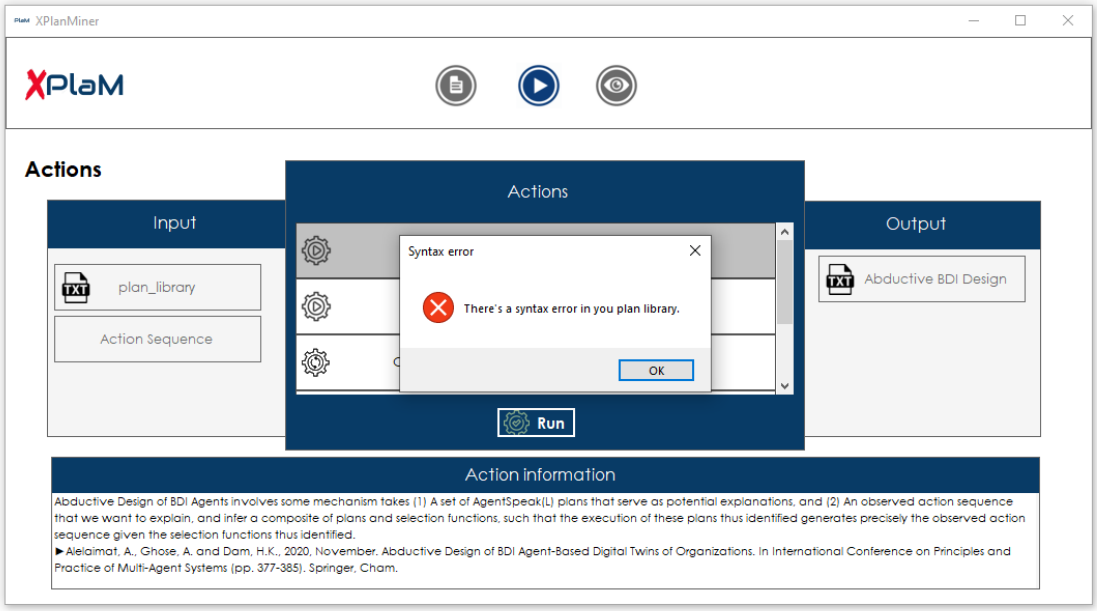


Figure 6.2: XPlaM: syntax error pop-up window

Recall that XPlaM does not provide a fully-fledged interpreter. Hence, it cannot detect run-time errors (e.g., the user did not provide the required plan for a sub-goal) nor logical errors (e.g., incorrect definition of context conditions). If the uploaded plans are syntax error-free, the XPlaM starts the abductive plan recognition algorithm (described in chapter 3). Our implementation of abductive plan recognition involves two steps: (1) search for a potential explanation, then (2) repeatedly search for a better explanation.

As defined in chapter 3, a behaviour log is a sequence of observations, each of which consists of an action name and a list of typed parameters. To avoid handling different groundings of the variables as distinct observations, we neglect the precise grounding of valuables. For each observation in the behaviour log, XPlaM

traverses each uploaded plan searching for matching. A matching, therefore, does not include the list of typed parameters. Once matching is found, XPlaM takes the current plan as a potential hypothesis and generates a list of lists to capture all possible instances of that plan. A plan instance is one way in which a plan can be executed, each of which is represented as a sequence of primitive actions. Continuing with the product release plan, the generated plan instances are:

instance 1: < advertise(x, email_list), launch(x), create_feedback_channel(x)>
instance 2: < advertise(x, socialMedia), launch(x), create_feedback_channel(x)>

Once the recognition is completed, XPlaM starts sorting potential hypotheses based on their plausibility (i.e., degree of fitness). The definition of the best explanation for a given sub-set of observations, therefore, requires mapping hypotheses in the set of potential explanations to a partially ordered set of integers. We shall refer to this mapping as plausibility. For example, if plan p1 is the most plausible hypothesis, p1 explains a sub-set of D_{All}, then p1 can be stated as the best explanation without the need for further search. Plausibility is the ratio of observations, the plan can explain the total number of the plan steps. For example, if plausibility(p1) > plausibility(p2) for a given sub-set of observations, then p1 is a better explanation than p2 for that sub-set.

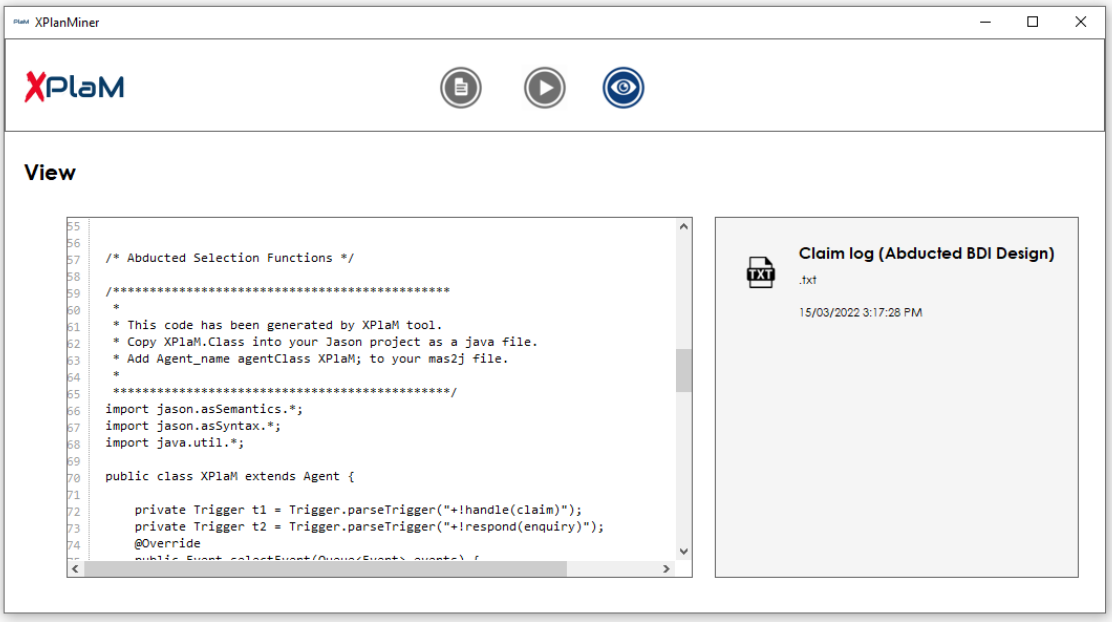


Figure 6.3: A snapshot of Abductive Design of BDI Agents outputs

Only the best explanations will be displayed as part of the plugin output. Plans that can partially explain a subset of the observed actions can also be recognized, but using Abductive Plan Editing plugin, as described in Section 6.2 in this chapter.

6.1.2 Abductive Design of BDI Selection Functions

For a BDI agent to faithfully imitate a target system, it must also re-create the internal workings of that system. Viewing the internal workings of the target system through the lens of BDI agents can be programmed by answering the following questions: How does the system prioritize its goals or changes in the environment?, How does the system select its know-how? And How does the system execute its competing attention? XPlaM suggests three common selection strategies to answer these questions: FIFO (First-In-First-Out), RR (Round-Robin) and UD (User Defined). XPlaM takes these strategies as hypotheses to explain the internal workings of the target system based on some observations, as detailed in Table 6.5.

Table 6.5: Main selection strategies for customizing BDI agents

Hypothesis	Observation
FIFO S_E	Pending events are handled based on their temporal order.
UD S_E	Pending events have different priorities.
FIFO S_O	There is a strict preference for one option.
UD S_O	Applicable options have different priorities.
FIFO S_I	Executing each intention to completion.
RR S_I	Executing a fixed number of steps of each intention in turn.
UD S_I	Competing intentions own different priorities.

The guiding intuition behind the abductive design of BDI selection functions can be summarized as follows: selection strategies that explain the behaviour of the target system can also be used to customize the internal workings of its BDI-based Digital Twin. As described in chapter 3, XPlaM considers S_E, S_O and S_I, which are methods of Agent Class. What XPlaM tries to achieve here is to extend the basic functionality of BDI selection functions to re-create the internal workings of the target system.

Abductive Design of S_E

The starting point to the abductive design of AgentClass is to abductively customise the way in which the BDI agent will, but not necessarily, prioritise events. Typically, this can be implemented in a FIFO (first-in-first-out) style, i.e., select the first event in the list of pending events. However, there are some exceptions where the target system has an obvious preference for a specific event. Our implementation focuses on settings with FIFO S_E due to the competitive settings of this work, but it offers guidance to developers to manually modify S_E to satisfy preference-based event selection. XPlaM starts recording all the events that trigger the execution of the inferred plans, e.g., beliefs and achievement goals, in a queue called Queue < Event >. The queue Queue<Event> orders the set of all recorded events in a FIFO style. Accordingly, the method selectEvent selects the first events in Queue<Event>. Once the queue is filled up, XPlaM generates the following .java code.

```
/* Abductive Design of AgentSpeak(L) Selection Functions */
/*********************************************
 *
 * This code has been generated by XPlaM toolkit.
 * Copy XPlaM.Class into your Jason project as a java file .
 * Add Agent_name agentClass XPlaM; to your mas2j file.
 *
 *********************************************/
import jason.asSemantics.*;
import jason.asSyntax.*;
import java.util.*;

public class XPlaM extends Agent {
   @Override
   public Event selectEvent(Queue<Event> events) {
         private Trigger e4 = Trigger.parseTrigger("+release(x)");
         private Trigger e7 = Trigger.parseTrigger("+active(x)");
         private Trigger e9 = Trigger.parseTrigger("+mature(x)");
         private Trigger e15 = Trigger.parseTrigger("+limited(x)");
         private Trigger e18 = Trigger.parseTrigger("+obsolete(x)");
         Iterator<Event> i = events.iterator();
```

```
        while (i.hasNext()) {
         Event e = i.next();
         i.remove();
         return e;}
         }
      return super.selectEvent(events);
  }
```

At run-time, the method selectEvent selects the first event in the list of pending events.

Abductive S_O

The second point to the abductive design of Agent Class is to abductively customise the way in which the BDI agent will select applicable plans. The default implementation of selectOption method in the Jason platform selects the first applicable plan from the set of applicable plans O_e according to the order in which they are written. This implementation is used if the set of applicable plans is not a singleton for the current unification and belief state. We shall refer to this implementation of selectOption as FIFO S_O.

The abductive customization of selectOption can be summarized as follows. For each possible unification and belief state, XPlaM determines O_e. If O_e is a singleton, it does nothing. Otherwise, it runs through O_e again, searching for irrational selections. XPlaM exploits the default implementation of selectOption by setting the order in which the set of applicable plans should be written in the recognized plan library, as depicted below.

```
/**********************************************
*
* Change the order in which plans are written in your AgentSpeak(L) plan
* library code as follows:
* @p8, then @p7
*
**********************************************/
```

Note that Abductive S_O can deal only with the rational selection of plans (i.e.,

the observation log shows only strict preferences for one option for each O_e). Irrational selection strategies (e.g., S_O = p7 at some instances and S_O = p8 at other instances) can also be explained, but using Explaining by Beliefs plugin, as described in Section 6.3 in this chapter.

Abductive S_I

The third point to the abductive design of Agent Class is to abductively customise the method selectIntention - the way in which the BDI agent will select intentions to be further executed in each reasoning cycle. Recall that there are two implementations of selectIntention: RR S_I and FIFO S_I.

XPlaM starts constructing intention stacks from the sequence of inferred plans H′. The procedure below shows how to construct intention stacks, as described in [75].

```
Construct_intention_stacks(H′)
While(E ≠ ∅):
    S_E(E) = e
    Re = unify_event(H′)
    Oe = check_context(R_e)
    if(external_event(e)): I = I ∪ [S_O(O_e)]
    if(internal_event(e)): i.push(S_O(O_e))
Endwhile
```

Given a sequence of inferred plans H′, we can recognize a new intention stack i if the parsed plan is triggered by an external event, or else it is pushed on the top of the current stack. Note that each intention i refers to an execution instance.

```
@Override
  public Intention selectIntention (Queue<Intention> intentions) {
       Iterator <Intention> i = intentions. iterator ();
       while (i.hasNext()) {
          Intention cit = i.next();
          boolean i1 = cit.hasTrigger(e4, new Unifier());
          boolean i2 = cit.hasTrigger(e7, new Unifier());
          boolean i3 = cit.hasTrigger(e9, new Unifier());
          boolean i4 = cit.hasTrigger(e15, new Unifier());
          boolean i5 = cit.hasTrigger(e18, new Unifier());
          if ( intentions.size() == 1) {
              if(i1 || i2 || i5){return cit;}
```

```
                                        }
else  if (i3  && i4) {
    i .remove();
    return  cit ;}   }
                }
```

Consider the two intentions $I = \{i_1, i_2\}$. FIFO S_I executes either i_1 or i_2 to completion before starting the other intention based on their temporal order. This can be extracted by checking for non-interleaving execution of their plans. RR S_I executes a fixed number of steps of i_1 and i_2 in turn. This can be extracted by checking for interleaving execution of their plans. After identifying which intentions can be interleaved and which cannot, XPlaM generates an override selectIntention as depicted above. Typically, the overridden method generates one possible flow of control (i.e., IF statement) for FIFO intentions combined with the logical operator OR and n number of possible flow of controls for each pair of RR intentions combined with the logical operator AND. This, in turn, prevents FIFO intentions from being interleaved with RR intentions at run-time and vice versa.

User-Defined Customization

Now, we consider the problem of what needs to be done when the above-cited selection strategies are found to be irrational or unable to explain a particular selection. This section offers guidance to developers to modify the first-cut customized selection functions by using simple tests as a preference-based learning problem. Our discussion focuses on settings where there is a strong preference for one selection. Preference-based UD customization of Agent Class involves some mechanism that takes the sequence of recognized plans H', the set of potential explanations H, i.e., $H' \subseteq H$, and generates preference relations between potential and inferred selections. The basic intuition behind this is that, for example, if a triggering event is always followed by another event, then this suggests that there is a preference relation between both events. To that end, we introduce the following notations and definitions.

Definition 6.1 (Regular weak and strict preferences). Let F be a totally ordered set of elements $\{a, b, c, \ldots, z\}$. A regular weak preference, denoted by '$\succeq$', is a bi-

nary relation on F. For the ordered pairs (a, b), we use the notation $a \succeq b$ to point out that "a is preferred or indifferent to b". Relation '$\succeq$' is transitive; whenever $a \succeq b$ and $b \succeq c$, then also $a \succeq c$. Relation '$\succeq$' is reflexive. Formally, $\forall a \in F : a \succeq a$. A regular strict preference, denoted by '$\succ$', is another binary relation on F. For the ordered pairs (a, b), we use the notation $a \succ b$ to indicate that "a is strictly preferred to b". Relation $\succ$ is asymmetric. Notationally, $a \succ b \Rightarrow \neg b \succ a$. Also, it is transitive [149].

Preference-based UD S_E

Let $b1, b2$ be two external events such that b1 triggers the execution of plan p1 and b2 triggers p2. Assume also that the creation time of p1 is t_j and p2 is t_k.

- $b1 \succeq b2$ *iff* there is an H' where p1 is always executed before p2 (i.e., $j < k$). Relation $\succeq$ describes regular weak preference relation on E. For example, the statement $b1 \succeq b2$ means that the external event b1 is preferred or indifferent to b2.
- $b1 \succ b2$ *iff* $b1 \succeq b2 \wedge \neg b2 \succeq b1$. Relation $\succ$ suggests a regular strict preference. For instance, the statement $b1 \succ b2$ indicates that "b1 is strictly preferred to b2", and can be computed from $\succeq$ relation.
- $S_E(E) = \{b1 \mid b1 \succ E\}$. Given the relation '$\succ$' on E we say that $S_E(E) = b1$ if for every other $b \in E$ we have $b1 \succ b$.

Regular strict preferences, in this context, describe how the target system prioritized external events based on their appearance in H'. Assume that "+release(x)" has a priority over other external events. This can be implemented by adding the following code to selectEvent method.

```
if (e4) {
    i.remove();
    return e;}
```

UD S_O

Let $p1, p2 \in O_e$ be two applicable plans for the external event b1. When both plans appear in the same observation log, two possible explanations might be adopted:

(1) Assuming that the target system has made an irrational decision by selecting p1 or p2, or (2) seeking more beliefs (as described in chapter 4) to explain each selection and try again to override the method selectOption accordingly. Note that preference-based UD S_O can be problematic since some of the target behaviour might be impossible to be re-crafted. Our focus, therefore, is on the latter explanation. We shall address this issue later in this chapter (section 6.3).

Preference-based UD S_I

Let $i_j, i_k \in I$ be two intention stacks.

- $i_j \succeq i_k$ *iff* there is an I such that $i_j, i_k \in I$ and $j < k$. Relation $\succeq$ describes regular weak preference relation on I. For example, the statement $i_j \succeq i_k$ means that "i_j is preferred or indifferent to i_k".
- $i_j \succ i_k$ *iff* $i_j \succeq i_k \wedge \neg i_k \succeq i_j$. Relation $\succ$ suggests a regular strict preference. For instance, the statement $i_j \succ i_k$ indicates that "i_j is strictly preferred to i_k", and it can be computed from $\succeq$ relation.
- $S_I(I) = \{i \mid i_j \succ I\}$. Given the relation '$\succ$' on I, we say that $S_I(I) = i_j$ if for every other intention $i \in I$ we have $i_j \succ i$.

Regular strict preferences here describe how the target system prioritizes different focuses of attention based on their appearance in H'.

6.1.3 Experimental Results

Aiming to establish that our proposed abductive design of BDI agents generates reasonable, reliable results, we ran an experiment with one of Jason's examples called the Domestic Robot[c] and another one with an amended IT incident management logs[d]. For the Domestic Robot example, data has been collected using a debugging tool called Mind Inspector [75]. The collected data contains 100 execution instances, with multiple event, option, and intention selections.

[c] http://jason.sourceforge.net/wp/examples
[d] https://www.kaggle.com/datasets/asjad99/it-incident-management-process

Performance Results of the Abductive Design

Our evaluation relies on the notion that an agent equipped with an abductive design should generate precisely the observed behaviour. Accuracy measure is, therefore, used to evaluate our abductive design. Accuracy is the ratio of correctly imitated actions over the total number of imitated actions. Figure 6.4 illustrates the accuracy results of the abductive design approach described in this thesis compared to other workflow-based approaches represented in [106], [107], [108], and [109]. Most of these approaches share a common inferring technique, i.e., transforming inferred models (e.g., workflow net) into well-defined plans.

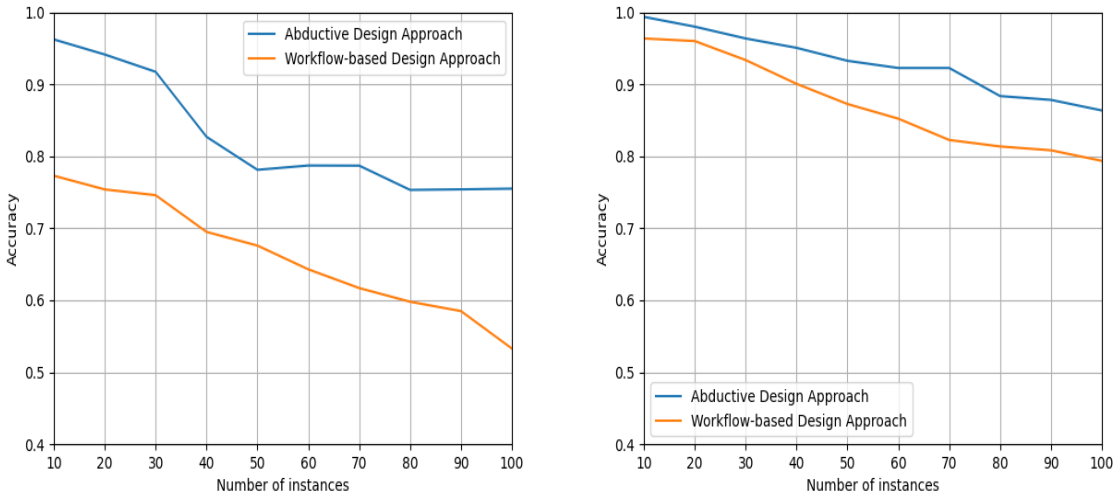


Figure 6.4: Accuracy results for IT incident management (a), and Domestic Robot (b) examples for different numbers of instances.

As illustrated in Figure 6.4, the accuracy results demonstrate that our abductive approach can be an adequate solution to recreate the observed behaviour in both examples. However, the reader should keep in mind two critical details. The more complex the target system's inner workings are, the lower accuracy we can obtain. This is not surprising because, in real-life settings, organizations are driven by a wide range of values that shape their inner workings. Second, we have considered artificially adding noise/irrational selections to the entries - as expected; the accuracy goes down as the instances number increases. With respect to these two details, the abductive designs performed a higher accuracy compared to other workflow-based design approaches. Note that we intentionally modified

the selection functions to implement different selection strategies of the queued elements in the IT incident management example. This explains the low accuracy values obtained using workflow-based design approaches in Figure 6.4 (a). For the second experiment, as illustrated in Figure 6.4 (b), our abductive design performed only slightly better than the workflow-based design approaches because what the Domestic Robot does is select the first event, plan and intention in every reasoning cycle (FIFO).

Complexity of the Abductive Design

In this subsection, we ran a series of experiments to find out how fast XPlaM would take to explain an organization behaviour. For any given behaviour, we noticed that four parameters can highly affect the computation time of the explanation techniques: (1) size of H, (2) branching factor, (3) depth of search, and (4) number of selections (S_E, S_O, S_I) the physical space made during data recording. We tested the time complexity of the explanation techniques by fixing and varying the above parameters as follows.

1. **Experiment 1. (Size of H).** We calculated the computing time in milliseconds needed for the abductive design by fixing the branching factor, depth of search, and the number of selections and varying the number of potential explanations (i.e., size of H). We set the branching factor = 1, depth of search = 1, and the number of selections = 8.

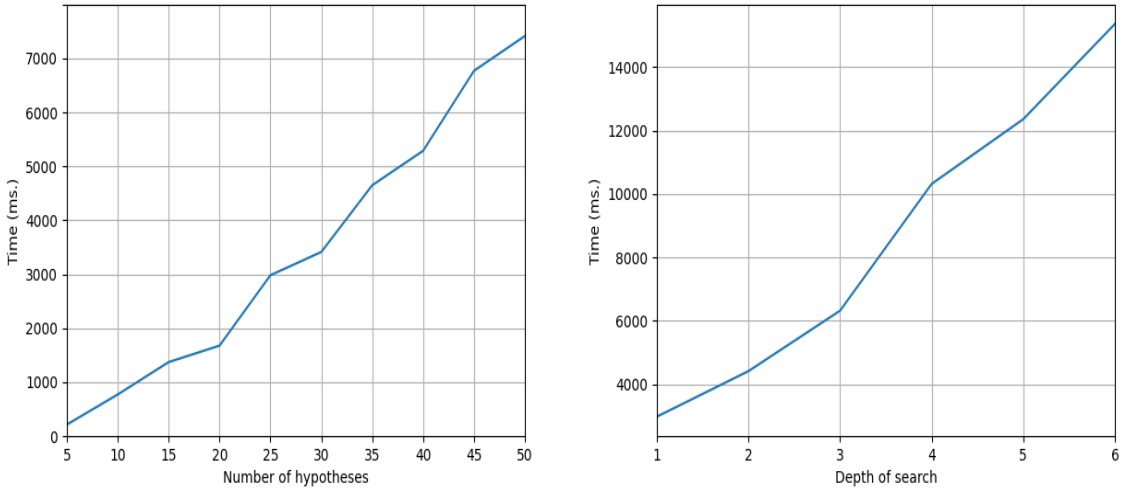
2. **Experiment 2. (Branching factor).** We calculated the computing time in milliseconds needed for the abductive design by fixing the size of H, depth of search, and the number of selections and varying the number of OR nodes presented in each plan. We set the size of H = 10, depth of search = 1, and the number of selections = 8.

3. **Experiment 3. (Depth of search).** We calculated the computing time in milliseconds needed for the abductive design by fixing the size of H, the branching factor, and the number of selections and varying the number of levels of the goal-plan tree up to which the abductive plan recognition algorithm will be applied. We set size of H = 10, branching factor = 1, and

number of selections = 8.

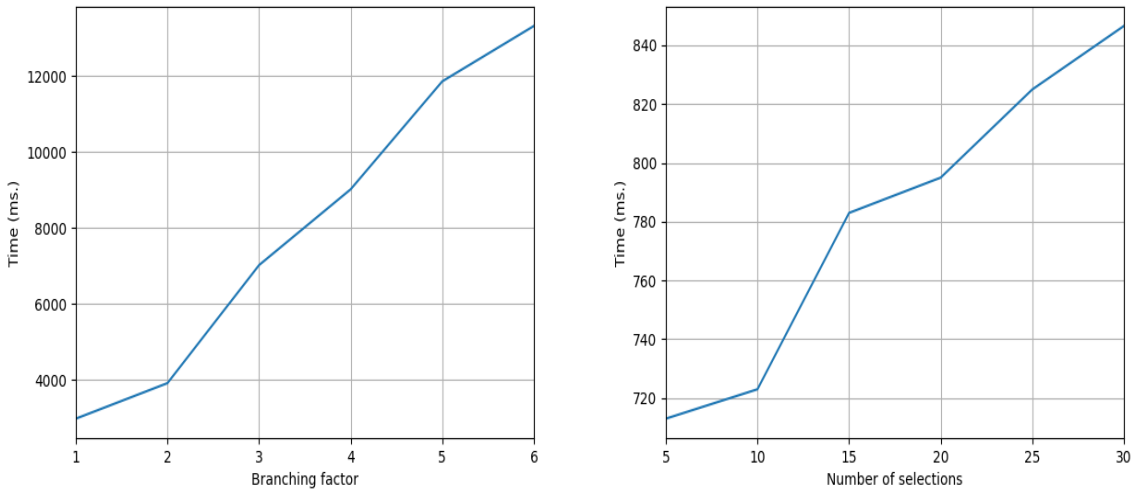
4. **Experiment 4. (Number of selections).** For the purposes of this experiment, we focused on the required time to explain by selection strategies. We calculated the computing time in milliseconds needed for the abductive design by fixing the size of H, depth of search, and branching factor and varying the number of selections. We set size of H = 10, depth of search = 1, and branching factor = 1.

Figure 6.5 depicts the time complexity for each of the above-detailed experiments.



(a) (b)

As illustrated in Figure 6.5 (a), the computing time is plotted linearly in respect of the number of available hypotheses. We view this as an unfortunate necessity. Recall that the abductive design of Digital Twins is about finding a correct and complete explanation of organization behaviour. It is necessary, therefore, to include all available hypotheses. A higher growth rate can be seen with the increase in search depth, as shown in Figure 6.5 (b). A similar trend is noted with respect to the number of OR nodes presented in the hypothesis body part, as shown in Figure 6.5 (c). As illustrated in Figure 6.5 (d), the number of selections the organization made during data recording can slightly affect the computation time of the abductive design approach. It is to be noted here that for each run, the abductive plan recognition algorithm generates a different set of explanations (different



(c) (d)

Figure 6.5: Computing time in milliseconds needed for recognition of a different number of hypotheses, depths of search, branching factors, and the number of selections.

H′). Thus, explaining by selection strategies generates new explanations. Nevertheless, compared to plan recognition, inferring the internal behaviour of the organization in terms of event, option, and intention selection functions is relatively tiny.

6.2 Plugin II: Abductive Plan Editing

The XPlaM Abductive Plan Editing plugin enables users to explain unknown plans (i.e., ones that involve new classes of observations) by modifying existing hypotheses. As detailed in chapter 4, Abductive Plan Editing plugin does not involve a plan recognition mechanism, but it works as a complementary for a variety of single-agent plan library-based plan recognition techniques. Hence, this plugin is not suitable for stand-alone use, but it requires a plan recognition mechanism. Table 6.6 illustrates the required inputs for this plugin action.

Table 6.6: Abductive Plan Editing inputs

Data	Description	Extension
Plan	A plan that explains partially a sequence of observation.	.xml
Unknown plan	An action sequence that we want to explain.	.csv
Capability library	A base comprises the set of all capabilities specifications.	.txt
Goal state	A specification of the final effect associated with the end state of the plan execution	.txt

To illustrate the implementation of Abductive Plan Editing plugin, we consider variants of Monroe County Corpus for emergency response domain [19] as a running example through this section.

Example 6.2 As shown below, the agent aims to provide medical attention to patients. It just receives requests from medical personnel, drives to the patient's location (loc), loads the patient (pt) into the ambulance (amb), drives back to the hospital (h), and takes the patient out of the ambulance. Moreover, the agent is also equipped with capabilities related to the emergency response domain problem.

```
@h1 +provide_medical_attention(pt)
    :   available (amb)
    <- !at(amb,pt);
       get_in(pt,amb);
       !at(amb,h);
       get_out(pt,amb).
@h2 +!at(amb,loc) :  not  at(amb,loc)   <- drive_to(loc).
@h3 +!at(ambulance,loc) : at(amb,loc)   <- true.
```

Consider the following scenarios that may arise in this emergency response domain:

i After arriving at the scene, the agent performed CardioPulmonary Resuscitation (CPR) on the patient and loaded the patient into the ambulance.

ii The agent called an air ambulance and drove back to the hospital without loading the patient into the ambulance due to rough terrain, poor weather, etc.

iii The agent drove back to the hospital without loading the patient into the ambulance as the patient went missing.

It is not difficult to recognize that the basic reactive behaviour of the BDI agent system (described in chapter 2) cannot produce the behaviours depicted in these scenarios on its own; since it does not have those plans in its plan library. Arguably, there is at least an extension to the system that enabled such edits in the agent behaviour. For example, the behaviour illustrated in scenario (i) involves an insertion edit (where the agent added CPR execution into the plan). Scenario (ii) involves a substitution edit (where the agent replaced loading the patient into the ambulance with calling an air ambulance). Finally, scenario (iii) represents a deletion edit (where the agent dropped loading and taking the patient into and out of the ambulance from the plan).

First, Plan Editing main method splits the first two inputs into sequences (only body literal of type Action are converted). Continuing with Example 6.2 scenario (ii), the following shows an example of an imperfect explanation (line 1) and an unknown plan (line 2) that holds insertion and deletion edits.

```
[1] [drive_to(pt),get_in(pt,amb),drive_to(h),get_out(pt,amb)]
[2] [drive_to(pt), call(air_am),drive_to(h)]
```

As a second input, the user needs to upload a .txt file that comprises the set of all available capabilities specifications related to the domain problem, which should be represented as follows.

```
[1]{not at(abm,loc)} drive_to(loc) {at(abm,loc)}
[2]{at(abm,loc)} get_in(pt,amb) {in(pt,abm)}
[3]{in(pt,abm),at(amb,h)} get_out(pt,amb) {out(pt,abm),at(pt,h)}
[4]{not reachable(pt)} call(air_amb) {at(pt,h)}
[5]{not breathing(pt),has(pt,pulse)} cpr(pt) {breathing(pt)}
```

Also, the user needs a .txt containing the specifications of the final effects associated with the end state of plan execution (the imperfect explanation).

```
[1]{at(abm,h),at(pt,h)}
```

Fundamentally, there are three methods that implement Abductive Plan Editing: Levenshtein Distance, Plan Editing and Plan Validating. Levenshtein Distance method

takes the first two inputs to measure the difference between them. Unlike the customary implementation of distance algorithms, Levenshtein Distance method computes and represents the edit distance (integer data type) and a list of required modifications to turn an imperfect explanation into the unknown plan (list data type). Table 6.7 shows an example of Levenshtein distance calculation between the above-cited plans.

Table 6.7: An example of Levenshtein Distance calculation between two plans

		drive_to(pt)	call(air_amb)	drive_to(h)
	0	1	2	3
drive_to(pt)	1	0	1	2
get_in(pt,amb)	2	1	1	2
drive_to(h)	3	2	2	1
get_out(pt,amb)	4	3	3	2

For the corresponding modifications to be computed, we need to trace back the choices that led to the minimum edit cost (bottom-right cell in the above table). This trace back in Table 6.7 is marked in yellow.

[1] replace call (air_amb) at step 2
[2] delete get_out(pt,amb) at step 4

Plan Validating method implements the state update operator (described in chapter 4) to check the validity of the modifications suggested by the above methods. After computing the state specifications of $s_1, \ldots s_n$, where s_1 is the initial state and s_n is the final state scenario associated with the execution of the modified plan and for each action a_i in the modified plan body, Plan Validating method checks the conditions illustrated in Table 6.8.

Table 6.8: Conditions for plan validity

Condition	Method	Returns
pre-conditions of a_i are exist in the preceding state s_{i-1}.	Contains(T)	Boolean
pre-conditions of a_i are consistent with s_{i-1}.	SatSolver	Boolean
goal specifications exist in the final effect scenario of the modified plan.	Contains(T)	Boolean
goal specifications are consistent with s_n.	SatSolver	Boolean

An edit is valid if all the four conditions in Table 6.8 are true. Note that each

s_i is created as a list of list data type, since each state can be non-deterministic in general.

```
[1] replace  call(air_amb) at step 2  valid  edit
[2] delete  get_out(pt,amb) at step 4  valid  edit
```

The code below shows the results of executing Abductive Plan Editing plugin using the inputs in Example 6.2.

```
Number of edits = 2
replace  call(air_amb) at step 2  valid  edit
delete  get_out(pt,amb) at step 4  valid  edit

======== Edited Plan ====================
@p1 +provide_medical_attention(patient)
   :   available(amb)
   <- drive_to(amb,pt);
      call(air_amb);
      drive_to(amb,h).


======== Validation: Plan Consistency ====
replace  call(air_amb) at step 2  valid  edit
delete  get_out(pt,amb) at step 4  valid  edit

======== Validation: Goal State ========
goal  state  is  reachable

Computing Time: 9 ms
```

6.2.1 Experimental Results

To conduct our evaluation, we use Monroe Plan Corpus [19]. For the purposes of this work, the corpus has been rewritten using AgentSpeak(L) programming language and executed using Jason interrupter [75]. Observation traces have been collected using a debugging tool called Mind Inspector [75]. Unknown plans have been classified using C4.5 decision tree algorithms [150].

Performance Results

We ran a number of experiments to test the scalability of our approach with respect to the number of new classes of observations. The corpus used in these experiments has been executed to consider all possible traces (the number of executed actions is 80). New classes of observations were artificially added to the observation traces. The number of capabilities is set to 20. Figure 6.6 compares the number of explanations generated by our approach for a different number of new classes of observations.

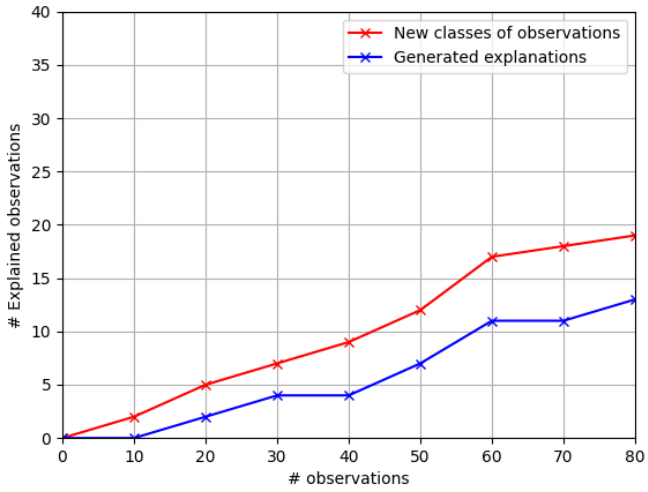


Figure 6.6: Number of generated explanations to new classes of observations

Our first study shows that with a moderate number of new classes of observations and a reasonable number of capabilities, abductive plan editing would possibly improve the explanatory power of plan libraries in open environment settings. Note that the scalability of the abductive plan editing should be tied to the performance of the classification task. Nevertheless, the results depicted in Figure 6.6 are partially independent of the used classifier (C4.5 decision tree). Actually, we allowed for unknown plans that contain unreliable actions to be considered as inputs. We argue that additional features are required for a more-fine classification of unknown plans.

Speed

Aiming to find out how fast abductive plan editing would take to explain unknown plans, we ran a number of experiments with Monroe Plan Corpus. Recall that executing action models (i.e., planning) is another technique to explain unknown plans. Hence, we use diverse planning [151] as a benchmark to assess the complexity of our approach. Diverse planning aims at discovering a set of plans that are within a certain distance from each other. The discovered set is then used to compute the closest plan to the observation sequence. Figure 6.7 shows the time required to discover a valid explanation for variant numbers of new classes of observations using diverse planning (as used by LPG-d planner [151]) to the time required by our plan editing approach.

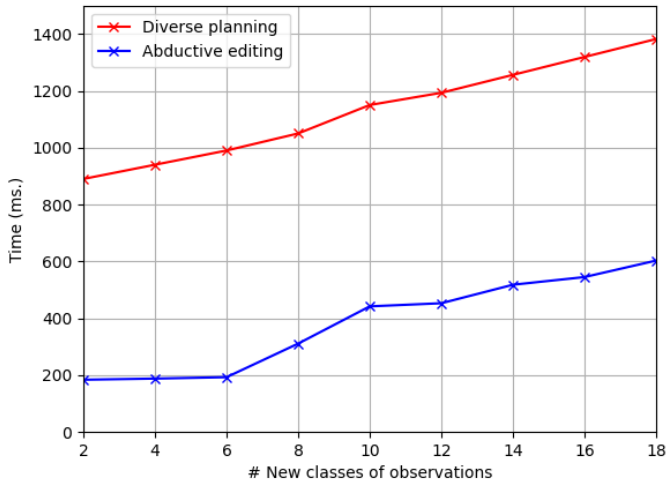


Figure 6.7: Required time for plan editing vs. diverse planning.

Figure 6.7 shows that, unlike diverse planning for different numbers of new classes of observations, plan editing is a relatively faster approach to explaining unknown plans. However, the reader should keep two details in mind. First, the performance of plan editing is tied to the performance of the used distance algorithm (in our case, Levenshtein distance). A more fine-grained evaluation, therefore, should include diffident distance algorithms (e.g., Hamming distance). Secondly, during our experiments, we noticed that the size of the capability library could highly affect the performance of diverse planning and plan editing. We will investigate these two details as part of our future work.

6.3 Plugin III: Explaining by Beliefs

The XPlaM Explaining by Beliefs plugin allows the user to obtain detailed explanations of observed actions using a number of settings. Fundamentally, this plugin helps answer the following question for any step of plan execution: *What must have been known in detail for the agent to perform a particular action over another?*. The answer to this question may also help resolve the problem of explaining irrational explanations. Table 6.9 illustrates the required inputs for Explaining by Beliefs plugin.

Table 6.9: Explaining by Beliefs inputs

Data	Description	Extension
Observation log	An alternating sequence of beliefs and actions	.csv
Plan library	A collection of AgentSpeak(L) plans whose execution generates the used log.	.xml
Knowledge base	A set of rules and axioms.	.txt

As Explaining by Beliefs plugin requires the past execution data associated with agent system execution, it takes an alternating sequence of beliefs and actions, which can be represented as a sequence of the form $b_0, a_1, \ldots, b_n, a_n$. Collecting such data can be implemented using audit logging tools such as Mind Inspector in Jason platform [75] and Design Tracing Tool (DTT) in JACK platform [73]. The plugin also requires a plan library whose execution generates the used observation log in a .xml extension and a knowledge base that contains rules and axioms related to the behaviour of the target system.

Running Explaining by Beliefs plugin over the above-cited inputs generates a set of association rules, each of which represents the relation between perceived beliefs and executed actions. For example, if the agent selected action a_i, then it must has known $b_i, \ldots, b_n$.

Table 6.10: Explaining by Beliefs output

Data file	Description	Extension
Association rule	A set of rules identifying the relation between actions and beliefs.	.txt

The default implementation of Explaining by Beliefs does not require the user to specify any measures of significance (e.g., support and confidence) as input. The only required inputs are a plan library, an observation log and a knowledge base. Unlike previously mentioned plugins, Explaining by Beliefs is particularly useful in collaborative settings (i.e., the target system would tell what its beliefs are at any given time). To illustrate the implementation of Explaining by Beliefs, we study part of the leading firefighter agent as described in [79].

Example 6.3. As shown in Figure 6.8, the leading FireFighter Agent (or simply FFAgent) has the goal of leading the firefighter team. FFAgent starts preparing to handle a fire incident whenever it acquires the belief fire alarm. If the equipment has not yet been collected, then the FFAgent collects the equipment, gets into the fire engine and calls the operator.

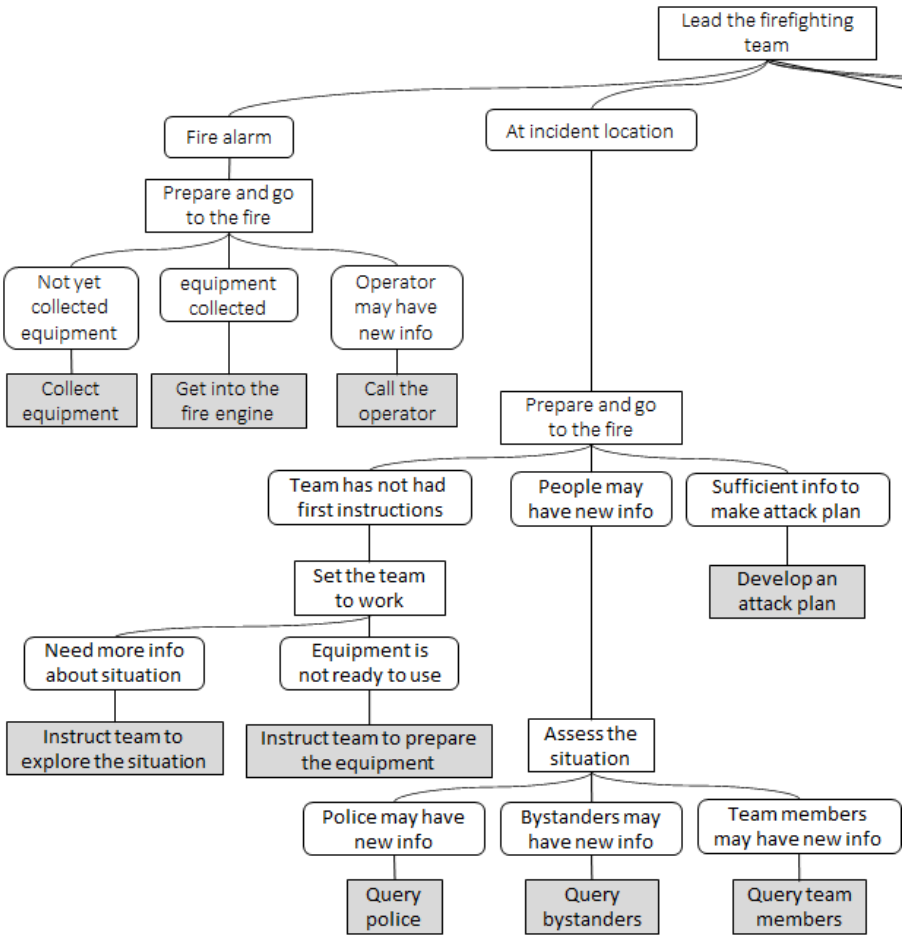


Figure 6.8: A GTH of FFAgent

Figure 6.8 illustrates the structure of the goal hierarchy to which the goals and beliefs of the FFAgent are updated. One may think of triggering action as the result of the agent's current goals ad beliefs. At every reasoning cycle, if there is no applicable action, the agent waits until its beliefs change. If multiple actions are applicable, the agent chooses an action based on a particular selection strategy (as described in section 6.1). We are interested in two modes of audit logging: (1) behaviour logs and (2) belief logs. An excerpt of the observation log associated with the plans in our example is recorded in Table 6.11.

Table 6.11: An observation log of FFAgent

timestamp	action
t3	collect(equipment)
t5	get_into(truck)
t7	call(operator)
t11	instruct(team,explore)
t13	instruct(team,prepare)
t16	query(police)
t19	develop(attack_plan)

timestamp	belief
t1	alarm(fire)
t2	not collected(equipment)
t4	collected(equipment)
t6	have(operator,new_info)
t8	at(incident,loc)
t9	not had(team,instructions)
t10	need(info,situation)

Consider the following knowledge base

$$KB = r \rightarrow \neg(d \wedge q)$$

representing a rule for the FFAgent. The propositional letter r can be read as there is no explosion danger when using water, d as it is not safe to go inside the house, and the letter q as an attack from the inside house is more effective.

There are two methods that implement Explaining by Beliefs plugin: belief Miner and Update Operator. The belief Miner method takes an alternative sequence of actions and beliefs to extract frequent if-then patterns of the form $b_0, \ldots, b_n \rightarrow a_i$ (i.e., if the agent selects the action a_i, then it has known that $b_0, \ldots, b_n$). The default implementation set *min conf* = 0.90 and *min supp* = 0.4. As XPlaM source code is published publicly, the user can tune the *min conf* and *min supp* thresholds. However, it should be noted that tuning association rules thresholds arbitrarily can be dangerous, as the typical problems of performance measurements can happen when extreme thresholds are provided without due care (e.g., high *min supp* can

return irrelevant rules). On the other hand, noisy observations might require high thresholds.

As shown above, each rule is a composite of two itemsets: (1) Antecedent, the left-hand-side of the rule, which is a set of beliefs and (2) Consequent, the right-hand-side of the rule, which is a single action. belief Miner method rules out rules that contain beliefs on both sides, actions on both sides, and rules with more than one action on the right-hand side. Also, it rules out any association rules that do not satisfy *min conf* and *min supp*. Next, the belief Miner method stores all the rules that satisfy the above conditions as a Dictionary $<$ string, List $<$ string $>>$. A dictionary (a collection of key-value pairs) datatype allows for only unique keys (actions). The value of each action is then a list of beliefs. The results below show the output of belief Miner over Table 6.11 in our running example.

[1] collect (equipment): [alarm(fire), not collected (equipment)]
[2] get_into(fire_engine): [collected (equipment)]
[3] call (operator): [have(operator,new_info)]
[4] instruct (team,explore): [at(incident , loc), ... , need(info , situation)]
[5] instruct (team,prepare): [not ready(equipment)]
[6] query(police): [have(people,new_info), have(police ,new_info)]
[7] develop(attack_plan): [sufficient (info)]

The Update Operator method takes the generated dictionary as an input to compute the updated belief-based explanation for each key (i.e., action) in the dictionary. For each contiguous keys in the dictionary, Update Operator combines their values in a new list, such that the later values take over the former ones. That is if the later list contains undoes (e.g., negation) or different assignments (e.g., location(Agent) = 1.1) than the former list, then Update Operator updates the new list with the later beliefs. The results below show the an expert of Update Operator over the dictionary illustrated above.

[1] collect (equipment): [alarm(fire), not collected (equipment)]
[2] get_into(fire_engine): [alarm(fire), collected (equipment)]

As discussed before, an updated belief-based explanation might be a set of non-deterministic possible belief states. Hence, a value of a specific key can be replaced with another value based on preceding keys. After the Explaining by Beliefs plugin

finishes parsing the uploaded inputs, the following window appears.

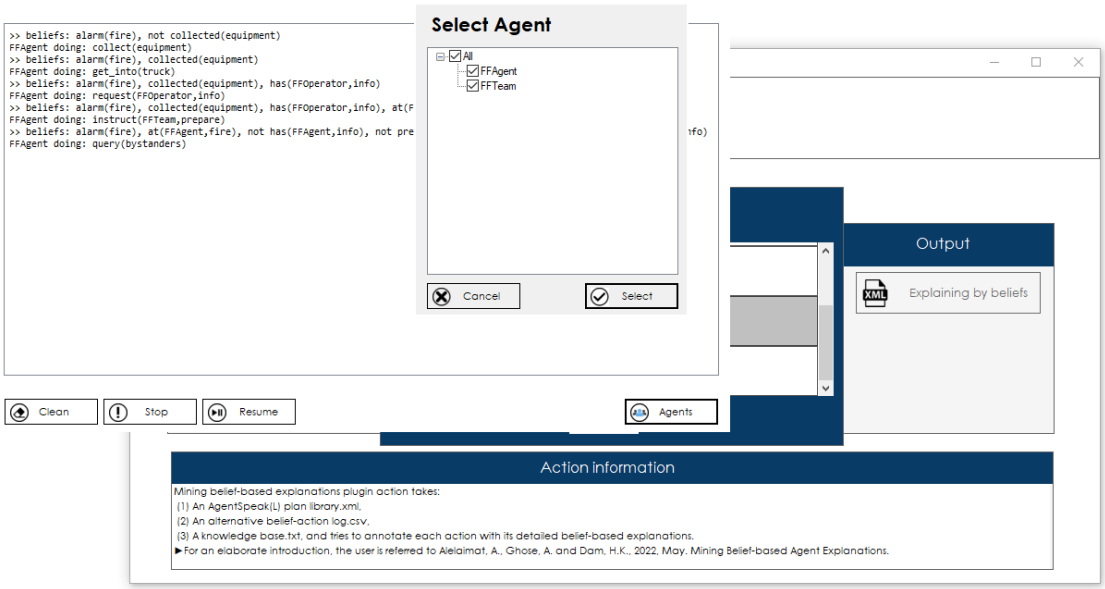


Figure 6.9: A snapshot of explaining by beliefs outputs

In this window, the user can specify which agent(s) to explain its actions in the case of multi-agent systems.

6.3.1 Experimental Results

This section presents the evaluation of Explaining by Beliefs plugin action. We evaluated Explaining by Beliefs using a synthetic log of 1000 execution instances, representing FFAgent past behaviour as described in [79]. We also used a plan library consisting of 15 plans.

We published the data sets supporting the conclusions of this section and its additional files online at https://www.kaggle.com/datasets/alelaimat/explainable-bdi-agents.

Performance Results

Our goal of this evaluation is to establish that Explaining by Beliefs can generate generally reliable explanations. To that end, we recorded the precision (i.e., number of correctly mined explanations over the total number of mined explanations) and recall (i.e., number of correctly mined explanations over the total number of actual explanations) obtained from applyingExplaining by Beliefs. We consider *min*

conf and *min supp* of the mined rules as the most important factors for mining belief-based explanations. The results are depicted in Figure 6.12 and are summarized below.

Table 6.12: Performance results for different *min conf* and *min supp*

min conf	1.00	0.95	0.9	0.85	0.8
Precision	0.8948	0.8910	0.8731	0.8487	0.8235
Recall	0.5678	0.6387	0.7265	0.7854	0.8547
min supp	0.5	0.4	0.3	0.2	0.1
Precision	0.9216	0.8741	0.8254	0.7703	0.6987
Recall	0.2257	0.3458	0.5361	0.6966	0.8815

min conf:

1. Confidence threshold considerably impacts performance results, i.e., higher *min conf* leads to higher precision but lower recall. For example, our approach achieved the highest precision of 0.89 with the lower recall of 0.56 for *min conf* of 1.00.

2. It is necessary, thus, to find a trade-off between precision and recall when varying *min conf* threshold. Hence, we use *min conf* of 0.9 for testing the impact of varying *min supp* threshold.

min supp

1. Similar to the *min conf*, varying *min supp* has a significant impact on the performance results. For example, we achieved the highest precision of 0.92 with an insignificant recall of 0.22 for *min supp* of 0.5.

2. Note that extracting association rules related to infrequent events needs *min supp* to be low. However, this, in turn, sacrifices the precision of the mined explanations.

Finally, we noticed some insightful observations through our experiments. First, the size of $\mathsf{D}_{\mathsf{All}}$ does not impact the performance results of the mined explanations *iff* $\mathsf{D}_{\mathsf{All}}$ includes all possible behaviours of the observed agent. For example, we

achieved close precision values for D_{All} of 200 and 800 execution instances, which were 0.836 and 0.842, respectively. Second, varying *min conf* and *min supp* have a significant impact on the performance of the mined explanations and, consequently, come with several limitations (e.g., time and effort).

6.4 Future Implementations

At this point in time, XPlaM has a number of limitations that require further research to address.

1. The model of plan recognition XPlaM leverages can be problematic in some cases since the target plans are not necessarily in the used library. Hence, other models of plan recognition (e.g., AI planning-based plan recognition [85]) can be more practical.

2. XPlaM has not been tested against real-world scenarios. Currently, we are improving XPlaM to deal with logs obtained from complex real-life organizations where incompleteness of knowledge and non-determinism might be present. A typical example of this is to leverage noise-tolerant plan recognition approaches, such as the work presented in [118].

3. Abductive plan editing views the leveraging of existing hypotheses to explain unknown plans (plans that involve new classes of observations). However, the state-of-the-art BDI agent framework exhibits a large number of extensions. Knowing which extension (i.e., what tools the target system has in hand) caused the new classes of observations permits a more fine-grained imitation of the target system. A number of these extensions are of interest, including the application of norms mining [152], learning planning risk mitigation [153] and change impact analysis in agent systems [154].

4. XPlaM views the process of explaining by the BDI handles (e.g., beliefs, plan and selection strategies) as an inclusive process. However, it is generally recognized that helpful explanations take explainees who are receiving the explanation into consideration. Thus, we plan to extend our evaluation by involving industrial practitioners familiar with agent-oriented programming.

6.5 Chapter Summary

This chapter described the implementation of the XPlaM toolkit, designed to support the explanation techniques offered in the previous three chapters. A key novelty of XPlaM is the fusion of two powerful styles: reverse engineering (to re-create the visible behaviour of a target system) and explainable agency (for viewing the behaviour of a target system through the lens of BDI agents).

For each plugin, we conducted a set of empirical evaluations, which suggest that XPlaM can provide generally reliable results. Compared to the workflow-based discovery of agent models, the XPlaM Abductive Design of BDI Agents plugin has achieved significantly higher accuracy on two synthetic yet realistic observation logs. A significant aspect of this high performance is related to XPlaM's ability to infer the internal workings of the target systems. The plugin achieved linear time complexity O(n) with respect to the number of observed instances. The XPlaM Abductive Plan Editing plugin has achieved good scalability in leveraging imperfect explanations with respect to the number of new classes of observations. The plugin achieved significantly higher speed than executing action models to best explain unknown plans. The XPlaM Explaining by Beliefs plugin has achieved 0.87 precision and 0.72 recall using *min conf* = 0.9. Also, it has achieved 0.92 precision but 0.22 recall using *min supp* = 0.5. As discussed in section 6.3, although Explaining by Beliefs plugin can archive good performance measurements; the user must find the suitable trade-off between precision and recall when varying *min conf* and *min supp* thresholds.

Despite all of the above, it should be emphasized that XPlaM cannot provide complete and correct BDI agent-based Digital Twins of anthropomorphic systems (especially in competitive settings). Hence, we discussed a number of future implementations in section 6.4.

Chapter 7

Conclusion and Future Work

This chapter concludes the contributions that we carried out through this thesis and provides pointers to a number of possible future directions of research. It is useful at this point to revisit the general problem being addressed, which was formulated as follows: “Find a BDI agent program that best explains the behaviour of a target system on the basis of its observation logs".

7.1 Thesis Contributions

We presented three data-driven explanation techniques to support developers in the design of Digital Twins of anthropomorphic systems. Through examples and initial results, each of these techniques demonstrated its applicability in a setting where there is limited information available about the target system behaviour. We summarize these explanation techniques as follows:

- **Abductive Design of BDI Agents (Chapter 3)**. We have proposed an end-to-end methodology for the development of BDI agent-based Digital Twins. Our contribution is able, to a certain extent, to re-create the externally visible behaviour and the internal workings of the target system through the lens of BDI agents. To this end, we proposed three explanation techniques: (1) Abductive plan recognition, (2) Explaining by beliefs, and (3) Explaining by selection strategies. Our evaluation of the abductive design of BDI-based Digital Twins using two synthetic yet realistic observation logs suggests the applicability of this contribution to addressing the research question of what

is an end-to-end methodology for the development of BDI agent-based Digital Twins.

- **Leveraging Imperfect Explanations (Chapter 4)**. A drawback to abductive plan recognition (APR) is the assumption that target plans are usually not from the used plan library. Actually, appealing to APR applications in such settings would only generate imperfect explanations. Hence, we have proposed an approach to address the problem of *leveraging imperfect explanations*, the process of modifying existing hypotheses to explain an observed sequence of actions. We have shown that when the observed system operates in a domain model known to the observer, imperfect explanations can be a valuable guide to explain unknown plans that involve new classes of observations. To avoid arbitrary modification of hypotheses, we have also introduced a classification model that can determine the settings (e.g., noisy or explanatory) in which an unknown plan has been observed. The evaluation of the proposed approach using the Monroe Plan Corpus demonstrates the responsibility of this contribution to answering the research question of how imperfect explanations can be leveraged to recognize unknown plans.

- **Explanation Mining (Chapter 5)**. We have developed a data-driven approach to mining and validating explanations (and specifically belief-based explanations) of target system actions. To do so, we have employed an association rule miner to discover regularities between external events and the action of the target system. Also, we developed a state update operator (i.e., an operator that defines how the specification of a belief state is updated as a consequence of the system's perception of the environment) to contextualise the explanation (i.e., provide users with detailed explanations) and validate the discovered explanations. The evaluation results demonstrated that mining belief-based explanations is able to answer the question of how to leverage the historical data associated with the business environment to provide detailed explanations about the organization's actions.

- **XPlaM (Chapter 6)**. We have developed a toolkit to support the algorithms and the conceptual components presented in this thesis. XPlaM includes a

user-friendly interface, requiring very little agent programming knowledge of the user. The results from each evaluation demonstrated the benefit of XPlaM in answering the research question of whether it is possible to build a toolkit to support programmers in developing first-cut Digital Twins of organizations. Recall that addressing this question is related to the applicability of the above-cited contributions.

7.2 Future Work

This section highlights a number of future directions that could be pursued on the path to a more fine-grained design of BDI agent-based Digital Twins.

- **Selection strategies**. Due to the practical results of this work, theoretical strategies to plan and intention selections have not been considered. Nevertheless, they are realistic and, potentially, practical selection strategies. For example, the target system may adopt a summary information-based intention selection similar to the one in [88] to implement positive or avoid negative interactions between goals. The target system may also select options based on their costs [89] or particular preferences and moral values [155]. Of these, we are interested in the concurrent execution of goals and cost-based selection strategies.

- **Learning Planning Digital Twins**. To provide support for run-time adaptation, Digital Twins must address the question of what will be carried out to handle given changes in the target system or the operating context. Given the current state of the target system and a specification of its goals, this can be viewed as a planning problem. The planning operators are capabilities that appear as actions either in the currently deployed plan design or in other designs in the target system repository (e.g., as illustrated in chapter 4). The output generated by a planner will therefore be action sequences representing ways in which the target system will react to given changes.

- **Explaining by Norms**. Designing Digital Twins of anthropomorphic systems is the problem of recreating the external behaviour and the internal workings of a target system, and related literature in this domain has limited

itself to this case. We have done so as well, disregarding that social norms are a powerful force in anthropomorphic systems. We view the problem of designing normative Digital Twins as a two-step task: identification and imitation. Norm identification can be viewed as the problem of inferring societal rules that govern the behaviour of a target system after observing its behaviour. Norm imitation can be seen as updating existing plans to best explain an observed sequence of actions.

Bibliography

(1) M. Halpern, *Busting the Myth of Digital Twins and Planning Them Realistically*, tech. rep., Gartner, Inc, 2017.

(2) Altair, *2022 Digital Twin Global Survey Report*, tech. rep., Altair, Inc, 2022.

(3) S. Turner, *Feasibility of an Immersive Digital Twin Report*, tech. rep., High Value Manufacturing Catapult, 2018.

(4) B. D. Allen, "Digital twins and living models at NASA", *Digital Twin Summit*, 2021.

(5) S. Hewitt, L. Margetts and A. Revell, "Building a digital wind farm", *Archives of Computational Methods in Engineering*, 2018, **25**, 879–899.

(6) B. Anderson, *What is Azure Digital Twins?*, 2020, `https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/digital-twins/overview.md`.

(7) O. Corporation, *Oracle Cloud Developing Applications with Oracle Internet of Things Cloud Service*, 2020, `https://docs.oracle.com/en/cloud/paas/iot-cloud/iotgs/oracle-iot-digital-twin-implementation.html`.

(8) K. E. Watkins, "Part five: Do organizations learn?. Of course organizations learn!", *New directions for adult and continuing education*, 1996, **1996**, 89–96.

(9) S. Kuz, M. P. Mayer, S. Müller and C. M. Schlick, "Using anthropomorphism to improve the human-machine interaction in industrial environments (Part I)", *International conference on digital human modeling and applications in health, safety, ergonomics and risk management*, 2013, 76–85.

(10) J. A. Andersen, "An organization called Harry", *Journal of Organizational Change Management*, 2008.

(11) A. D. Rubio and J. de Lara, "Semantic Digital Twins for Organizational Development", *Second International Workshop on Semantic Digital Twins*, 2021.

(12) U. V. Riss, H. Maus, S. Javaid and C. Jilek, "Digital twins of an organization for enterprise modeling", *The Practice of Enterprise Modeling: 13th IFIP Working Conference, PoEM 2020, Riga, Latvia, November 25–27, 2020, Proceedings 13*, 2020, 25–40.

(13) R. Parmar, A. Leiponen and L. D. Thomas, "Building an organizational digital twin", *Business Horizons*, 2020, **63**, 725–736.

(14) M. Georgeff and A. Rao, "Modeling rational agents within a BDI-architecture", *Proc. 2nd Int. Conf. on Knowledge Representation and Reasoning (KR'91). Morgan Kaufmann*, 1991, 473–484.

(15) M. P. Georgeff and A. Rao, "An abstract architecture for rational agents", *Proc. of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992, 439–449.

(16) K. Hribernik, G. Cabri, F. Mandreoli and G. Mentzas, "Autonomous, context-aware, adaptive Digital Twins—State of the art and roadmap", *Computers in Industry*, 2021, **133**, 103508.

(17) H. H. Zhuo, Y. Zha, S. Kambhampati and X. Tian, "Discovering underlying plans based on shallow models", *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2020, **11**, 1–30.

(18) B. R. Barricelli, E. Casiraghi and D. Fogli, "A survey on digital twin: Definitions, characteristics, applications, and design implications", *IEEE access*, 2019, **7**, 167653–167671.

(19) N. Blaylock and J. Allen, "Generating artificial corpora for plan recognition", *International Conference on User Modeling*, 2005, 179–188.

(20) A. S. Rao, "AgentSpeak (L): BDI agents speak out in a logical computable language", *European workshop on modelling autonomous agents in a multi-agent world*, 1996, 42–55.

(21) S. A. Kumar, R. Madhumathi, P. R. Chelliah, L. Tao and S. Wang, "A novel digital twin-centric approach for driver intention prediction and traffic congestion avoidance", *Journal of Reliable Intelligent Environments*, 2018, **4**, 199–209.

(22) F. Biesinger, D. Meike, B. Kraß and M. Weyrich, "A Case Study for a Digital Twin of Body-in-White Production Systems General Concept for Automated Updating of Planning Projects in the Digital Factory", *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, **1**, 19–26.

(23) F. Xiang, Z. Zhi and G. Jiang, "Digital Twins technolgy and its data fusion in iron and steel product life cycle", *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, 2018, 1–5.

(24) A. Karakra, F. Fontanili, E. Lamine, J. Lamothe and A. Taweel, "Pervasive Computing Integrated Discrete Event Simulation for a Hospital Digital Twin", *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, 1–6.

(25) H. Laaki, Y. Miche and K. Tammi, "Prototyping a Digital Twin for Real Time Remote Control Over Mobile Networks: Application of Remote Surgery", *IEEE Access*, 2019, **7**, 20325–20336.

(26) B. Schleich, N. Anwer, L. Mathieu and S. Wartzack, "Shaping the digital twin for design and production engineering", *CIRP Annals*, 2017, **66**, 141–144.

(27) E. Negri, L. Fumagalli and M. Macchi, "A review of the roles of digital twin in cps-based production systems", *Procedia Manufacturing*, 2017, **11**, 939–948.

(28) F. Tao, H. Zhang, A. Liu and A. Nee, "Digital Twin in Industry: State-of-the-Art", *IEEE Transactions on Industrial Informatics*, 2018.

(29) D. Renzi, D. Maniar, S. McNeill, C. Del Vecchio et al., "Developing a digital twin for floating production systems integrity management", *OTC Brasil*, 2017.

(30) T. Poddar et al., "Digital Twin Bridging Intelligence Among Man, Machine and Environment", *Offshore Technology Conference Asia*, 2018.

(31) M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems", *Transdisciplinary perspectives on complex systems: New findings and approaches*, 2017, 85–113.

(32) S. I. Shafiq, C. Sanin, E. Szczerbicki and C. Toro, "Virtual engineering object/virtual engineering process: a specialized form of cyber physical system for Industrie 4.0", *Procedia Computer Science*, 2015, **60**, 1146–1155.

(33) J. Grotepass, H. Speyer and R. Kaiser, "Generation and use of human 3D-CAD models", *Third International Conference on Experimental Mechanics*, 2002, **4537**, 501–505.

(34) M. Shafto, M. Conroy, R. Doyle, E. Glaessgen, C. Kemp, J. LeMoigne and L. Wang, "Draft modeling, simulation, information technology & processing roadmap", *Technology Area*, 2010, **11**.

(35) K. Borodulin, G. Radchenko, A. Shestakov, L. Sokolinsky, A. Tchernykh and R. Prodan, "Towards digital twins cloud platform: Microservices and computational workflows to rule a smart factory", *Proceedings of the10th International Conference on Utility and Cloud Computing*, 2017, 209–210.

(36) M. Schluse, L. Atorf and J. Rossmann, "Experimentable digital twins for model-based systems engineering and simulation-based development", *2017 Annual IEEE International Systems Conference (SysCon)*, 2017, 1–8.

(37) Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison", *Ieee Access*, 2018, **6**, 3585–3593.

(38) A. Khan, M. Dahl, P. Falkman and M. Fabian, "Digital Twin for Legacy Systems: Simulation Model Testing and Validation", *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 2018, 421–426.

(39) M. Eckhart and A. Ekelhart, "Towards Security-Aware Virtual Environments for Digital Twins", *Proceedings of the 4th ACM Workshop on Cyber-Physical System Security*, 2018, 61–72.

(40) M. R. Shahriar, S. N. Al Sunny, X. Liu, M. C. Leu, L. Hu and N.-T. Nguyen, "MTComm Based Virtualization and Integration of Physical Machine Operations with Digital-Twins in Cyber-Physical Manufacturing Cloud", *2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2018, 46–51.

(41) D. Preuveneers, W. Joosen and E. Ilie-Zudor, "Robust digital twin compositions for Industry 4.0 smart manufacturing systems", *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*, 2018, 69–78.

(42) S. Malakuti and S. Grüner, "Architectural aspects of digital twins in IIoT systems", *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, 2018, 12.

(43) R. Anderl, S. Haag, K. Schützer and E. Zancul, "Digital twin technology–An approach for Industrie 4.0 vertical and horizontal lifecycle integration", *it-Information Technology*, 2018, **60**, 125–132.

(44) B. Korth, C. Schwede and M. Zajac, "Simulation-ready digital twin for realtime management of logistics systems", *2018 IEEE International Conference on Big Data (Big Data)*, 2018, 4194–4201.

(45) N. Stojanovic and D. Milenovic, "Data-driven Digital Twin approach for process optimization: an industry use case", *2018 IEEE International Conference on Big Data (Big Data)*, 2018, 4202–4211.

(46) W. Luo, T. Hu, W. Zhu and F. Tao, "Digital twin modeling method for CNC machine tool", *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, 2018, 1–4.

(47) G. S. Martınez, S. Sierla, T. Karhela and V. Vyatkin, "Automatic Generation of a Simulation-Based Digital Twin of an Industrial Process Plant", *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, 2018, 3084–3089.

(48) A. Redelinghuys, A. Basson and K. Kruger, "A Six-Layer Digital Twin Architecture for a Manufacturing Cell", *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*, 2018, 412–423.

(49) M. Mühlhäuser, "Smart products: An introduction", *European Conference on Ambient Intelligence*, 2007, 158–164.

(50) R. Schmidt, M. Möhring, R.-C. Härting, C. Reichstein, P. Neumaier and P. Jozinović, "Industry 4.0-potentials for creating smart products: empirical research results", *International Conference on Business Information Systems*, 2015, 16–27.

(51) A. Radziwon, A. Bilberg, M. Bogers and E. S. Madsen, "The smart factory: exploring adaptive and flexible manufacturing solutions", *Procedia engineering*, 2014, **69**, 1184–1190.

(52) A. Bécue, Y. Fourastier, I. Praça, A. Savarit, C. Baron, B. Gradussofs, E. Pouille and C. Thomas, "CyberFactory# 1—Securing the industry 4.0 with cyber-ranges and digital twins", *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018, 1–4.

(53) H. Zhang, G. Zhang and Q. Yan, "Dynamic resource allocation optimization for digital twin-driven smart shopfloor", *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, 2018, 1–5.

(54) J. Banks, *Handbook of simulation: principles, methodology, advances, applications, and practice*, John Wiley & Sons, 1998.

(55) A. Borshchev, *The big book of simulation modeling: multimethod modeling with AnyLogic 6*, AnyLogic North America Chicago, 2013.

(56) J. Merel, Y. Tassa, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne and N. Heess, "Learning human behaviors from motion capture by adversarial imitation", *arXiv preprint arXiv:1707.02201*, 2017.

(57) M. Eckhart and A. Ekelhart, "A Specification-based State Replication Approach for Digital Twins", *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, 2018, 36–47.

(58) L. Atorf and J. Roßmann, "Interactive Analysis and Visualization of Digital Twins in High-Dimensional State Spaces", *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2018, 241–246.

(59) W. Luo, T. Hu, C. Zhang and Y. Wei, "Digital twin for CNC machine tool: modeling and using strategy", *Journal of Ambient Intelligence and Humanized Computing*, 2019, **10**, 1129–1140.

(60) R. Lamb, "Calibration of a conceptual rainfall-runoff model for flood frequency estimation by continuous simulation", *Water Resources Research*, 1999, **35**, 3103–3114.

(61) R. Zhao, D. Yan, Q. Liu, J. Leng, J. Wan, X. Chen and X. Zhang, "Digital twin-driven cyber-physical system for autonomously controlling of micro punching system", *IEEE Access*, 2019, **7**, 9459–9469.

(62) N. Nikolakis, K. Alexopoulos, E. Xanthakis and G. Chryssolouris, "The digital twin implementation for linking the virtual representation of human-based production tasks to their physical counterpart in the factory-floor", *International Journal of Computer Integrated Manufacturing*, 2019, **32**, 1–12.

(63) M. Wooldridge, *An introduction to multiagent systems*, John wiley & sons, 2009.

(64) N. R. Jennings, "An agent-based approach for building complex software systems", *Communications of the ACM*, 2001, **44**, 35–41.

(65) M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice", *The knowledge engineering review*, 1995, **10**, 115–152.

(66) S. Franklin and A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", *International workshop on agent theories, architectures, and languages*, 1996, 21–35.

(67) P. Maes, "Modeling adaptive autonomous agents", *Artificial life*, 1993, **1**, 135–162.

(68) J. E. Laird, *The Soar cognitive architecture*, MIT press, 2019.

(69) J. R. Anderson, M. Matessa and C. Lebiere, "ACT-R: A theory of higher level cognition and its relation to visual attention", *Human–Computer Interaction*, 1997, **12**, 439–462.

(70) M. Bratman, "Intention, plans, and practical reason", 1987.

(71) A. I. Goldman, "The psychology of folk psychology", *Behavioral and Brain sciences*, 1993, **16**, 15–28.

(72) D. C. Dennett, *The intentional stance*, MIT press, 1987.

(73) M. Winikoff, "JACK™ intelligent agents: an industrial strength platform", *Multi-Agent Programming*, 2005, 175–193.

(74) K. V. Hindriks, F. S. De Boer, W. Van der Hoek and J.-J. C. Meyer, "Agent programming in 3APL", *Autonomous Agents and Multi-Agent Systems*, 1999, **2**, 357–401.

(75) R. H. Bordini, J. F. Hübner and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*, John Wiley & Sons, 2007.

(76) A. B. Arrieta, N. Dıaz-Rodrıguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcıa, S. Gil-López, D. Molina, R. Benjamins et al., "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI", *Information fusion*, 2020, **58**, 82–115.

(77) R. Roscher, B. Bohn, M. F. Duarte and J. Garcke, "Explainable machine learning for scientific insights and discoveries", *Ieee Access*, 2020, **8**, 42200–42216.

(78) S. Anjomshoae, A. Najjar, D. Calvaresi and K. Främling, "Explainable agents and robots: Results from a systematic literature review", *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, 2019, 1078–1088.

(79) M. Harbers, K. van den Bosch and J.-J. Meyer, "Design and evaluation of explainable BDI agents", *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2010, **2**, 125–132.

(80) F. Kaptein, J. Broekens, K. Hindriks and M. Neerincx, "Personalised self-explanation by robots: The role of goals versus beliefs in robot-action explanation for children and adults", *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017, 676–682.

(81) M. Liu, S. Fang, H. Dong and C. Xu, "Review of digital twin about concepts, technologies, and industrial applications", *Journal of Manufacturing Systems*, 2021, **58**, 346–361.

(82) C. Jennings, D. Wu and J. Terpenny, "Forecasting obsolescence risk and product life cycle with machine learning", *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 2016, **6**, 1428–1439.

(83) T. Levitt et al., *Exploit the product life cycle*, Graduate School of Business Administration, Harvard University, 1965, vol. 43.

(84) W. Van der Aalst, T. Weijters and L. Maruster, "Workflow mining: Discovering process models from event logs", *IEEE transactions on knowledge and data engineering*, 2004, **16**, 1128–1142.

(85) M. Ramırez and H. Geffner, "Probabilistic plan recognition using off-the-shelf classical planners", *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

(86) A. Abdulrahman, D. Richards and A. A. Bilgin, "Reason explanation for encouraging behaviour change intention", *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2021, 68–77.

(87) M. Waters, L. Padgham and S. Sardina, "Evaluating coverage based intention selection.", *AAMAS*, 2014, **14**, 957–964.

(88) J. Thangarajah and L. Padgham, "Computationally effective reasoning about goal interactions", *Journal of Automated Reasoning*, 2011, **47**, 17–56.

(89) K. H. Dam and M. Winikoff, "Cost-based BDI plan selection for change propagation", *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, 2008, 217–224.

(90) J. Faccin and I. Nunes, "Bdi-agent plan selection based on prediction of plan outcomes", *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2015, **2**, 166–173.

(91) A. Karakra, F. Fontanili, E. Lamine, J. Lamothe and A. Taweel, "Pervasive computing integrated discrete event simulation for a hospital digital twin", *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, 1–6.

(92) X. Chen, E. Kang, S. Shiraishi, V. M. Preciado and Z. Jiang, "Digital behavioral twins for safe connected cars", *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018, 144–153.

(93) M. P. Sindlar, M. M. Dastani, F. Dignum and J.-J. C. Meyer, "Mental state abduction of BDI-based agents", *International Workshop on Declarative Agent Languages and Technologies*, 2008, 161–178.

(94) M. P. Sindlar, M. M. Dastani, F. Dignum and J.-J. C. Meyer, "Explaining and predicting the behavior of BDI-based agents in role-playing games", *International Workshop on Declarative Agent Languages and Technologies*, 2009, 174–191.

(95) P. A. Jaques and R. M. Viccari, "A BDI approach to infer student's emotions", *Ibero-American Conference on Artificial Intelligence*, 2004, 901–911.

(96) A. Guerra-Hernández, A. E. Fallah-Seghrouchni and H. Soldano, "Learning in BDI multi-agent systems", *International Workshop on Computational Logic in Multi-Agent Systems*, 2004, 218–233.

(97) D. Singh, S. Sardina, L. Padgham and S. Airiau, "Learning context conditions for BDI plan selection", *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 2010, 325–332.

(98) M. Bosello and A. Ricci, "From programming agents to educating agents–a jason-based framework for integrating learning in the development of cognitive agents", *International Workshop on Engineering Multi-Agent Systems*, 2019, 175–194.

(99) C. Olivia, C.-F. Chang, C. F. Enguix and A. K. Ghose, "Case-based BDI agents: an effective approach for intelligent search on the world wide web", *AAAI spring symposium on intelligent agents*, 1999, 22–24.

(100) W. Van Der Aalst, *Process mining: data science in action*, Springer, 2016, vol. 2.

(101) A. Rozinat and W. M. van der Aalst, "Decision mining in ProM", *International Conference on Business Process Management*, 2006, 420–425.

(102) M. De Leoni and W. M. van der Aalst, "Data-aware process mining: discovering decisions in processes using alignments", *Proceedings of the 28th annual ACM symposium on applied computing*, 2013, 1454–1461.

(103) S. Suriadi, M. T. Wynn, J. Xu, W. M. van der Aalst and A. H. ter Hofstede, "Discovering work prioritisation patterns from event logs", *Decision Support Systems*, 2017, **100**, 77–92.

(104) A. Pika, M. T. Wynn, C. J. Fidge, A. H. ter Hofstede, M. Leyer and W. M. van der Aalst, "An extensible framework for analysing resource behaviour using event logs", *International Conference on Advanced Information Systems Engineering*, 2014, 564–579.

(105) A. Senderovich, M. Weidlich, A. Gal and A. Mandelbaum, "Mining resource scheduling protocols", *International Conference on Business Process Management*, 2014, 200–216.

(106) V. Augusto, X. Xie, M. Prodel, B. Jouaneton and L. Lamarsalle, "Evaluation of discovered clinical pathways using process mining and joint agent-based discrete-event simulation", *2016 Winter Simulation Conference (WSC)*, 2016, 2135–2146.

(107) B. Van Dongen, J. van Luin and E. Verbeek, "Process mining in a multi-agent auctioning system", *Proceedings of the 4th International Workshop on Modelling of Objects, Components, and Agents, Turku*, 2006, 145–160.

(108) A. Rozinat, S. Zickler, M. Veloso, W. M. van der Aalst and C. McMillen, "Analyzing multi-agent activity logs using process mining techniques", *Distributed autonomous robotic systems 8*, 2009, 251–260.

(109) H. Xu, B. T. R. Savarimuthu, A. Ghose, E. Morrison, Q. Cao and Y. Shi, "Automatic BDI plan recognition from process execution logs and effect logs", *International Workshop on Engineering Multi-Agent Systems*, 2013, 274–291.

(110) L. Padgham and P. Lambrix, "Agent capabilities: Extending BDI theory", *AAAI/IAAI*, 2000, 68–73.

(111) L. De Silva, S. Sardina and L. Padgham, "First principles planning in BDI systems", *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 2009, 1105–1112.

(112) R. C. Cardoso, L. A. Dennis and M. Fisher, "Plan library reconfigurability in BDI agents", *International Workshop on Engineering Multi-Agent Systems*, 2019, 195–212.

(113) P. Singla and R. J. Mooney, "Abductive markov logic for plan recognition", *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

(114) C. W. Geib and R. P. Goldman, "A probabilistic plan recognition algorithm based on plan tree grammars", *Artificial Intelligence*, 2009, **173**, 1101–1132.

(115) R. Mirsky, Y. Gal and S. M. Shieber, "CRADLE: an online plan recognition algorithm for exploratory domains", *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2017, **8**, 1–22.

(116) O. Uzan, R. Dekel, O. Seri et al., "Plan recognition for exploratory learning environments using interleaved temporal search", *AI Magazine*, 2015, **36**, 10–21.

(117) O. Amir et al., "Plan recognition in virtual laboratories", *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

(118) J. Massardi, M. Gravel and E. Beaudry, "Error-tolerant anytime approach to plan recognition using a particle filter", *Proceedings of the International Conference on Automated Planning and Scheduling*, 2019, **29**, 284–291.

(119) S. Sohrabi, A. V. Riabov and O. Udrea, "Plan Recognition as Planning Revisited.", *IJCAI*, 2016, 3258–3264.

(120) G. Sukthankar and K. P. Sycara, "Hypothesis Pruning and Ranking for Large Plan Recognition Problems.", *AAAI*, 2008, **8**, 998–1003.

(121) D. Avrahami-Zilberbrand and G. A. Kaminka, "Keyhole adversarial plan recognition for recognition of suspicious and anomalous behavior", *Plan, activity, and intent recognition*, 2014, 87–121.

(122) R.-J. Dzeng and Y.-C. Lin, "Intelligent agents for supporting construction procurement negotiation", *Expert Systems with Applications*, 2004, **27**, 107–119.

(123) V. Padmanabhan, A. Sattar, G. Governatori and K. Babu, "Incorporating temporal planning within a BDI architecture", *Proceedings of the 5th Indian International Conference on Artificial Intelligence, IICAI 2011*, 2011, 1618–1636.

(124) J. Thangarajah, J. Harland, D. Morley and N. Yorke-Smith, "Aborting tasks in BDI agents", *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007, 1–8.

(125) L. Padgham and P. Lambrix, "Formalisations of capabilities for BDI-agents", *Autonomous Agents and Multi-Agent Systems*, 2005, **10**, 249–271.

(126) V. P. Governatori and A. Sattar, "Actions Made Explicit in BDI", *AI 2001: Advances in Artificial Intelligence: 14th International Joint Conference on Artificial Intelligence, Adelaide, Australia, December 10-14, 2001, Proceedings*, 2003, **2256**, 390.

(127) R. Reiter, "The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression.", *Artificial and Mathematical Theory of Computation*, 1991, 359–380.

(128) R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving", *Artificial intelligence*, 1971, **2**, 189–208.

(129) P. Stringer, R. C. Cardoso, X. Huang and L. A. Dennis, "Adaptable and verifiable BDI reasoning", *arXiv preprint arXiv:2007.11743*, 2020.

(130) L. A. Dennis and M. Fisher, "Verifiable self-aware agent-based autonomous systems", *Proceedings of the IEEE*, 2020, **108**, 1011–1026.

(131) C. Fritz and S. A. McIlraith, "Monitoring Plan Optimality During Execution.", *ICAPS*, 2007, 144–151.

(132) A. Ghose and G. Koliadis, "Auditing business process compliance", *International Conference on Service-Oriented Computing*, 2007, 169–180.

(133) J. Thangarajah, L. Padgham and M. Winikoff, "Detecting & exploiting positive goal interaction in intelligent agents", *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003, 401–408.

(134) M. Harbers, K. v. d. Bosch and J.-J. C. Meyer, "A study into preferred explanations of virtual agent behavior", *International Workshop on Intelligent Virtual Agents*, 2009, 132–145.

(135) D. Calvaresi, Y. Mualla, A. Najjar, S. Galland and M. Schumacher, "Explainable multi-agent systems through blockchain technology", *International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems*, 2019, 41–58.

(136) R. S. Verhagen, M. A. Neerincx and M. L. Tielman, "A Two-Dimensional Explanation Framework to Classify AI as Incomprehensible, Interpretable, or Understandable", *International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems*, 2021, 119–138.

(137) Y. Mualla, "Explaining the behavior of remote robots to humans: an agent-based approach", 2020.

(138) A. Abdulrahman, D. Richards, H. Ranjbartabar and S. Mascarenhas, "Belief-based agent explanations to encourage behaviour change", *Proceedings of the 19th ACM International Conference on Intelligent Virtual Agents*, 2019, 176–178.

(139) *Multi-engine Aeroplane Operations and Training*, tech. rep., Civil Aviation Safety Authority, 2007.

(140) N. Howden, R. Rönnquist, A. Hodgson and A. Lucas, "JACK intelligent agents-summary of an agent infrastructure", *5th International conference on autonomous agents*, 2001, **6**.

(141) M. L. Ginsberg and D. E. Smith, "Reasoning about action I: A possible worlds approach", *Artificial intelligence*, 1988, **35**, 165–195.

(142) M. S. Winslett, *Reasoning about action using a possible models approach*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1988.

(143) P. Jackson, "On the Semantics of Counterfactuals.", 1989, 1382–1387.

(144) L. Maruster, A. Weijters, W. v. d. Aalst and A. v. d. Bosch, "Process mining: Discovering direct successors in process logs", *International Conference on Discovery Science*, 2002, 364–373.

(145) P. Fournier-Viger, U. Faghihi, R. Nkambou and E. M. Nguifo, "CMRules: Mining sequential rules common to several sequences", *Knowledge-Based Systems*, 2012, **25**, 63–76.

(146) F. Kaptein, J. Broekens, K. Hindriks and M. Neerincx, "The role of emotion in self-explanations by cognitive agents", *2017 Seventh International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*, 2017, 88–93.

(147) P. Sequeira and M. Gervasio, "Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations", *Artificial Intelligence*, 2020, **288**, 103367.

(148) D. Allemang, M. C. Tanner, T. Bylander and J. R. Josephson, "Computational Complexity of Hypothesis Assembly.", *IJCAI*, 1987, **87**, 1112–1117.

(149) A. Mas-Colell, M. D. Whinston, J. R. Green et al., *Microeconomic theory*, Oxford university press New York, 1995, vol. 1.

(150) J. R. Quinlan, *C4. 5: programs for machine learning*, Elsevier, 2014.

(151) T. A. Nguyen, M. Do, A. E. Gerevini, I. Serina, B. Srivastava and S. Kambhampati, "Generating diverse plans to handle unknown and partially known user preferences", *Artificial Intelligence*, 2012, **190**, 1–31.

(152) D. Avery, H. K. Dam, B. T. R. Savarimuthu and A. Ghose, "Externalization of software behavior by the mining of norms", *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 2016, 223–234.

(153) A. Alelaimat, M. Santipuri, Y. Gou and A. Ghose, "Learning Planning Model for Semantic Process Compensation", *Service Research and Innovation*, 2015, 35–47.

(154) H. K. Dam and A. Ghose, "Automated change impact analysis for agent systems", *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, 33–42.

(155) S. Cranefield, M. Winikoff, V. Dignum and F. Dignum, "No Pizza for You: Value-based Plan Selection in BDI Agents.", *IJCAI*, 2017, 178–184.