Context-aware Plan Repair in Environments shared by Multiple Agents

by

Mohannad Babli

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

A thesis submitted for the degree of

Título de Doctor por la Universitat Politècnica de València

Supervisors:	Prof. Eva Onaindía de la Rivaherrera
	Prof. Óscar Sapena Vercher

July 2023

It is not the most intellectual of the species that survives; it is not the strongest that survives; but the species that survives is the one that is best able to adapt and adjust to the changing environment in which it finds itself.

Leon C. Megginson, Civilisation Past and Present, 1963

Acknowledgements

Throughout this doctoral journey, there have been many people who, in one way or another, have helped to make this work progress. Along these lines, I want to thank you for your support.

First, I would like to thank my supervisors, and it is my honour to be their student. Thank you, professor Eva Onaindia, for the impeccable supervision and advice, for believing in my capabilities and research ideas, for giving me the opportunity to work with you, and for passing on your knowledge and your working spirit to me, as well as for your patience in the productive discussions that have helped to improve this entire work. I would like to thank my co-supervisor, professor Óscar Sapena Vercher. I am genuinely grateful to you for putting me back on track during the hard times of my PhD and for your dedication and friendly structured supervision.

I would also like to express my deep gratitude to Dr Louise Cooke. She kept the persistence spark lit inside me and taught me that doing a PhD is 90% about persistence and hard work and 10% a combination of intellectual curiosity, good supervision, and a topic that is genuinely worthy of investigation.

I am forever thankful to Dr Ann O'Brien, who, although no longer with us, continues to inspire by her example and dedication to the students she served throughout her career.

In loving memory, I would like to dedicate this thesis to my father, Fouad Babli, who is no longer with us. This thesis would not have been possible without his support and love for science. As your name literally means, you are indeed the heart that nourished me throughout your existence. I hope to be as good as you thought of me, and I wish you were here to see that I achieved what I promised.

I am grateful to my beloved wife, Asmaa Ghojal. It is tough to describe all my gratitude for her in a single paragraph. Thank you so much for everything that you have done for me. Words cannot adequately express my gratitude and love for you. You always believed in me and encouraged me to progress through this doctoral journey and achieve my career goals. I also know that you were instrumental in caring for our son Adam and keeping this family together. For that, I can never thank you enough.

I am grateful for my colleagues and friends, Jaime Andres Rincon Arango, Cesar Gonzalez, Jaume Jordán Prunera, and Elena del Val Noguera, who made my time during my doctoral studies less difficult.

Abstract

EXECUTION MONITORING is crucial for the success of an autonomous agent executing a plan in a dynamic environment as it influences its ability to react to changes. While executing its plan in a dynamic world, it may suffer a failure and, in its endeavour to fix the problem, it may unknowingly disrupt other agents operating in the same environment. Additionally, being rational requires the agent to be context-aware, gather information and extend what is known from what is perceived to compensate for partial or incorrect prior knowledge and achieve the best possible outcome in various novel situations.

The work carried out in this PhD thesis allows the autonomous agents executing a plan in a dynamic environment to adapt to unexpected events and unfamiliar circumstances, utilise their perception of context and provide context-aware deliberative responses for seizing an opportunity or repairing a failure without disrupting other agents. This work is focused on developing a domain-independent architecture capable of handling the requirements of such autonomous behaviour. The architecture pillars are the intelligent system for execution simulation in a dynamic environment, the context-aware knowledge acquisition for planning applications and the plan commitment repair.

The intelligent system for execution simulation in a dynamic environment allows the agent to transform the plan into a timeline, periodically update its internal state with real-world information and create timed events. Events are processed in the context of the plan; if a failure occurs, the simulator reformulates the planning problem, reinvokes a planner and resumes the execution. The simulator is a console application and has a GUI designed specifically for smart tourism.

The context-aware knowledge acquisition module utilises semantic operations to dynamically augment the predefined list of object types of the planning task with relevant new object types. This allows the agent to be context-aware of the environment and the task and reason with incomplete knowledge, boosting the system's autonomy and context-awareness.

The novel plan commitment repair strategy among multiple agents sharing the same execution environment allows the agent to repair its plan responsibly when a failure is detected. The agent utilises a new metric, plan commitment, as a heuristic to guide the search for the most committed repair plan to the original plan from the perspective of commitments made to other agents whilst achieving the original goals. Consequently, the community of agents will suffer fewer failures due to the sudden changes or will have less lost time if the failure is inevitable.

All these developed modules were investigated and evaluated in several applications, such as a tourist assistant, a kitchen appliance repair agency and a living home assistant.

Resumen

LA MONITORIZACIÓN de la ejecución de un plan es crucial para un agente autónomo que realiza su labor en un entorno dinámico, pues influye en su capacidad de reaccionar ante los cambios. Mientras ejecuta su plan puede sufrir un fallo y, en su esfuerzo por solucionarlo, puede interferir sin saberlo con otros agentes que operan en su mismo entorno. Por otra parte, para actuar racionalmente es necesario que el agente sea consciente del contexto y pueda recopilar y ampliar su información a partir de lo que percibe para poder compensar su conocimiento previo parcial o incorrecto del problema y lograr el mejor resultado posible ante las nuevas situaciones que aparecen.

El trabajo realizado en esta tesis permite a los agentes autónomos ejecutar sus planes en un entorno dinámico y adaptarse a eventos inesperados y circunstancias desconocidas. Pueden utilizar su percepción del contexto para proporcionar respuestas deliberativas conscientes y ser capaces así de aprovechar las oportunidades que surgen o reparar los fallos sin perturbar a otros agentes. Este trabajo se centra en el desarrollo de una arquitectura independiente del dominio capaz de manejar las necesidades de agentes con este tipo de comportamiento autónomo. Los tres pilares de la arquitectura propuesta los forman el sistema inteligente para la simulación de la ejecución en entornos dinámicos, la adquisición de conocimiento consciente del contexto para ampliar la base de datos del agente y la reparación de planes ante fallos u oportunidades tratando de interferir lo mínimo con los planes de otros agentes.

El sistema inteligente de simulación de la ejecución permite al agente representar el plan en una línea de tiempo, actualizar periódicamente su estado interno con información del mundo real y disparar nuevos eventos en momentos concretos. Los eventos se procesan en el contexto del plan; si se detecta un error, el simulador reformula el problema de planificación, invoca de nuevo al planificador y reanuda la ejecución. El simulador es una aplicación de consola y ofrece una interfaz gráfica diseñada específicamente para una aplicación inteligente de turismo.

El módulo de adquisición de conocimiento sensible al contexto utiliza operaciones semánticas para aumentar dinámicamente la lista predefinida de tipos de objetos de la tarea de planificación con nuevos tipos relevantes. Esto permite que el agente sea consciente de su entorno, enriquezca el modelo de su tarea y pueda razonar a partir de un conocimiento incompleto. Con todo esto se consigue potenciar la autonomía del sistema y la conciencia del contexto.

La novedosa estrategia de reparación de planes le permite a un agente reparar su plan al detectar un fallo de manera responsable con el resto de agentes que comparten su mismo entorno de ejecución. El agente utiliza una nueva métrica, el *compromiso* del plan, como función heurística para guiar la búsqueda hacia un plan solución comprometido con el plan original, en el sentido de que se trata de respetar los compromisos adquiridos con otros agentes al mismo tiempo que se alcanzan los objetivos originales. En consecuencia, la comunidad de agentes sufrirá menos fallos por cambios bruscos en el entorno o requerirá menos tiempo para ejecutar las acciones correctoras si el fallo es inevitable.

Estos tres módulos han sido desarrollados y evaluados en varias aplicaciones como un asistente turístico, una agencia de reparación de electrodomésticos y un asistente del hogar.

Resum

EL MONITORATGE de l'execució d'un pla és crucial per a un agent autònom que realitza la seua labor en un entorn dinàmic, perquè influeix en la seua capacitat de reaccionar davant els canvis. Mentre executa el seu pla pot patir una fallada i, en el seu esforç per solucionar-lo, pot interferir sense saber-ho amb altres agents que operen en el seu mateix entorn. D'altra banda, per a actuar racionalment és necessari que l'agent siga conscient del context i puga recopilar i ampliar la seua informació a partir del que percep per a poder compensar el seu coneixement previ parcial o incorrecte del problema i aconseguir el millor resultat possible davant les noves situacions que apareixen.

El treball realitzat en aquesta tesi permet als agents autònoms executar els seus plans en un entorn dinàmic i adaptar-se a esdeveniments inesperats i circumstàncies desconegudes. Poden utilitzar la seua percepció del context per a proporcionar respostes deliberatives conscients i ser capaces així d'aprofitar les oportunitats que sorgeixen o reparar les fallades sense pertorbar a altres agents. Aquest treball se centra en el desenvolupament d'una arquitectura independent del domini capaç de manejar les necessitats d'agents amb aquesta mena de comportament autònom. Els tres pilars de l'arquitectura proposada els formen el sistema intel·ligent per a la simulació de l'execució en entorns

dinàmics, l’adquisició de coneixement conscient del context per a ampliar la base de dades de l’agent i la reparació de plans davant fallades o oportunitats tractant d’interferir el mínim amb els plans d’altres agents.

El sistema intel·ligent de simulació de l’execució permet a l’agent representar el pla en una línia de temps, actualitzar periòdicament el seu estat intern amb informació del món real i disparar nous esdeveniments en moments concrets. Els esdeveniments es processen en el context del pla; si es detecta un error, el simulador reformula el problema de planificació, invoca de nou al planificador i reprén l’execució. El simulador és una aplicació de consola i ofereix una interfície gràfica dissenyada específicament per a una aplicació intel·ligent de turisme.

El mòdul d’adquisició de coneixement sensible al context utilitza operacions semàntiques per a augmentar dinàmicament la llista predefinida de tipus d’objectes de la tasca de planificació amb nous tipus rellevants. Això permet que l’agent siga conscient del seu entorn, enriquisca el model de la seua tasca i puga raonar a partir d’un coneixement incomplet. Amb tot això s’aconsegueix potenciar l’autonomia del sistema i la consciència del context.

La nova estratègia de reparació de plans li permet a un agent reparar el seu pla en detectar una fallada de manera responsable amb la resta d’agents que comparteixen el seu mateix entorn d’execució. L’agent utilitza una nova mètrica, el *compromís* del pla, com a funció heurística per a guiar la cerca cap a un pla solució compromés amb el pla original, en el sentit que es tracta de respectar els compromisos adquirits amb altres agents al mateix temps que s’aconsegueixen els objectius originals. En conseqüència, la comunitat d’agents patirà menys fallades per canvis bruscos en l’entorn o requerirà menys temps per a executar les accions correctores si la fallada és inevitable.

Aquests tres mòduls han sigut desenvolupats i avaluats en diverses aplicacions com un assistent turístic, una agència de reparació d'electrodomèstics i un assistent de la llar.

Contents

List of Figures

List of Tables

Part I

Foundation

CHAPTER 1

Introduction

WHEN diving into this PhD thesis, one will find a work that joins two fields in the realm of Artificial Intelligence (AI), the field of Automated Planning (AP) and the field of Knowledge Representation (KR). From planning to action execution, we have been exploring different approaches to boost the autonomy of intelligent agents, enabling them to react intelligently to the change in dynamic execution environments and accomplish their objectives without assistance. Moving cautiously to a world where intelligent agents constantly surround people in all daily life aspects, agents are expected to handle complex tasks, respond to change, and take the initiative according to their goals, exhibiting a higher level of autonomy. This PhD thesis presents a domain-independent approach and an integrated framework for an intelligent agent executing a plan while being context-aware of the change in the dynamic execution environment to seize an opportunity or repair a plan execution failure. This chapter is included for the readers to understand this PhD thesis's aim fully.

1.1 Motivation

Since AI was born in 1956, it has held great enthusiasm by embracing the idea of replicating human faculties such as *autonomy*. In *Autonomous Agent*, the word *Agent* comes from the Latin *agere*, which means to act; on the other hand, the word *Autonomous* etymologically comes from the Greek word *autonomous* where *auto* means self and *nomos* means rules. Therefore, an *autonomous agent* is an agent that relies on its own percepts while executing a task to achieve a goal, does not restrict itself to the prior knowledge of its designers, and has the degree of freedom to regulate its behaviour to the best outcome under specific circumstances.

The original motives behind this work are three aspects:

- To date, autonomous agents still lack autonomy.
- The potential benefits of semantic knowledge, plan execution, monitoring, and failure repairing to autonomous agents.
- The autonomous robot, shown in the episode "The Quality of Life" in the science fiction Star Trek: The Next Generation series (1992), faces a brand new situation and deliberately interferes with the plan it is supposed to execute to handle the change in the dynamic environment.

1.2 Aims and Objectives

The interest in this PhD thesis is developing a domain-independent approach and methods to boost the autonomy aspect of an agent that generates a plan to satisfy a goal state, executes that plan in a dynamic environment, and then faces unexpected events.

1.2.1 Aims

The author’s contributions allow autonomous agents to deal with the dynamic environment change, not depending solely on the prior knowledge of designers but instead relying on own percepts to provide context-aware deliberative responses for seizing an opportunity or repairing a failure. Specifically, this PhD thesis has two aims:

1. Seize an opportunity that may arise during the execution of the plan.
2. Repair a failure that occurs during the execution of the plan.

1.2.2 Objectives

To achieve the aims mentioned above, we identified five objectives:

1. To design a domain-independent monitoring and execution system (Babli, Ibáñez-Ruiz, et al. 2016) as the backbone of the whole framework.
2. To develop the techniques and the mechanisms that allow autonomous agents to deal with new data and formulate new goals to seize an opportunity or fix a plan execution failure. (Babli, Onaindia, and Marzal 2018; Babli, Marzal, et al. 2018)

3. To enhance the accuracy of autonomous agents when dealing with new information by filtering incompatible information that is unmanageable by the agents' capabilities (Babli and Onaindia 2019).

4. To design a deliberation architecture (Babli, Rincon, et al. 2021) to increase autonomous agents' autonomy and context awareness when generating goals and to fix plan execution failures.

5. To design and implement a repairing property that ensures a responsible repairing policy of execution failures among agents. The approach aims at reducing the revisions of plans among agents and the lost time due to failures (Babli, Sapena, et al. 2023).

1.3 Thesis Outline

Previous sections have briefly summarised the motivation behind this PhD thesis, the aim and the objectives. In this section, we present the outline of this PhD thesis, which is organised as follows:

1. Chapter 2 contains a state-of-the-art for automated planning, plan execution and monitoring, survival of the agent (failures and opportunities) and knowledge representation.

2. Chapter 3 provides the formalisation of the problem, a preface describing the problems addressed in this PhD thesis, an abstract description of the method to achieve the aims of this PhD thesis, a schematic sketch of the proposed framework modules, and a brief description of each module.

3. Chapter 4 presents the intelligent system for execution simulation in a dynamic environment.

4. Chapter 5 presents how planning knowledge is extended using ontologies and presents a context-aware knowledge acquisition mechanism for planning applications using ontologies to seize opportunities.

5. Chapter 6 presents the plan commitment repair approach.

6. Chapter 7 presents the concluding remarks of this PhD thesis and the future research lines.

7. Appendix A presents the case study of using the intelligent system for execution simulation in a tourism assistant application.

8. Appendix B presents the case study of using the context-aware knowledge acquisition approach in two applications: a repairing agency application and a tourist assistant application.

9. Appendix C presents the case study of a complete application as a deliberative context-aware ambient intelligence system for assisted living homes.

1.4 Related Research Activities

The results of this PhD thesis work have given rise to a series of publications referenced throughout this thesis's chapters. This section lists the research activities performed during the development of this PhD thesis, namely, the related scientific publications and research projects.

1.4.1 Publications in Journals

- Babli, M., Ó. Sapena, and E. Onaindia (2023). Plan commitment: Re-planning versus plan repair. In: *Engineering Applications of Artificial Intelligence*. 123, p. 106275. **Impact Factor (2022): 8.0, ranks 5/90 (Q1), 1st decile, in the category "ENGINEERING, MULTIDISCIPLINARY"**.

- Babli, M., J. A. Rincon, E. Onaindia, C. Carrascosa, and V. Julian (2021). Deliberative Context-Aware Ambient Intelligence System for Assisted Living Homes. *Human-centric Computing and Information Sciences (HCIS)*. Article number: 11:19. **Impact Factor (2021): 5.900, ranks 27/164 (Q1) in the category "COMPUTER SCIENCES, INFORMATION SYSTEMS"**.

1.4.2 Publications in Conferences

- Babli, M. and E. Onaindia (2019). "A context-aware knowledge acquisition for planning applications using ontologies". In: Proceedings of the 33rd International Business Information Management Association Conference, IBIMA 2019: Education Excellence and Innovation Management through Vision 2020, pp. 3602–3614. Granada, Spain. **CORE B**.

- Babli, M., Marzal, E. and E. Onaindia (2019). "On the use of ontologies to extend knowledge in online planning". In: Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS), ICAPS (2018). pp. 54-61. Delft, The Netherlands.

- Babli, M., Onaindia, E. and E. Marzal (2018). "Extending planning knowledge using ontologies for goal opportunities". In: Proceedings of the 31 st International Business Information Management Association Conference, IBIMA 2018: Education Excellence and Innovation Management through Vision 2020, pp. 3199–3208. Milan, Italy. **CORE B**.

- Marzal E, Babli M, Onaindia E, and Sebastia L. (2017). "Handling PDDL3.0 State Trajectory Constraints with Temporal Landmarks". In: Proceedings of the Workshop on Constraint Satisfaction for Planning and Scheduling (COPLAS). ICAPS'17. pp. 54-61. Delft, The Netherlands. **CORE B**.

- Babli, M., Ibáñez-Ruiz, J., Sebastia, L. Tejero, A. G. and Onaindia, E. (2016). "An Intelligent System for Smart Tourism Simulation in a Dynamic Environment". In: Proceedings of the 2nd Workshop on Artificial Intelligence and Internet of Things co-located with the 22nd European Conference on Artificial Intelligence (ECAI 2016), The Hague, Netherlands, August 30, 2016. Ed. by Spyropoulos, C. D., Pierris, G. and Tzortzis, G. Vol. 1724. CEUR Workshop Proceedings. CEUR-WS.org, pp. 15–22. **CORE C**.

1.4.3 Research Project

"GLASS: Goal management for Long-term Autonomy in Smart citieS" under grants (MINECO TIN2014-55637-C2-2-R-AR) and (TIN2017-88476-C2-1-R) (Main Researcher: Eva Onaindia). The long-term goal of fully autonomous systems is a highly ambitious objective for many fields, including artificial intelligence or robotics. The project's main objective is to analyse the problem of goal management for long-term autonomous systems, design appropriate algorithms for addressing the different components of goal management, and develop software tools that help the application of this technology to smart cities tasks.

CHAPTER 2

State of the Art

HOMO SAPIENS - man the wise - is what we call ourselves because we value our intelligence. AI is the ability of machines to use methods based on the intelligent behaviour typically evident in humans and other animals to solve complex problems. Defining intelligence as the ability to deal with new situations, solve problems, answer questions, and design plans (Coppin 2004). In this chapter, we present the background more closely related to the research lines of this PhD thesis.

2.1 Autonomy

According to (Russell et al. 2020), AI is concerned with understanding and building intelligent entities —machines that can compute how to act effectively and safely in a wide variety of novel situations. An agent is an entity that can perceive the environment through sensors and act through actuators.

On the other hand, a rational agent acts to achieve the best outcome or the best-expected outcome in the case of uncertainty. A rational agent should be autonomous —it should learn what it can to compensate for partial or incorrect prior knowledge.

Whether in open or closed, well-structured environments, autonomously performing tasks require deliberation capabilities to adapt to changing circumstances. The work of (Ingrand et al. 2017) conceptually distinguishes six functions required for successful deliberation: Planning, Acting, Perceiving, Monitoring, Learning, and Reasoning.

Autonomous behaviour is divided into three approaches (Karpas et al. 2020):

1. The programming-based approach, where there is a human programmer manually specifying the agent's desired behaviour for each possible scenario.
2. The model-based approach, where the agent uses a world model for reasoning about possible courses of action and their effects on the state of the world. Automated planning is the model-based approach to autonomous behaviour.
3. The learning-based approach, where the agent learns the correct behaviour from experience.

This PhD thesis focuses on the planning-based agent architecture (depicted in Figure 2.1), where decision-making is split between execution and planning.

More specifically, this PhD thesis is concerned with two aspects that enhance the agent's autonomy. The first is our offline processing of the model. The second is the online execution involving acting, perceiving, monitoring, replanning, or repairing. However, to fully understand our methods, we must

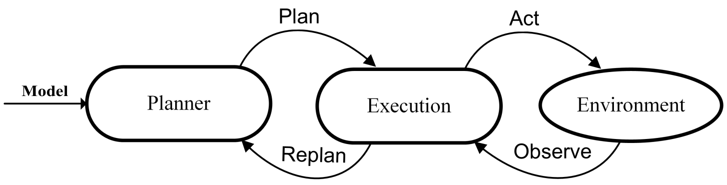


Figure 2.1: A planning-based agent architecture.

first introduce the fundamental concepts of automated planning, execution, monitoring, failures, opportunities, KR and ontologies.

2.2 Automated Planning

Planning a course of action is a crucial requirement for an autonomous agent to automatically synthesise plans of basic actions to achieve a high-level goal. This section contains the state-of-the-art of automated planning, where the key concepts, formalisations, languages and state-of-the-art planners are presented.

2.2.1 Planning Models and Modelling Languages

Unlike simple problem-solving agents that use atomic representations of the state of the world, a planning-based agent uses a structured representation of the world that is visible to the problem-solving algorithm. Most automated planning algorithms are domain-independent, meaning the same planning algorithm can solve planning problems from different domains. The input to the planner is a description of a planning problem to be solved, and the planner can work in any planning domain that satisfies some simplifying assumptions. Planning models vary depending on the type of environment, the presence of uncertainty, the types of goals, the type of actions, and whether the synthesis of plans is done offline or online. A brief description of the different model properties follows.

1. Fully observable vs partially observable environment. In fully observable environments, the agent planning algorithm has access to the complete state of the environment at all times, while partial observability may be due to noisy and inaccurate sensors or because parts of the state are simply missing or unknown.

2. Deterministic vs non-deterministic environment. The environment is deterministic if the current state can determine the next state and the action being executed. Usually, fully observable environments are deterministic and partial observability leads to non-determinism. However, it is also possible that the effects of actions are deterministic, but the environment is not if the execution environment is dynamic or partially observable.

3. Static vs dynamic environment. If the environment can change while an agent is deliberating, then we say that the environment is dynamic for that agent; otherwise, it is static.

4. Sequential vs durative actions. Sequential planning does not allow two actions to be executed simultaneously. Temporal planning uses durative actions that span over time and allow concurrency and parallelism.

5. Discrete vs continuous numeric actions' effects. Discretised effects mean the change occurs on the endpoints of the period over which an action takes place. In contrast, continuous numeric action effects involve dealing with the continuous change of time-dependent numeric variables.

6. Hard vs soft goals. An agent may have hard goals, which are unavoidable and must be met for a plan to be valid, or soft goals, which are desirable but have no bearing on the plan's success.

7. Offline vs online planning. Offline planning is used for static, known environments with available models. In contrast, online planning uses execution monitoring and replanning or repairing to recover from failures due to non-deterministic actions, unexpected events in the dynamic execution environment, or incorrect models. Online planning may return action at each iteration or be viewed as offline planning then online execution.

Different formalisms can be used for describing planning problems. Classical planning is the simplest type of planning model. The different formalisms compactly describe a transition system consisting of a set of possible states of the world, a set of possible transitions between these states, an initial state, and a set of goal states. A plan is a sequence of transitions that leads from the initial state to some goal state. Different formalisms differ in how exactly they represent this transition system.

The basic language for modelling planning problems is the input language to the Stanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson 1971). STRIPS declares actions using preconditions and effects. In STRIPS, a set of Boolean propositions describes the possible states, where a state is represented by the set of all propositions that hold in that state. Transitions are achieved by actions, where each action has preconditions, add effects, and delete effects. An action can be applied in a state if its preconditions are contained in the current state, leading to a state equal to the current state minus the delete effects plus the add effects. Finally, the goal states are represented by a set of facts, where a state satisfies the goal if and only if the goal facts are contained in that state.

Inspired by STRIPS, the PDDL (McDermott et al. 1998) was created to facilitate comparing the performance of different planners in the context of the International Planning Competition (IPC). PDDL provides more flexibility and expressiveness for modelling planning problems.

In this PhD thesis, we follow a temporal planning model, which can be solved with methods similar to classical planning where actions are durative and deterministic, and the planning tasks is represented in PDDL. On the other hand, we allow the execution environment to be dynamic and partially observable, and we deal primarily with hard goals. The following section explains the PDDL representation and versions.

2.2.2 The PDDL Representation and Versions

PDDL divides the definition of a planning problem into two parts: the domain and the problem. The planning domain characterises the behaviour of the planning task and is a fixed description of what the planner can do to achieve a set of goal conditions starting from an initial state. In contrast, the planning problem varies depending on the particular problem to be solved. Basic PDDL can handle classical planning domains, and extensions can handle non-classical domains that are continuous, partially observable, concurrent, and multi-agent.

The PDDL Versions

As planning has evolved, so has PDDL since its birth in 1998. Different versions of PDDL allow different levels of expressivity. Here follows a brief summary of each version:

1. PDDL version 1.2 (McDermott et al. 1998), used in 1st and 2nd IPC, is the basic syntax supported by planners and is primarily driven by concepts

set out for STRIPS. The language also supports some of the features of the Action Description Language (ADL) (Pednault 1989) in addition to conditional effects.

2. PDDL version 2.1 (Fox and Long 2003) was released in 2002 as the language of the 3rd IPC to deal with the challenges related to features of both temporal and numeric considerations. Such as durative actions, numeric fluents and plan metrics to optimise the plan where goals alone are not enough to generate the desired high-quality behaviour.

3. PDDL version 2.2 (Edelkamp et al. 2004) was released in 2004 as the language of the 4th IPC. PDDL 2.2 introduced two new concepts, derived predicates for creating simple reusable predicates and timed initial literals (TILs) to model expected exogenous events that occur at a specified moment of time, deadlines, or time windows.

4. PDDL version 3.0 (Gerevini and Long 2005) was released in 2006 and used in the 5th IPC. PDDL 3.0 introduced soft and state-trajectory constraints to AI planning. A soft constraint or a soft goal is a preference that we would like to satisfy, and if it is not possible, the plan is still valid, but it incurs a penalty. Preferences help differentiate the relative quality of different plans. On the other hand, state-trajectory constraints assert conditions that must be met by the entire sequence of states visited during the execution of a plan.

5. PDDL version 3.1 (Kovacs 2011) was proposed in 2008 and used in the 6th and 7th IPC in 2008 and 2011, respectively. PDDL 3.1 introduced object fluents where functions range could also be an object type.

In addition, there are other variations of PDDL, such as: PPDDL to model probabilistic environments, MAPL and MA-PDDL for Multi-Agent Planning and PDDL+ (Fox and Long 2002), to model processes that run over time and continuously affect numeric values, and events that are an instantaneous version of processes expressing uncontrollable change in the world.

There are several widely known planning tasks in the planning community, such as the domains that simulate tourists visiting points of interest, package delivery logistics, satellite observations, or rovers navigating a planet's surface. In the following two subsections, we will show an example from the Driverlog domain to demonstrate the modelling process in PDDL 2.2. The Driverlog domain has drivers that can walk between locations and trucks that can drive between locations. The trucks can be loaded or unloaded with packages, and the goal is to transport packages between locations.

The PDDL Domain

The planning domain describes the knowledge common to all problems that the domain can solve. The PDDL domain:

1. Is defined by a name. E.g., `(define (domain driverlog)...)`.
2. Has a finite set of requirements, E.g., `(:requirements :typing :durative-actions :fluents :timed-initial-literals)`.
3. Has a finite set of types defined in a reasonable hierarchy, as shown in Figure 2.2, to describe the type of objects the problem may have.
4. Has a finite set of Boolean variables (predicates), as shown in Figure 2.3, which describes the properties or relationships associated with a type.
5. Has a finite set of definitions of function variables, as shown in Figure 2.4, mapping to a numerical rational value.

```
(:types
    location locatable  — object
    agent vehicle deliverable — locatable
    driver — agent
    truck — vehicle
    package — deliverable)
```

Figure 2.2: Hierarchy of types of the Driverlog domain. Types situated on the left are subtypes of the types on the right.

```
(:predicates
    (at ?x — locatable ?y — location)
    (in ?x — deliverable ?y — vehicle)
    (driving ?x — agent ?y — vehicle)
    (empty ?x — vehicle))
```

Figure 2.3: Predicates of the Driverlog domain.

```
(:functions
    (duration-driving ?x — location ?y — location)
    (duration-walking ?x — location ?y — location)
    (duration-load-unload ?x — deliverable)
    (duration-board-disembark))
```

Figure 2.4: Function variables of the Driverlog domain.

6. Has a finite set of operators (action schemas) representing operations that can be performed, e.g., `drive-vehicle` shown in Figure 2.5.

The operator consists of the operator name, a list of all the parameters used in the operator, a set of condition that needs to hold at start, at end or overall of the action, and a set of effects that takes place at start or at end of the action. Instantiating operators with the problem objects generates the set of actions.

```
(:durative−action drive-vehicle
    :parameters (?a − agent ?t − vehicle ?l1 ?l2 − location)
    :duration (= ?duration (duration-driving ?l1 ?l2))
    :condition(and
        (at start (at ?t ?l1))
        (at start (driving ?a ?t))
        (over all (driving ?a ?t))
        (at end (driving ?a ?t)))
    :effect(and
        (at start (not (at ?t ?l1)))
        (at end (at ?t ?l2))))
```

Figure 2.5: Durative operator drive-vehicle of the Driverlog domain.

The PDDL Problem

The planning problem represents a particular problem instance to be solved. The PDDL problem contains; a set of objects which are instances of the defined types in the domain; the initial state characterised by the known values of both the finite set of propositional state-facts and the finite set of numeric state variables; and a set of goals as instantiated predicates conditions that need to be satisfied by the plan. In addition, the PDDL problem may contain a metric to be optimised. PDDL uses the closed-world assumption, which means that fluents (aspects of the world that change over time) that are not known to be true are considered false. Figure 2.6 shows an example of a problem encoded in PDDL2.2 of the Driverlog domain.

```
(define (problem dlog-1)
(:domain driverlog)
(:objects
    driver0 - driver
    truck0 - truck
    package0 - package
    s0 s1 - location)
(:init (empty truck0) (at driver0 s0) (at package0 s0)
    (= (duration-walking s0 s1) 800) (at 5 (at truck0 s0))
    (= (duration-walking s1 s0) 800)
    (= (duration-driving s0 s1) 200)
    (= (duration-driving s1 s0) 200)
    (= (duration-load-unload package0) 20)
    (= (duration-board-disembark) 10))
(:goal (and (at package0 s1)))
(:metric minimize (total-time)))
```

Figure 2.6: Example of a problem encoded in PDDL2.2 of the Driverlog domain.

2.2.3 Planning Systems

After explaining the models and modelling languages typically used for planning, we move to the planning algorithms. Planning systems are classified according to the direction of the heuristic-based search algorithm they run while exploring the search space to provide plans. Planners can perform a forward search starting from the initial state towards the goal state (progression search), a backward search starting from the goal state towards the initial state (regression search), or a bidirectional search where forward and backward search are interleaved to find a solution.

There are several planning systems, many of which have participated in different IPCs. The Heuristic Search Planner (HSP) (Bonet et al. 2001) is one of the first state-based systems which uses domain-independent heuristic search. The heuristic function for solving a problem is obtained by relaxing the problem by ignoring all delete effects. A relaxed plan is solved when every goal proposition has been added by some action.

Like the HSP system, Hoffman presented Fast-Forward (FF) (Hoffmann and Nebel 2001), which relies on a forward state space search, using a heuristic that estimates goal distances by ignoring delete lists. FF identifies the potentially most promising successors and introduces the added goal deletion heuristic to inform the search about goal orderings.

LPG (Gerevini, Saetti, et al. 2003) is a domain-independent planner for temporal planning in domains specified with PDDL2.1 that was awarded for outstanding performance of the first order at the 3rd IPC. LPG supports durative actions and numerical quantities and is based on a stochastic local search method and on a graph-based representation called temporal action graphs, where action nodes are marked to estimate the earliest time when the corresponding action terminates. In contrast, fact nodes are marked to estimate the earliest time when the corresponding fact becomes true.

LPG-td (Gerevini, Saetti, et al. 2006) is a prize winner in the temporal track of the 4th IPC in domains involving "Timed Initial Literals" and in plan quality (satisficing planning track). LPG-td is a new version of LPG that extends and improves its previous version to handle PDDL2.2 TILs and derived predicates.

A temporally-disjunctive action graph represents the causal structure of the plan (TDA-graph). Action ordering and scheduling constraints are managed by efficient constraint-based reasoning, and the plan search is based on a stochastic local search procedure.

Fast Downward (FD) (Helmert 2006) is a classical progression planner based on the heuristic search that has proven remarkably successful and won the classical track of the 4th IPC. FD can deal with general deterministic planning problems encoded in the propositional fragment of PDDL2.2. It uses hierarchical decompositions of planning tasks for computing its causal graph heuristic function. The heuristic evaluator proceeds starting from top-level goals, and the algorithm recurses further and further down the causal graph until all remaining subproblems are basic graph search tasks.

Due to its performance, FD has inspired several other planners, such as LAMA (Richter et al. 2010), which showed the best performance among all planners in the sequential satisficing track of the 6th IPC. LAMA uses a temporal landmarks heuristic of propositions that must be true in every solution of a planning task.

Partial-Order Planning (POP) is another paradigm in planning based on the least commitment principle (Weld 1994) to represent plans as a partially ordered sequence that enables deferring decisions about the order of the actions and the value of the variables as long as possible during the search, instead of committing prematurely to a complete totally ordered sequence of actions. Originally POP started as regression and moved to progression, which allowed combining partial-order reasoning with an explicit representation of states and using state-based heuristic functions. Examples of temporal POP planners are OPTIC (Benton et al. 2012), SGPlan (Chen, Wah, et al. 2006), and FLAP (Sapena, Onaindia, and Torreño 2015).

Optimising Preferences and Time-Dependent Costs (OPTIC) (Benton et al. 2012) is a temporal planner for problems where preferences or time-dependent goal-collection costs determine the plan cost. OPTIC stands out for the high speed of building successor states and the domain-independent heuristics it employs in many planning domains. At each state, OPTIC records what actions achieve, delete and depend on each fact, to apply time constraints only if necessary to ensure that preconditions are met. Additionally, to reduce search overhead, parameters are bound, and some ordering constraints are committed eagerly.

Hierarchical planning involves decomposing a task into more minor tasks or activities at every lower level; hence the computational cost of finding the best arrangement is less (Tate 1976). Instead of finding a path to a goal state, a Hierarchical Task Network (HTN) planner produces a sequence of actions that perform some activity or task. The Planning operators (actions) and methods prescribe how to break down a task into its subtasks, and a partially-ordered set of tasks to be accomplished exists instead of a goal formula. During planning, tasks are recursively decomposed into subtasks until primitive tasks can be performed directly using planning operators. The SHOP2 planner (Nau et al. 2003), developed at the University of Maryland and further enhanced into SHOP3 (Goldman and Kuter 2019), is considered one of the most mature and comprehensive HTN implementations.

In this PhD thesis, we use LPG-td and OPTIC. They are chosen because they can both handle PDDL2.2 and OPTIC can handle PDDL3.0.

Through the organisation of meetings, including the annual International Conference on Automated Planning and Scheduling (ICAPS) conference, since its creation in 1998, the IPC has strived to advance automated planning and scheduling, assess state of the art in the field, and coordinate new challenging benchmarks. As part of the evaluation of planning approaches, the validity of the plans is checked using VAL: The Plan Validator from the IPC (Howey et al. 2004b). VAL is an automatic validation tool For PDDL, including PDDL3 and PDDL+, for parsing and type-checking domains, problems, and plans before confirming that plans are valid. Furthermore, the plans are evaluated according to three metrics: coverage, quality, and time spent on generation. Coverage refers to the number of problems solved per domain. Plan quality refers to the length of the plan or the sum of the costs of all actions.

Planning deals with the task of generating plans to be later executed. The execution system applies the plan and also monitors the new states. The following section will discuss plan execution and its vital role in successfully applying planning systems in real-world environments.

2.3 Plan Execution Monitoring

While state-of-the-art planners are very effective, they focus mostly on closed environments. Several thorny issues arise when implemented in real-world open environments. Particularly, the environment's partial observability, non-determinism and unexpected changes can affect the system's success. Three trivial approaches for making closed-world planners adapt to the dynamic environment (Talamadupula, Benton, et al. 2010) are; assuming that the world is closed, attempting to close the world by obtaining all the missing knowledge before planning or incorporating all contingencies before planning. The first will result in frequent replanning, the second requires sensing sweeps and is infeasible, and the third requires the designers to anticipate and design every possible contingency.

The execution of a plan allows its actions to be carried out. It plays a crucial role in the effectiveness of planning systems since it determines whether the execution is going smoothly and how to respond if it is not. Discrepancies indicate anomalies or differences between what the agent expects and what it experiences. Agents apply discrepancy detection during execution monitoring as a first step for adapting successfully to dynamic environments. Discrepancy detection during plan execution is a crucial mechanism by which an agent knows something unexpected has occurred.

In the literature, execution monitoring and discrepancy detection often go hand in hand with failure detection, replanning, goal generation and sometimes opportunities. First, we will describe the main approaches to monitoring.

Russel et al. (2020) describe three approaches for monitoring during plan execution (Russell et al. 2020):

1. Action monitoring, where the agent lazily verifies that all action conditions hold only immediately before the action is executed.
2. Plan monitoring, where the agent verifies before executing an action that the remaining plan will succeed.
3. Goal monitoring, where the agent checks if it could achieve a better set of goals before executing an action.

The survey (Vattam et al. 2013) describes five approaches for monitoring:

1. Plan monitoring: the agent monitors the execution of the plan by assessing whether its remaining actions' preconditions have been met in the current state or achieved as a result of a previously planned action.
2. Periodic monitoring: the agent may monitor the entire environment rather than focus exclusively on the current plan and execution.

3. Expectation monitoring: the agent utilises past experiences to build models or expectations that can be used to monitor certain aspects of the environment.

4. Domain-specific monitoring: the agent implements domain-specific monitoring strategies, which periodically test the values of specified state variables during the execution of a plan when it is difficult to predict future states.

5. Object-based monitoring: Object-based monitoring systems specify which types of new objects should be considered as discrepancies.

2.4 Survival of the Agent, Failures and Opportunities

Table 2.1 lists several works from the literature and specifies their classification from a monitoring perspective and the purpose of each work. Follows an overview of these approaches and how they build on discrepancy detection to either avoid, detect or predict a failure, repair or replan after a failure, generate a goal or seize an opportunity.

Based on hard-coded expectations for the agent, Rao and Georgeff (1995) describe an air traffic management system in which discrepancy detection is performed as periodic domain-specific monitoring by sampling the environment to reassess its current course of action. The actions selection function is re-applied after observed events (similar to replanning) to adjust the goals in response to real-time changes (Rao et al. 1995).

The agent in (Norman et al. 1995) performs periodic domain-specific monitoring by observing the environment variables that may cause an alternative goal to be generated and periodically evaluates a function that, if it exceeds a threshold,

Table 2.1: Approaches to Monitoring

Papers	Monitoring Approach	Purpose
(Rao et al. 1995)	Periodic	Replanning
(Norman et al. 1995)	Periodic	Goal generation
(Cox 2007)	Domain-specific	Goal generation
(Dannenhauer and Muñoz-Avila 2013), (Dannenhauer and Muñoz-Avila 2015)	Domain-specific	Goal generation
(Ayan et al. 2007)	Plan monitoring	Replanning
(Klenk et al. 2013), (Wilson et al. 2013), (Powell et al. 2011)	Plan monitoring	Goal generation
(Fox, Gerevini, et al. 2006)	Plan monitoring	Plan repair
(Hanheide et al. 2010), (Hawes et al. 2011)	Object based	Goal generation
(Coddington et al. 2005)	Object based	Goal generation
(Talamadupula, Benton, et al. 2010)	Object based	Opportunistic
(Cashmore et al. 2018)	Object based	Opportunistic
(Reiterer et al. 2017)	Object based	Opportunistic
(Borrajo et al. 2021)	Plan monitoring	Opportunistic

the generated goal gets activated. So it avoids reaching a state that suffers a failure to operate continuously and independently.

The initial introspective cognitive agent INTRO (Cox 2007) for story understanding uses domain-specific expectation monitoring to determine its own goals by interpreting and explaining unusual events. If something anomalous is detected, the system will explain it and integrate it into the current understanding of the story. INTRO generates two types of goals: reducing dissonance between what is expected and what is observed and reducing the likelihood of repeating the reasoning failure.

In (Dannenhauer and Muñoz-Avila 2013), the authors presented LUiGi, an agent that generates goals in the Starcraft strategy game in order to operate continuously and independently. Using domain-specific expectation monitoring, the failure is detected when inferred facts are not consistent with the expectations of current actions. The agent is instructed to pursue a goal based on explaining the inconsistency and rules that specify which goals should be pursued under what circumstances. The approach was further enhanced in LUiGi-H (Dannenhauer and Muñoz-Avila 2015); it became (1) hierarchical to reason with expectations, discrepancies and explanations at varying levels of abstraction and (2) case-based by referring to a predefined library of plans annotated with expectations.

In the context of simulated noncombatant evacuation operations for dynamic environments, the hierarchical ordered task replanning system HotRide (Ayan et al. 2007) employs plan monitoring, specifically, action monitoring. When an action fails, a dependency graph identifies which task decompositions are no longer valid and must be replanned.

ARTUE (Klenk et al. 2013), M-ARTUE (Wilson et al. 2013) and T-ARTUE (Powell et al. 2011) are three systems that generate alternative goals to avoid failures for strategy simulation relying on plan monitoring. In particular, they monitor immediate expectations to identify discrepancies using a set difference operation between the expected and observed states. ARTUE uses manually constructed rule-based goal creation. M-ARTUE utilises domain-independent heuristic goal creation. T-ARTUE learns goal creation interactively from humans through queries, criticism and case-based.

The study by (Fox, Gerevini, et al. 2006) employs plan monitoring to identify discrepancies between the expected value of the state and its observed value arising from uncontrollable characteristics of the domain. The work involves repairing a plan after a failure while preserving the original plan to produce a stable repaired plan.

Dora the explorer (Hanheide et al. 2010; Hawes et al. 2011) is a framework for goal generation and management. To detect discrepancies, Dora deploys object-based domain-specific monitoring techniques to detect faults or gaps in the system knowledge base where the discovery of new objects may interrupt the execution of the plan and create new goals. A number of active processes in Dora keep track of the system's state (both the external world and the internal representations). The system has encoded motivators for generating goals relating to exploring newly detected rooms and determining their functions.

The Madbot (Coddington et al. 2005) is a motivated and goal-directed robot that uses object-based domain-specific monitoring that permits new goals to be formed at any time. If a new goal is generated, it is passed to the planner, and the planner subsequently replans, taking into account both the remaining current plan and the new goal.

As demonstrated in the studies in this section, arguably the most important part of the planning systems is the execution monitoring component which detects anomalies and then triggers replanning. Replanning is widely used to recover from a failure or respond to dynamic environment changes in real time after generating a goal, thus allowing the agent to function continuously and independently. Responding to opportunities, on the other hand, requires a different approach. The following studies are noteworthy since they emphasise the importance of seizing opportunities.

The opportunistic planning approach in (Talamadupula, Benton, et al. 2010) for urban search and rescue operations generates plans to locate objects (the victims) that are unknown before execution. The discovery of objects triggers goal generation and replanning. The system uses object-based domain-independent monitoring as new objects and facts may be brought to light through either external sources like the mission commander or action execution. The system incorporates knowledge about how new objects may be detected and uses the notion of open-world quantified goals to specify rewards for open sets of objects.

Using opportunistic planning, the authors in (Cashmore et al. 2018) employ object-based domain-independent monitoring to monitor instances of objects of a particular type that may exist at random locations and offer high rewards if certain operations are performed. The authors' approach is domain-independent and includes an example application about executing a plan by an autonomous underwater vehicle (AUV). The vehicle may experience an unexpected event during a mission, such as detecting partially submerged anchor chains or other structures; therefore, it can perform unplanned inspections or chain-following activities to enhance the utility of the operation. Opportunity goals do not require the planner to incorporate probabilities. They may be described as dynamically occurring soft goals that can be managed without modelling or anticipating them. An opportunistic goal is generated when a new object is found, and replanning is performed. By adopting conservative planning methods, the authors exploit classical planning techniques while simultaneously considering the unseen stochastic nature of the execution environment so that well-researched methods in temporal-metric planning may be utilised.

The work of (Reiterer et al. 2017) illustrates another opportunistic planning approach with robot safety recovery. Despite the domain independence of object-based monitoring, the opportunistic approach may require tailoring for different domains. The authors use opportunity templates (OT) to originate from PDDL actions and add a plan fragment consisting of parametrised actions that can have parameters matching the parameters of the OT and are therefore assigned values during the decision process. As part of a safe fallback plan, pessimistic assumptions are used for all safety-related variables to ensure that only safe actions are included in the plan. In execution, the main plan is checked against the available OTs according to a series of semantic steps tailored to this domain (other OTs may require considering different semantics). The semantic steps encompass checking that the opportunity conditions hold in the current state, checking whether effects break any conditions for the actions of the main plan, estimating the recovery period, the recovery policy with time and resources, and evaluating the utility against a cost function). Failure of any semantic step causes the opportunity candidate to be dropped. Recovery from failed opportunities involves enriching each candidate with a recovery policy that maps each opportunity state to a recovery plan that leads to the main plan; however, no evaluation or validation has been shown.

Planners focus on the given initial state; hence, they do not consider alternative scenarios. However, during the execution in real-world dynamic environments, finding an object that was not present during planning can make executing the existing plan non-rational. Such as when a series of actions of the existing plan is to obtain a key, and a spare key is found during execution. A more recent and different take on opportunities is presented in (Borrajo et al. 2021) for computing opportunities to augment plans for novel replanning during execution. The authors differentiate replanning from opportunities (replanning-O) from typical replanning from failures (replanning-F). A plan-based opportunity o is a static condition of action a not in the plan that achieves an effect that is also achieved by some other action a_0 in the plan. So, if o is true, the planner could use a instead of a_0 in the plan. The authors utilise plan monitoring (goals-regression expectations), and even more importantly, they propagate opportunities back through the plan. Monitoring such opportunity facts does not result in a goal, nor is there is any failure in the plan, but rather, if the opportunity is captured and a new plan is generated, it will result in more intelligent behaviour than that of the old plan in light of the new environment change.

We have reviewed the related state-of-art in planning and execution monitoring and how it enables the agent to handle failures, generate goals and seize opportunities. In the next section, we move to KR and its role in autonomous agents.

2.5 Knowledge Representation

In light of the adaptive nature of human intelligence and as envisioned in (Berners-Lee et al. 2001), researchers from both industry and academia investigate the development of autonomous agents that can work in harmony, coordinate searches and results retrieval and significantly reduce users' efforts. Focusing on developing systems that incorporate knowledge to reason based on that knowledge, solve problems and mimic human behaviour. An explicit and symbolic representation of domain knowledge characterises most such systems, including those reviewed in the previous sections. Such symbolic representation of knowledge has the advantage of being decoupled from the procedural aspects of its application, and it can, therefore, be reused across multiple agents. A fundamental rationale and role for such representation are that it is a surrogate for the things that exist in the world and is used for reasoning (Davis et al. 1993).

In its narrow view definition, KR is an area of AI that is concerned explicitly with representing information about the world in a format that agents can use to solve complex tasks. However, any AI system may be called a knowledge-based system when it has the following: (1) declarative knowledge; a knowledge base that might include facts, rules, representation formalisms, and different types of knowledge, and (2) procedural knowledge; an inference algorithm that might be a deduction, non-monotonic or probabilistic reasoning, abduction, or even planning.

2.5.1 Knowledge Representation Forms

The KR forms are divided into four categories (Patel et al. 2018): network-based representation (semantic network), structure-based representation (frame-based), production rule representation, and logical representation.

In network-based representation, knowledge is structured into semantic groups to provide a natural means of mapping knowledge between natural language and these networks. Capturing the semantics graphically as a directed or undirected graph consisting of vertices representing concepts (with no distinction between individuals and classes) and edges representing semantic relations between concepts (Deliyanni et al. 1979).

Derived from semantic networks, the frame-based representation was coined by Marvin Minsky in 1975 to represent stereotypical situations. However, the emphasis does not lie on vertices but on instances or classes and slots (properties) and their fillers (values) as opposed to links and connections, in addition to inheritance through which values in frames are transferred from classes to instances (Minsky 2019).

Emil Post invented production systems that process strings using rules to create new strings (Mol 2006). Production rule-based representation represents knowledge in terms of multiple rules (productions) that specify what should be concluded in different situations. Productions consist of two parts: a precondition (or "IF" statement) to be matched and an action (or "THEN"). This type of representation provides modularity and explainability and is similar to human cognition.

In a logical representation, logic is used to formalise the facts representing the world, establish unambiguous semantics, and for computational purposes. Under the logic-based representation category, there are several types of logic, such as predicate logic, Description Logic (DL), and proposition logic. As its name suggests, DL is aimed to describe the important notions of a domain using concept descriptions (expressions built from atomic concepts, unary predicates, and atomic roles), and to overcome the deficiency encountered in frames and semantic networks as they lack formal logic-based semantics (Baader et al. 2008).

The semantic web, in essence, refers to the field of research that focuses on creating and maintaining the semantic web (as an artefact), as well as all the tools and methodologies necessary to accomplish these tasks and for application (Hitzler 2021). The semantic web artefact perspective can be traced back to 2001 and is envisioned as an enhancement of the current web, where information is given well-defined meaning so that it can be shared and reused across computer systems, enabling people and computers to cooperate better (Berners-Lee et al. 2001). Natural languages are not suitable for creating models that can be understood by computers and software agents, since they are too ambiguous; therefore, formal machine-interpretable metadata languages are used to store and retrieve resources in the semantic web (Maedche et al. 2001). Several languages were created to encode the metadata, all based on the World Wide Web Consortium (W3C) specifications, such as the Resource Description Framework (RDF) (Klyne 2004) that was adopted as a W3C recommendation in 1999 and Web Ontology Language (OWL) (Horrocks et al. 2003).

2.5.2 Ontologies

For most of the 2000s, work in the field had the notion of ontology at its centre, acting as the backbone of the semantic web; reusable ontologies serve as a foundation to integrate, share, and discover knowledge (Hitzler 2021). Perhaps the most quoted definition in the literature and the ontology community is Thomas Gruber's definition; "An ontology is an explicit specification of a conceptualisation" (Gruber 1995).

Table 2.2 summarises the history of ontologies as described in (Gómez-Pérez et al. 2004), coinciding with the recent review study (Machado et al. 2020), starting as philosophies of ancient Greek and reaching today's term in the context of information systems, which is focused on concrete problems of modelling domains of knowledge in computational artefacts (Poli et al. 2010).

Agents need knowledge about the world to reach sound decisions and act intelligently. In order to establish an agreement on the meaning of terms, an ontology language is employed to conceptualise the domain while separating knowledge from the procedure. This facilitates the collection and sharing of knowledge by multiple agents. The terminology may vary depending on the ontology language; generally, an ontology consists of classes to represent concepts, individuals who are particular instance entities belonging to specific concepts, and properties to describe the possible relations between individuals and axioms to state what is true in an ontology.

Table 2.2: Points in the History of Ontology

Time	Description
Ancient Greek	Philosophies of Being and the differentiating between the essence and the existence.
Middle ages	Universals and symbolic paradigm (William of Ockam).
Modern age	Copernican turn, the essence is determined not only by the thing itself but also by the contribution of whoever perceives and understands them. Emmanuel Kant (1724-1804)
Present	An important research in computer science as the systematic, formal, axiomatic ontological engineering, where an ontology consists of a generalisation/specialisation hierarchy of concepts and relationships meant to describe facts about the real world (Cocchiarella 2007; Guarino et al. 2009)

2.5.3 Ontologies and Autonomous Agents

Several rich review articles discuss in-depth literature about knowledge and ontologies for intelligent agents, such as (Thosar et al. 2018; Paulius et al. 2018; Alarcos et al. 2019). This subsection overviews relevant research on ontologies for autonomous agent applications and how ontologies support autonomy.

In the work of (Bouguerra et al. 2007), semantic domain knowledge is applied to interleave planning and execution to validate the effects of actions (information gathering, planning, and execution). Expectations of the current location are derived where unpredictable and complex situations are handled during run-time. Based on DL, a language with ontology-like syntax is used to define concepts, relations and assertions, specify constraints on relations and concepts and express assertions about individual objects.

Using the SemantiCore framework (Blois et al. 2007), developers can build semantic applications without having to deal with all the implementation details involved. SemantiCore uses an OWL ontological representation that maps all the elements an agent has, including concepts and instances, providing rich knowledge that can be used to reason about itself and other agents in the domain.

In LUiGi (Dannenhauer and Muñoz-Avila 2013), an agent with ontological representations using OWL formalism maintains an ontology of both low-level facts about the environment as well as high-level concepts that can be derived from the low-level facts. Using the ontology, Starcraft maps' regions can be classified as controlled, contested, or unknown. At any given moment, the ontology represents the state of the game, and the objective is to defeat the opponent using pre-designed strategies. The ontology is used to derive rich expectations during execution monitoring and to generate explanations of anomalies.

KnowRob 2.0 (Beetz et al. 2018) is an advanced KR system for robots designed to make decisions about parameterising motions to achieve manipulation duties. A unique feature of this framework is that its data structures, representations, and parameters are driven by the associated axiomatisations of the ontology encoded in OWL, allowing the robot control system to use the ontology as a symbolic knowledge base. KnowRob ontology defines robots, objects, tasks, situational context and environment and is critical to grounding its concepts in perception, reasoning, and control. The robotic agent can infer knowledge about entities, like dirty dishes belonging in the dishwasher and perishables in the refrigerator.

An OWL ontology is adopted as the basis for declarative KR of an intelligent system, named the skill server (Haage et al. 2011), capable of supporting automatic and semi-automatic reconfiguration of manufacturing processes. The KR of the ontology is centred around the concepts of devices and skills as a static part, whereas tasks can be seen as combinations of skills and reasoning is used about skills matching particular tasks.

The Perception and Manipulation Knowledge (PMK) ontological-based reasoning framework (Diab, Akbari, et al. 2019) acts as a knowledge base to enhance task and motion planning (TAMP) in autonomous robots. It reasons for perception, object features, the situation, and planning. The ontology was implemented using OWL. For perception, a sensing class represents knowledge of sensors, measurement processes, and their relation with the robot. For planning KR, symbolic tasks are defined in level-three (Tasks), composed of level-two simpler actions (Sub-Tasks), and level-one includes primitive actions, preconditions, conditions, and atomic functions.

The work of (Bruno et al. 2019) seeks to support active and healthy ageing and reduce caregiver burden by developing culturally sensitive and competent elder care robots. Knowledge is primarily represented by ontologies using OWL 2 formalism (Group et al. 2009) to enhance robot task planning and execution and human-robot interaction. In the cultural knowledge base ontology, the TBox consists of classes and properties (data properties relating instances of a class to literal data and object properties relating instances of a class to other instances). In contrast, instances of classes and properties are stored in the ABox. The relationship between the TBox and the ABox of the ontology consists of culture-generic knowledge (TBox - I) for all cultures of the world, national culture-specific settings (CS-ABox - II), Person-specific settings (PS-ABox - III), and an assessment and adaptation algorithm for the discovery of person-specific settings.

2.5.4 Ontologies and Planning

Knowledge enhances planning and robot capabilities, giving more autonomy to the robots in solving challenging problems. The work of (Bermejo-Alonso 2018) presents a review of existing tasks and planning vocabularies, taxonomies and ontologies as a necessary first step in an ontology engineering process that addresses the autonomous system planning needs. Typically ontologies are used with automated planning, primarily not to perform planning as scaleable and efficient AI planners do per se but rather to enrich, facilitate some planning decisions and extend the planning and execution experience. Most notably, utilising ontologies with planning shines in formalising the knowledge of a specific domain, such as in (Teixeira et al. 2021), detecting discrepancies and filling knowledge gaps, such as in Dora the explorer (Hanheide et al. 2010; Hawes et al. 2011), failure explanation and recovery such as in LUiGi (Dannenhauer and Muñoz-Avila 2013; Diab, Pomarlan, et al. 2019), and finding missing connections (conditions) of actions in restricted-size problems (Beßler et al. 2018). This subsection presents several notable works from the literature on using knowledge with planning.

For assembly manipulation tasks, the authors of the work (Diab, Pomarlan, et al. 2019) describe their ontological formulation that provides a common understanding for the robot to interpret the causes of the failures in automated planning and execution. By querying their proposed ontology, the agent can analyse the cause by interpreting the failure and deciding on the following action or generating intermediary goals. Their technique for static analysis works if the tasks are appropriately characterised in detail, each task concept must have sufficient axioms to define it, as well as object concepts restricting what may participate in a task. The reasoning process can only be done within some models of the world. The reasoner would be required to find a model for an example event to find which failure symptom is possible for each task. Queries would be run for every pair of named task and failure symptom concepts in an ontology, as failure recovery is essential for automated planning and execution.

Dialogue trees are time-consuming to build when dealing with complex health issues. In addition to domain knowledge, this requires expertise in dialogue modelling and AI planning, which is not frequent among dialogue authors, limiting the adoption of such a powerful approach. Hence, AI planning can benefit from formalising the knowledge of a specific domain using ontologies to build an efficient policy (a plan). The work of (Teixeira et al. 2021) proposes an early-stage approach for domains with limited complexity to automate the generation of a dialogue manager capable of handling goal-oriented dialogues for the health domain. Their system integrates a conversational ontology (Convology) and AI planning. Where their dialogue manager uses questions and obtains answers from the patients filling values in the ontology, a translation is done to a PDDL domain and a problem so that a state-of-the-art (domain-independent) planner can compile it.

A logic-based planner for assembly tasks is combined with geometric reasoning capabilities to enable robots to perform their tasks under spatial constraints in cluttered environments (Beßler et al. 2018). The geometric reasoner is integrated into logic-based reasoning by using decision procedures associated with symbols in several ontologies (assembly, action, and planning ontologies). In the planning process, individuals in the knowledge base are compared to their terminological model, inconsistencies are found, and action items are developed to resolve them according to OWL entailment rules; this means identifying what information is missing or false about an individual. For example, if the asserted type of an entity is a car, the planner will create action items to add wheels if such an entity does not have them. Knowledge-based reasoning determines whether an individual fulfils the criteria for membership in the classes to which it belongs, identifies individuals based on their relationships with others, and identifies a set of individuals linked by specific properties. RDF triple stores are queried to determine whether appropriate triple assertions have been made or can be inferred. The system is domain-independent and specifically targets assembly problems. An evaluation was performed in a toy plane assembly targeted at 4-year-old children with 21 parts; however, no performance metrics were presented, and scalability was not discussed for complex assembly tasks or problem domains besides assembly.

LUiGi-H (Dannenhauer and Muñoz-Avila 2015) is a goal-driven autonomy agent that uses an approach that is (1) hierarchical and ontological to reason with expectations, discrepancies and explanations at varying levels of abstraction and (2) case-based by referring to a predefined library of plans annotated with expectations. One of the main benefits of their ontology usage is that it uses facts to represent atoms, which can be initial or inferred facts. In addition, it allows the representation of high-level concepts. Therefore expectations can also be primitive or compound expectations (that are inferred). Explanations are directly linked to an expectation. For primitive expectations, the explanation is simply the negation of the expectation when that expectation is not met. For compound expectations (e.g. expectations that are the consequences of rules or facts that are inferred from description logic axioms), the explanation is the trace of the facts that lead up to the relevant rules and axioms that cause the inconsistency. Using an ontology, semantic annotations via inferred facts enable LUiGi-H to reason at different levels of abstraction, providing explainability and flexibility to do local repairs.

2.6 Additional Prevalent Challenges in State-of-the-Art and the Thesis' Responsive Strategies

In this chapter, the author reviewed a range of related works, identifying their strengths and research gaps. Each problem is then addressed in the following chapters, tying the identified limitations in current methods directly to the specific objectives of this thesis. The ensuing subsections present an illuminating discussion, describing additional prevalent challenges inherent to state-of-the-art approaches. Simultaneously, an overarching perspective of the author's research efforts to mitigate these challenges is presented.

2.6.1 Additional Prevalent Challenges from the State-of-the-Art

1. Interfacing Ontology and Planning is a challenging task. As (Bouguerra et al. 2007) and (Bermejo-Alonso 2018) demonstrated, ontologies can provide rich representations of the world. However, interfacing this with planning algorithms to make actionable decisions is challenging, especially in dynamic, unpredictable situations or high abstraction.

2. Adapting to Complex Situations: A related challenge is adaptability to diverse contexts. As demonstrated by LUiGi's ontology (Dannenhauer and Muñoz-Avila 2013), it can perform well in the specific environment it was designed for. Still, its performance in different, more complex, or dynamic environments is challenging and yet to be tested.

3. Contextual Reasoning and Adaptability: Building adaptable systems that can reason about different kinds of processes is a challenge, as noted in the skill server model proposed by (Haage et al. 2011). The model works

well for manufacturing processes, but its adaptability to diverse contexts or different processes is challenging.

4. Cultural Sensitivity in Interaction: Another challenge in the field is incorporating cultural nuances into autonomous systems. As noted by (Bruno et al. 2019), building culturally sensitive and competent elder care robots is challenging because of the complexity and subtlety of culture and behavior.

5. Scalability and Efficiency of Reasoning and Planning: Lastly, the works of (Diab, Pomarlan, et al. 2019), and (Beßler et al. 2018) illustrate the challenge of scalability and efficiency. As the complexity of tasks or the size of the problem domain increases, the efficiency of the ontological and planning systems might decrease, and their ability to scale effectively is not fully explored.

2.6.2 The Author's Approach to the Challenges

This subsection briefly outlines the overarching strategies for the identified additional prevalent challenges. The original contributions of this work are designed to navigate the obstacles facing the field with a strong focus on building a context-aware deliberative framework and enhancing the autonomy of intelligent agents to seize opportunities or repair failures. The strategies proposed range from innovating the interfacing between ontology and planning, to improving adaptability to complex situations, strengthening contextual adaptability, acknowledging the importance of cultural sensitivity in interaction, and recognising scalability and efficiency.

1. Interfacing Ontology and Planning: Inspired by the goal to boost the autonomy of intelligent agents, the thesis introduces a context-aware

deliberative framework in Chapter 3, detailed in Chapters 4, 5 and 6. This framework, exemplified in the tourist assistant system in Appendix A, unites ontology and planning in Appendix B, facilitating a seamless integration of ontological knowledge into planning strategies. Incorporating two distinct distance calculations adds another layer of dynamism, enabling the integration of new objects into the planning process. Additionally, new object integration using ontologies from Appendix C further exploits ontologies for planning applications.

2. Adapting to Complex Situations: The thesis presents a multi-pronged domain-independent approach to enhancing the adaptability of autonomous agents. The system is equipped to handle complex scenarios with methodologies like the plan commitment property, neighborhood constraint method, and testing in various dynamic environments (Chapters 3 to 6). Appendices B and C illustrate this adaptability through the recognition and incorporation of new objects and adjustments to users' emotional states.

3. Contextual Reasoning and Adaptability: The context-aware deliberative framework (Chapter 3 and subsequent chapters) empowers the system to dynamically recognise and incorporate new object types and new objects, allowing the generation of new goals or repair the failures to reach the existing goals through the use of new objects possibly of a new type. This adaptability is showcased in the Appendices, where the system modifies its responses based on the changing contexts, whether new physical objects, user goals, or emotional states.

4. Cultural Sensitivity in Interaction: While not a focus of the research, the built-in adaptability and context-awareness of the system suggest potential for cultural sensitivity in interaction. The emotion recognition

capability introduced in Appendix C allows for consideration of cultural nuances, as cultural norms can heavily influence emotional responses.

5. Scalability and Efficiency: The thesis focuses on the domain-independent design of the simulator, ontological alignment, ontological distances, and commitment repair mechanism. However, it is essential to note that this work's primary goal is not explicitly demonstrating the system's scalability. The system has been designed with an orientation towards efficiency and has shown promising performance in the tested problems. While full-scale testing and verification yet remain an avenue for future studies, the inherent design principles suggest the potential for scalability.

In conclusion, this thesis endeavours to leverage various deliberative functionalities of autonomous agents. It does so by bridging the gap between ontology and planning, thereby facilitating the development of an adaptable, context-aware autonomous system capable of efficiently operating within dynamic environments with a higher degree of autonomy than its predecessors. This system seizes opportunities and autonomously repairs failures, thus striving to advance the field of autonomous systems.

Now that the author has presented the state-of-the-art related to the research lines of this PhD thesis, the next part of this PhD thesis is concerned with the authors' contributions.

Part II

Contributions

CHAPTER 3

System Architecture

CONSIDERING the three contributions of this PhD thesis: designing a domain-independent monitoring and execution simulation system, designing a context-aware knowledge acquisition method to allow the agent to seize an opportunity or fix a failure, and designing a plan repairing method that provides a responsible repair policy among agents. It is only fitting that the focus of this chapter is fourfold:

1. Presenting the formalisation of the problem addressed in this PhD thesis by explaining our modelling of the: planning problem, the dynamic planning problem and the OWL ontology formalisms in Section 3.1.

2. Providing a preface describing the problem addressed in this PhD thesis in Section 3.2.

3. Providing an abstract description of the method to achieve the aims of this PhD thesis in Section 3.3.

4. Providing a sketch and an overview of the framework scheme modules in Section 3.4, followed by a brief description of its elements. On the other hand, the modules are explained in detail in Chapters 4, 5 and 6.

3.1 Problem Formalisation

It is conspicuous that the richness of the ontological representations and the planning representations are different since the requirements of each field are different. Ontologies have high expressivity to allow more detailed domain descriptions. On the other hand, planning requires limitations on expressivity to provide efficient planning. Forcing representational constraints to make these fields more compatible will not work. Ontologists will not be willing to give up the current richness of ontologies, and planners will not be willing to settle for less efficient planning systems. Perhaps this conflict of interest can be resolved most effectively, as suggested by (McNeill et al. 2005), by allowing different representations to exist simultaneously. Utilising the rich (OWL) representation and the efficient (PDDL) representation simultaneously offers two main advantages:

- Allowing the agent to utilise efficient planning techniques grants greater flexibility and autonomy for seizing an opportunity or recovering from failed execution.
- Allowing the agent to use the additional detailed description of the knowledge domain provided by ontologies for more context awareness.

In the following subsection, the author explains the formalisms of the planning and knowledge representations used in the contributions.

3.1.1 Planning Formalisation

Our approach primarily builds upon elements that make up PDDL 2.2 specifications (Edelkamp et al. 2004). Specifically, for a planning task, we will use the following components of a domain:

1. A finite set of types is defined in a reasonable hierarchy $T = \{t_1, t_2, \dots\}$, such that every object of a planning problem belongs to a single type, and this type is a leaf node of a type hierarchy.
2. A finite set of variables $V = \{v_1, v_2, \dots\}$ that are propositional (boolean) or numeric representing properties associated with a type or a relationship established among types in T. Each $v \in V$ is defined as $v = (name(v)\ arg(v))$; where the first element, $name(v)$, is the variable name and the second element, $arg(v)$, is the set of typed arguments.
3. A finite set of operators $OP = \{op_1, op_2, \dots\}$ (actions schemas) that are ungrounded PDDL durative actions. An operator $op \in OP$ is defined as $op = (head(op), dur(op), cond(op), eff(op))$, where:
   - $head(op) = (name(op)\ par(op))$ is the operator head defined by its name and the corresponding typed parameters list of types in T.
   - $dur(op_i)$ is the duration of the operator, represented by a numeric variable.
   - $cond(op_i) = cond(op_i)_{\vdash} \cup cond(op_i)_{\leftrightarrow} \cup cond(op_i)_{\dashv}$ is the set of conditions of the operator op_i that are required at different time instances through its duration, where $\vdash$, $\leftrightarrow$,$\dashv$ denote at start, over all and at end, respectively.

- $eff(op_i) = eff(op_i)_{\vdash} \cup eff(op_i)_{\dashv}$, is the set of effects of the operator op_i that occur at start and at end.

On the other hand, the following components are used for the problem:

1. A finite set of objects O, each belonging to one of the types defined in T for the particular problem instance.
2. A finite set of state variables constituting the initial state I. State variables, also called fluents, are variables from V whose arguments are instantiated with objects from O and bound to a value, either boolean or numerical.
3. A finite set of TILs: state variables denoting time-stamped changes to the world as the information that is known to happen at a future time.
4. A finite set of hard goals G; each $g \in G$ is a fluent that the planner must achieve.

Hence for our purposes, a planning task is defined as $\phi = (\mathit{Dom}, \mathit{Prob})$. A domain is a tuple $\mathit{Dom} = (T, V, OP)$, and problem is a quadruple $\mathit{Prob} = (O, I, \mathit{TILs}, G)$. The problem may vary depending on the particular instance to solve. The domain encapsulates the behaviour dynamics as a fixed description of what the planner can do to achieve G, starting from what is known: I and $TILs$. The finite set of actions $A = \{a_1, a_2, \ldots\}$ are grounded from the operators OP using objects in O, modelling all possible actions.

Upon receiving the domain and the problem, a planner generates a temporal plan π as a collection of actions to satisfy G, where each action has a start time and a duration. During the execution of the plan, discrepancies between the expected state and the actual state of the environment may be detected, possibly making the execution of the rest of the plan impossible. At this point, we define a dynamic planning problem as a tuple $(G, \pi, O', I', \mathit{TILs}')$, where:

- π is the original plan, designed to achieve goals G according to the initial beliefs (I and TILs).
- (I', TILs') describe the new observed situation when a failure occurs.
- O is distinguished from O' as new objects may be detected during the execution of π.

We prohibit using conditional effects, derived predicates and numeric effects that are continuous, non-linear, or depend on actions' durations.

3.1.2 Knowledge Formalisation

Based on DL, OWL represents knowledge domains using ontologies. Our contributions build upon some elements that make up an OWL ontology (Horrocks et al. 2003). Specifically, for an ontology, we will use the following components:

1. A finite set of OWL classes (concepts) $\Omega = \{\omega_1, \omega_2, \dots\}$, which are constructs or templates representing the entities of the knowledge domain of the ontology. The classes are arranged in a taxonomy to make the knowledge hierarchically structured.
2. A finite set of individuals (instances) $\Lambda = \{\lambda_1, \lambda_2, \dots\}$, representing members of different classes.
3. A finite set of OWL object properties $\Psi = \{\psi_1, \psi_2 \dots\}$, defining the relationships that may exist between classes to specify how the individuals of these classes can link.

4. A finite set of OWL annotation properties $\Xi = \{\xi_1, \xi_2, \dots\}$, representing the annotations that can be used to add information (metadata) to classes, such as `rdfs:label [type:  xsd:string]` annotations.

Hence for our purposes, we define an ontology as $\eta = (\Omega, \Lambda, \Psi, \Xi)$. In addition, we make use of the following symbol ::, which refers to a subconcept (subclass). For instance, $\omega_i :: \omega_j$ means ω_j is a subconcept of ω_i within the same ontology.

3.2 Problem Description

> *"The human organism cannot survive as a bundle of neural reflexes, or even of stimulus-response learning pathways. In order to perform within the infinitely complex ecosystem to which it became adapted, it needed to establish* autonomy *from the genetically determined instructions that had shaped its behaviour through the long aeons of its evolution. The system that has evolved to provide this* autonomy *is the self. The function of the self is to mediate between the genetic instructions that manifest themselves as "instinctual drives" and the cultural instructions that appear as norms and rules. The self must prioritise between these various behavioural instructions and select among them the ones it wants to endorse."* (Csikszentmihalyi 1988, p. 17)

Human beings are endowed with the ability to adapt and cope with a plethora of challenges encountered in daily life. Even when unexpected events and change occurs, we can handle them intuitively and solve problems effectively. When faced with a challenge, we look around, think outside the box, and utilise all available means to find solutions. Moreover, we are civilised; we treat each other with sympathy and respect as we embark on life's journey towards reaching our goals. Similarly, we strive to increase our skills and capabilities to capitalise on opportunities when they arise and achieve more. In contrast, developing an intelligent agent who can adapt to unexpected events and unfamiliar circumstances in a dynamic environment remains challenging. An autonomous agent should strive to seize opportunities and handle failures without harming other agents (as the author strive to achieve in this PhD thesis).

AI has been ambitiously described in relation to human intelligence as machines capable of reasoning like a human or acting intelligently (Russell et al. 2020; Coppin 2004). Following that vision, the effort in this PhD thesis can be viewed as ambitious yet practical steps in that direction. As mentioned previously, we want to endow autonomous agents executing a plan in a dynamic environment with the ability to adapt and utilise their own perception of context, enabling them to provide context-aware deliberative responses for seizing an opportunity or repairing a failure without disrupting other agents that are working in the same execution environment. In order to achieve our aims, we had to highlight and tackle the following barriers:

1. The lack of a monitoring system that takes a general domain description, simulates the execution of the plan in a dynamic environment, monitors the execution of the plan and the execution environment in a lively fashion, handles unexpected live events, and in addition, monitors the goals.

2. The restricted sense of autonomy and context-awareness in autonomous agents, only anticipating opportunities or repairing failures pertaining to existing objects or new objects of existing types in the agent model and ignoring new objects of new types that could lead to a loss of an opportunity or failing to repair an execution failure.

3. The lack of an approach that takes it on its responsibility to responsibly repair the agent's plan's failures by generating a plan that commits to the original plan, as much as possible, to not disrupt others within a community of agents.

3.3 Method Description

The abstract diagram in Figure 3.1 outlines this PhD thesis's primary objectives to reach our aims as a context-aware deliberative framework to seize an opportunity or repair a failure. The diagram consists of three modules: the simulator module, the context-aware knowledge acquisition module and the committed plan repair module.

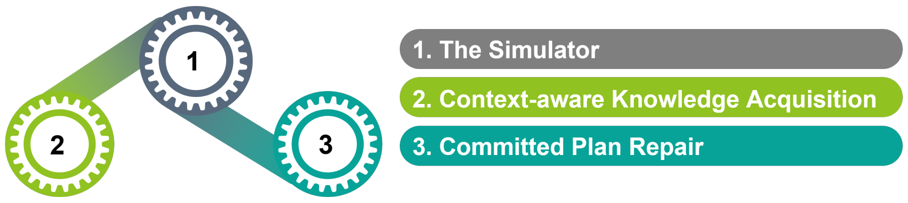


Figure 3.1: Context-aware Deliberation to Seize an Opportunity or Repair a Failure.

A brief description pertaining to each of the mentioned modules follows. Systems that provide a personalised plan (activities) to achieve specific goals are becoming more common thanks to IoT (Internet of Things) and open data platforms (Santiago et al. 2012; Mínguez et al. 2010; Sebastia et al. 2009; Vansteenwegen et al. 2011; Refanidis et al. 2014). It is common to add, remove, or rearrange activities; however, most of these systems use static information while the activities are being performed. Agents executing their plans in a dynamic environment are impacted by this shortcoming. Adapting and personalising activities in real-time is imperative (Neuhofer et al. 2015), even if they are pre-planned ahead of time. One essential prerequisite for intelligent technology is real-time synchronisation, which implies that information is not limited to an a-priori collection but can be collected and updated in real time. In order to create a flexible and agile plan that responds to the changing environment, it is necessary to monitor the plan and ensure that activities are taking place as planned, as well as to find a new plan if required. Although several monitoring and simulation frameworks can be customised for particular scenarios, they cannot take a general domain description and simulate its behaviour in highly dynamic environments. To overcome the previous limitations, we created the simulator (module 1 in Figure 3.1), which will be the backbone of the whole framework.

Whether to repair a failure or seize an opportunity, context and context awareness are crucial for any intelligent agent that operates in a dynamic environment. Currently, approaches for goal-directed behaviour (Cox 2007; Dannenhauer and Muñoz-Avila 2013; Klenk et al. 2013), online planning (Sapena and Onaindia 2002; Sapena and Onaindia 2008) and opportunistic planning (Cashmore et al. 2018) are used to address unanticipated changes related to objects or object types already covered in the planning task that is being solved. A genuinely autonomous intelligent agent must depend on its own percept without being programmed to handle every possible scenario (Russell et al. 2020) that could stem from encountering new objects unknown to the agent or object types unfamiliar to the agent. In addition to solving this problem, we must address an additional issue: the agent must reliably distinguish relevant and manageable new objects from irrelevant or unmanageable ones. Whether to repair a failure or seize an opportunity, we provide the agent with context-aware knowledge acquisition capability (module 2 in Figure 3.1). Therefore, allowing the agent to extend its planning knowledge with new information when encountering unforeseen circumstances if and only if the information relates to the execution environment and is solvable by the agent’s capabilities (action schemas). Therefore, being context-aware of the dynamic execution environment and the agent's task. In that regard, to maintain privacy, we assume that semi-cooperative agents would only share information related to types, relationships, and heads of operators that can be applied to these types, with no private details such as goals, states, or conditions effects of the action schemas.

In an ideal world, each agent is in charge of synthesising and executing its plan and performing replanning to handle changes in the state of the world. However, in the real world, there are many such agents, each with its own set of goals, but they are all linked by a common execution environment. While executing their plans in a dynamic environment, autonomous agents may encounter discrepancies between the expected and actual context and thus must replace their obsolete plans with new ones. While doing so and attempting to reach its original goals, an agent may unknowingly disrupt other agents. Therefore, the literature effort for dealing with multiple agents operating in the same execution environment is multi-facet. It includes agents' communication (Chopra et al. 2013), agents' privacy (Maliah et al. 2017), agents' negotiation and level of cooperation (such as altruistic and cooperative agents, lying and deceptive agents or self-interested agents) (Kraus 1997; Zlotkin et al. 1989; Shim et al. 2012), agents' various types of algorithms used for planning (such as central or distributed) (Nwana et al. 1997) and the metric to be optimised when replanning (such as the plan's makespan or execution safety with minimal perturbation) (Fox, Gerevini, et al. 2006; Kambhampati 1990). In this regard, we are concerned with ensuring a robust execution of an agent's plan in a community of agents so that it can be adapted to unforeseen situations arising from discrepancies between what an executing agent expects and what it observes. More importantly, ensuring that the new adapted plan does not propagate negative consequences to other agents operating in the same execution environment when a failure occurs. Our approach is independent of the way agents communicate and share information. To maintain privacy between agents and regulate their communications, we will assume that each agent has a set of private goals and an external entity tracks and keeps the relevant information of the environment in a repository where the agents can consult this information. The rest of the information is private unless an agent decides to share it with another agent as facts expected to occur in the future; we will model such facts as TILs.

3.4 Framework Overview

The proposed framework integrates the use of various deliberative functions, namely: planning, execution, monitoring, failure detection, knowledge augmentation, goal generation (formulation), replanning and plan repairing. Figure 3.2 shows the three main components of our context-aware deliberative framework. The framework takes a planning task (domain and problem) as input. The planner is external to the system and is either used to synthesise the original plan of actions or perform replanning. The framework is divided into an offline and an online part. The offline part has one module: the context-aware knowledge acquisition module. On the other hand, The online part consists of two modules; The simulator and the committed plan repair module. Here follows a description of each module of the architecture.

- Module 1 Simulator. The simulator is responsible for the following deliberative functions; perception, acting and monitoring and triggering goal generation, replanning or plan repairing.

  – Perception: provides our system with the critical capability of observing by sensing the IoT objects' values characterising the en-

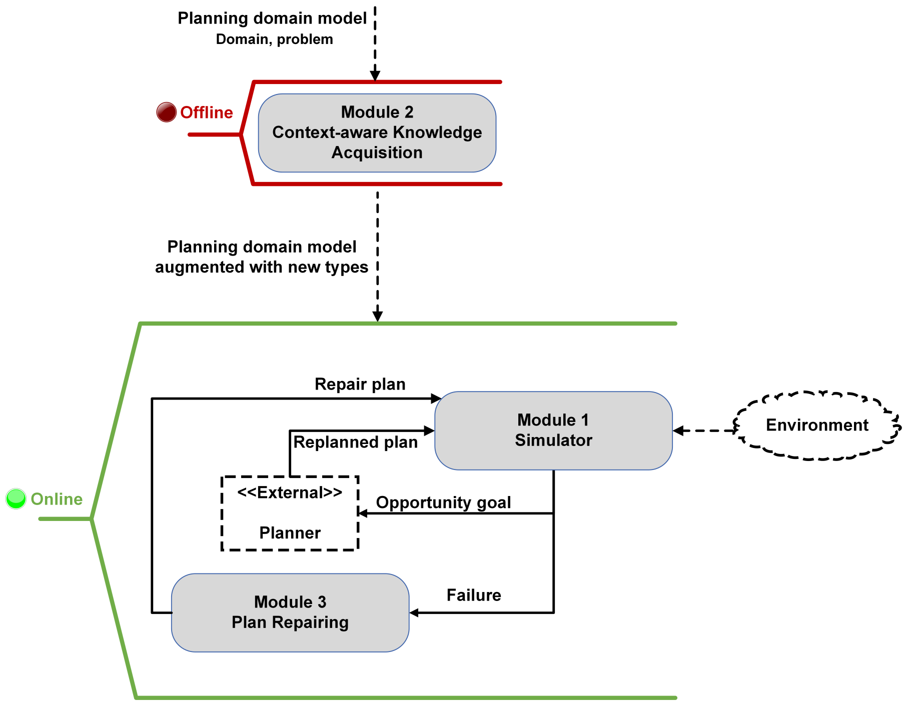


Figure 3.2: Context-aware Deliberative Framework Schema.

vironment's actual state. If an opportunity is spotted or a failure occurs, perception provides reactivity. During the plan's execution (Acting and Monitoring), the agent may encounter new objects of the relevant types not predefined in the model, integrates them into the planning problem, and uses them to repair failures or seize an opportunity.

– Acting and Monitoring: is responsible for executing the plan, monitoring actions, and detecting failures that may occur.

• Module 2 Context-aware Knowledge Acquisition. This offline cognitive knowledge augmentation module is initialised with a planning task. It creates an ontological representation of the planning domain and dynamically augments the predefined list of object types of the planning task with relevant new object types. This allows the agent to be context-aware of the environment and the task being performed and reason with incomplete knowledge. Consequently, boosting the system's autonomy and context awareness as the agent will be attentive (during execution simulation) to seize opportunities (goal generation) or fix failures (replanning using an external planner or plan repairing).

- Module 3 Plan Repairing. This module provides a novel plan repair strategy among multiple agents sharing the same execution environment. It is responsible for repairing an agent's plan when a failure is detected concerning the new observed state and the original plan of the agent. This module utilises a new metric, plan commitment, as a heuristic to guide the search for the most committed repair plan to the original plan from the perspective of commitments made to other agents (the resources used in the original plan) whilst achieving the original goals. An agent that uses plan commitment to repair its plan will have no adverse effects on other agents. Consequently, the community of agents will suffer fewer failures due to the sudden changes (reduced revisions) or will have less lost time if the failure is inevitable.

Section 3.4 introduced a brief description of the framework modules constituting the contributions of this PhD thesis. Whereas Chapters 4, 5, and 6 provide a complete and detailed description of each module.

3.5 Chapter Summary

This section aims to summarise and highlight the main points that were presented in this chapter:

- Describing the usefulness of allowing two different representations to exist simultaneously to capitalise on the OWL ontological representation's richness and the PDDL representation's efficiency.
- Presenting the various components of the planning and the knowledge formalisations and the terminology used in the rest of this PhD thesis.
- Presenting the problem description with autonomous agents and human beings analogy of the efforts in this PhD thesis to handle unexpected events, whether to seize an opportunity or repair a failure with no adverse effects on other agents. In addition, describing the barriers tackled to bridge the gap between the two sides of the analogy.
- Presenting the abstract diagram and description of the method justifying the development of a three-modelled context-aware deliberative framework to seize an opportunity or repair a failure.
- Presenting the framework overview and the schema of the context-aware deliberative framework consisting of three main modules: the simulator, context-aware knowledge acquisition and the plan repairing module. In addition, a brief description of each module is presented.

The following chapter provides a complete and detailed description of the simulator module.

CHAPTER 4

An Intelligent System for Execution Simulation in a Dynamic Environment

EXECUTION MONITORING is crucial for the success of an autonomous agent that is executing a plan in a dynamic environment as it influences its ability to react to changes quickly. Due attention must be paid to changes; otherwise, the agent might suffer a plan execution failure or lose opportunities. Changes result from noisy observation, partial observability, non-determinism, dynamic environment or other agents operating in the same environment. Motivated by the previously mentioned reasons, the author's first contribution is an intelligent system for execution monitoring simulation in a dynamic environment (Babli, Ibáñez-Ruiz, et al. 2016).

In a nutshell, the key component of this intelligent system is the simulator in charge of the plan monitoring and execution. The simulator periodically updates its internal state with information from open data platforms and maintains a snapshot of the real-world scenario through live events that communicate sensible environmental changes. It builds a new planning problem when an unexpected change affects the execution and calls a planner to generate a new plan (replanning). It also constitutes the basic building block used in the author's contributions whenever a simulation of execution is needed.

Just like PDDL provides a straightforward way to model any application domain, the simulator is domain-independent and is developed as a *console application* to serve any application domain. Initially designed for smart tourism simulation in a dynamic environment, the simulator system is a console application that features a *graphical user interface* (GUI) tailored to smart cities' tourism environments. The simulator can easily be linked to other GUIs if needed. The development of different specialised GUIs is left to the users, depending on the requirements of their application domain. A complete smart tourism application domain example is presented in a case study in Appendix A.

4.1 Background

The development of products and services is facilitated by the exponential growth of the Internet of Things (IoT), and the proliferation of open data platforms cities and governments provide to leverage urban data and address societal challenges (Rathore et al. 2016; Vermesan et al. 2013). By partnering with smart technology embedded in every organisation and entity, organisations can increase their competitiveness by addressing the users' needs and enhancing their experiences (Gretzel et al. 2015). Through the personalisation of services before, during, and after the service and the use of information aggregation, ubiquitous mobile connectivity, and real-time synchronisation (Buhalis et al. 2015).

Dynamic adaptation and personalisation of activities are required in real-time to enhance the user experience, even when actions are pre-planned in advance. The ability to collect and update information in real-time is essential for smart technology to operate effectively; this means that data cannot only be collected a priori but must be collected and updated in real-time (Neuhofer et al. 2015).

PDDL provides a straightforward way to express the “physics” of the domain and the particular problem that instantiates it. In PDDL, the activities can be identified to be executed similarly to rules, with their preconditions, effects and other interesting features like duration, cost and reward. Then, planning generates a plan as an agenda of activities or actions (McDermott et al. 1998).

According to the literature, AI is used to ensure the development of smart cities for different application areas; such as urban traffic management and public transport operations, public safety and cybersecurity, education, human resource management, healthcare, tourism, smart homes, agriculture and many other public services (Rjab et al. 2019). As an example of a smart city application, the smart tourism domain provides many applications for planning travel, routes, trips and activities or catering to users’ needs by providing customised itineraries (agenda) with suggested activities for the user (Santiago et al. 2012; Mínguez et al. 2010; Sebastia et al. 2009; Vansteenwegen et al. 2011; Refanidis et al. 2014). Further, The Global Positioning System (GPS) technology allows a Recommender System (RS) to locate the future user’s location and suggest the most interesting places to visit based on their location. Such applications often feature a simple dynamic interaction between the user and the agenda. The user can add or remove activities or change their order as part of this dynamic interaction. The majority of these applications, however, work with fixed information throughout the execution of the activities. The problem is that they do not react readily to changes in the execution environment, such as a museum closing, a fully booked restaurant or a diverted bus. In turn, this significantly impacts how users view their experiences and are profoundly affected by such a static method.

In any given application domain, to develop an agile and adaptable plan that keeps pace with the dynamic environment, the agent must trace the execution of the plan and verify that activities occur as expected while the plan is being executed. This entails monitoring the plan and finding a new plan if a particular activity cannot be achieved. Therefore, we propose an intelligent system that monitors and simulates the plan execution in real-time to account for the dynamic nature of the execution environment.

4.2 Design Decisions

This section aims to explain two design decisions the author has followed in this PhD thesis for the simulation system presented in this chapter and also for repairing failures in Chapter 6.

4.2.1 Bespoke vs Off-the-shelf

> "*Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any*" (Brooks 1987).

This quote fits well with monitoring and simulation systems, as various models, infrastructures and applications' requirements call for different simulation systems. Multiple simulation frameworks for different programming languages support discrete event-based simulations, depending on the programming language. The work of (Lees et al. 2009) identifies and compares a variety of simulation systems such as LEE, JAMES II, SeSAm, RePast and SWARM, in addition there exist many other simulation and validation systems such as DESMO-J[1], SimPy[2], SUMO (López et al. 2018) and VAL (Howey et al. 2004a). The author does not claim that the proposed simulation system is better than other simulation frameworks in terms of performance; however, it is tailored specifically for this PhD thesis's needs. The tools mentioned earlier are valuable, provide utility and can be programmed to simulate particular scenarios. However, they are either too holistic or far too micro, do not consider the dynamic changes of the world and cannot react lively to them. In addition, many of these tools are domain-specific and have targeted application contexts such as artificial life (using neural networks), reactivity, cognitive agents or traffic and urban mobility. Consequently, these off-the-shelf systems cannot meet the specific needs of this PhD thesis. The requirements of this PhD thesis in terms of a simulation framework are:

- Simulate the behaviour of a general domain description in highly dynamic environments.
- Accept a general PDDL domain+problem description as an input description of the planning task.
- Receive live events during the execution in real-time.
- To execute and adapt the plan in light of new information about the dynamic execution environment.
- Monitor the achievement of PDDL hard and soft goals (preferences) in the plan's execution and ensure that they are pursued in the new plan should a failure occur.

[1] More info at http://desmoj.sourceforge.net/home.html
[2] More info at http://simpy.readthedocs.io/en/latest/

- Ease of integration with the rest of our planned deliberative infrastructure modules to repair a failure or seize an opportunity.

The proposed system acts as an intelligent system for smart cities' applications simulation in a dynamic environment and is tailored to overcome the previous limitations. In particular:

- The simulator is domain-independent. It accepts a PDDL planning problem description to allow the users to simulate a wide range of scenarios with different goals (either given or possibly obtained dynamically from a RS, a cognitive system or a deep-learning system), activities, preconditions and effects. A planner is invoked to get a plan.
- The plan is analysed, and then its execution is simulated just like it is executed in a real environment, simulating dynamic changes in the environment in real time. Based on the changes, the simulator checks if the real world matches the expected world.
- Suppose a failure in the plan is detected. In that case, the simulator reformulates the PDDL planning problem and reinvokes a planner to generate a new plan that satisfies the pending hard goals and as many of the soft goals, giving higher priority to those initially included (pursued) in the original plan.

This intelligent system was conducted as part of the Goal-management for Long-term Autonomy in Smart citieS (GLASS) project[3].

[3]More info at http://www.plg.inf.uc3m.es/~glass

4.2.2 Single-agent vs Multi-agent Re/Planning and Execution

Ideally, a given planning agent would be responsible for synthesising and executing plans and replanning to account for changes to the world state that the agent cannot foresee. However, there are many such agents in the real world, each with different objectives, yet all tied together by a common execution environment they share.

Planned actions by one agent affect the world's state and the conditions under which other agents must plan and act, resulting in complex dependencies. Full multi-agent planning can resolve the issues arising from changing plans due to failures in such cases, but it is far from a scalable solution for real-world domains (Talamadupula, Smith, et al. 2013). Instead, this multi-agent space filled with dependencies can be projected down into a single-agent space. Talamadupula et al. (2013) proposed to bring single-agent planning and multi-agent systems together in a unified theory. In this PhD thesis, we follow the central argument presented in (Talamadupula, Smith, et al. 2013), as a design decision, that the single-agent planning community needs to heed the changes to the world state by generating a new (single-agent) plan that remains consistent with the larger community of agents sharing the world.

This technique generally interleaves planning with plan execution and execution monitoring and applies single-agent re/planning to multiple agents and is followed by several intra-agent works in the literature, such as the works of (Mohalik et al. 2018; Cooksey et al. 2017). Therefore, in this PhD thesis, simulating execution and repairing failures are viewed from a single-agent perspective. On the one hand, in this chapter, we assume each agent has its own simulator responsible for reading the world from repositories or live events and executing its own plan. On the other hand, each agent is accountable for responsibly repairing its failures without disrupting other agents, as explained in Chapter 6.

Furthermore, the methods used in this PhD thesis are independent of how agents communicate and share information, i.e., the agent's communications are out of this PhD thesis scope. The agents can read data from repositories and can receive live events in the form of TILs. Therefore for this chapter, as mentioned previously, an agent's simulator periodically updates its internal state with information from open data platforms and maintains a snapshot of the real-world scenario in our implementation through live events that communicate sensible environmental changes. Similarly, in a community of agents (Chapter 6), if an agent wishes to share information with another agent, it is assumed to be received as a live event in the form of TILs.

4.3 System Architecture

Figure 4.1 shows the GLASS architecture, consisting of a two-process loop: a planning module and a simulation+monitoring module that share information.

The input information of our system is retrieved from different data sources:

1. The idea was to apply different strategies for obtaining the set of goals (i.e. user recommendations based on previous plans executions by other users) for each user using different utility RSs. First, a user profile containing the explicit interests of the user, the goals and preferences (such as places they want to visit and the temporal constraints) is retrieved.

2. Second, a different set of databases is used to identify and categorise the points of interest objects (such as attractions, museums and restaurants), their timetables, and geographical sources to find routes, distances, and

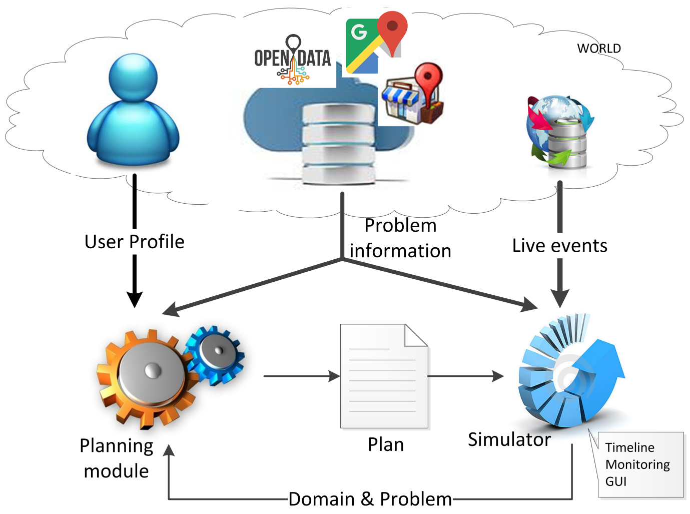


Figure 4.1: GLASS Architecture

times between points. We use Google Places[4] and Directions[5] APIs for this, although other open databases — like OpenStreetMap[6] — can also be used. The list of objects could also be supplied explicitly if no database is available.

3. Third, a snapshot of the environment or real-world scenario exists where the plan is executed. Due to the dynamic nature of this world and the possibility of changes frequently occurring (e.g., the opening hours of a museum changed, a restaurant was fully booked, or the dining duration was longer than expected), we get the new information as it happens as live events.

This work focuses on the online execution of the generated plan in a dynamic environment, whereas implementation details of how the system obtains the

[4]More info at https://developers.google.com/places/web-service
[5]More info at https://developers.google.com/maps/documentation/directions
[6]More info at http://wiki.openstreetmap.org/wiki/API

input set of goals from a RS utilising user profiles are abstracted out from this work due to the domain-independent nature of the system. An example of domain-dependent application use of users' profiles and RS in e-Tourism to provide a list of goals tailored to different tourists travelling styles can be found in the work of (Ibáñez-Ruiz et al. 2016).

On the one hand, the planning module takes the user profile and the scenario information to create a planning problem in PDDL format and invoke a planner. We need to model the users' preferences, constraints and actions. As a result, the user gets a personalised plan of the actions that must be taken.

On the other hand, in GLASS, rather than having a real-life group of users equipped with sensors to determine their current positions, pending and already accomplished goals, we simulate the execution as a proof of concept. This simulation process (detailed in Section 4.5) takes the plan and creates a timeline structure to run a timed events based execution. The system simulates and monitors the changes in the world state resulting from the changes in the plan, that is, responses to actions effects and, possibly, changes resulting from live events. The simulation process and output can be shown either as console line text commands or in the tourism domain; we have developed a specially designed Graphical User Interface that shows what is happening at any time.

During the simulation process, there may be a discrepancy between the expected and the actual state, as some live events prevent the remaining actions in the plan from being executed. In this case, a (re)planning module is required. A new PDDL problem specification will be reformulated to adapt the plan to the newly emerging scenario so that the planning module can be recycled for replanning, thus closing the loop.

The rest of this chapter is organised as follows. Section 4.4 presents the planning description, highlights the main components of a planning problem and presents hard goals and preferences. Section 4.5 examines the simulation behaviour in more detail, emphasising the reformulation of a planning problem.

4.4 Planning Module

In order for the system to provide a personalised itinerary (plan) to a given user, the user's preferences should be reflected in the resulting plan based on the user's profile (demographic classification, previous visits, and current preferences). In addition, the system must consider the availability of objects and the time needed to finish performing each activity. Therefore, to solve this problem, a planning system is necessary that supports durative actions (to represent the duration of actions), temporal constraints (to express the availability of objects as time windows), goals to express obligatory goals and soft goals (if any) to express the user's preferences. A RS returns the list of potential goals for the user that is considered as input to the system. Among the few automated planners capable of handling temporal planning problems with preferences, we opted for OPTIC (Benton et al. 2012) because it handles version 3.0 of PDDL (Gerevini, Haslum, et al. 2009), including soft goals. The information required by OPTIC to build the plan is compiled into a planning problem encoded in PDDL3.0 language.

A planning problem's initial state refers to the world at the time the plan starts being executed. The initial state must reflect the initial location of the objects, the availability of objects and the durations costs. Some information is expressed with fluents, while other information is represented by TILs.

Goals and Preferences

The simulator is prepared to deal with two types of goals:

- *Hard goals* represent the completion of mandatory activities specified by the user.
- *Soft goals or preferences* represent the completion of desirable activities that please the user and increase the plan's utility if achieved, but they are non-mandatory for the execution to be successful.

As preference definitions are expressed in PDDL3.0, we need to define how satisfaction or violation of each preference in a given plan will affect the plan's quality (penalties). When selecting the best plan, OPTIC will consider the penalties for the violations of preferences (costs) and choose the plan that satisfies most of the preferences and thus minimises the penalties for violations. In other words, the objective is to find a plan that achieves all the hard goals while minimising the plan metric to maximise preferences satisfaction; when a preference is not fulfilled, a penalty is added to the metric.

Specifically, we allow defining two types of penalties. The first penalty is the *penalty for violated preferences*; it helps OPTIC prioritise selecting activities and thus achieve the preferences that have a higher priority for the user. For a plan π, the penalty is calculated as the ratio between the priority of the preferences not included in π and the priority of the whole set of preferences recommended to the user RA:

$$P_{violated} = \frac{\sum_{a \in RA-\pi} Pr^a}{\sum_{a \in RA} Pr^a} \tag{4.1}$$

For example, if the priority for preference v_3 is x, and the sum of the priorities of all the preferences is y, the penalty for not satisfying preference v_3 would be expressed in PDDL as: `( / (*` X `(is-violated` $v3$`))` y`)`. The priority of the activities (Pr^a) is calculated by a hybrid RS, which returns a value between 0 and 300 according to the estimated degree of interest of the user in the preference. The RS also uses the value of Pr^a to return a time interval that encompasses the minimum and maximum recommendable activity duration following a normal distribution $N\left(\mu_a, \sigma_a^2\right)$, where μ_a represents the average activity duration for a typical user (Ibáñez-Ruiz et al. 2016). Thus, the higher the value of Pr^a, the longer the activity duration.

The second penalty (optional): *penalty for non-achiever* activities is to minimise time spent on activities that do not achieve goals (such as moving between locations) and encourage activities close to each other to be performed in succession. This penalty is calculated as the ratio between the duration of all non-achiever actions of π (π_m) and the plan's total makespan:

$$P_{non\text{-}achiever} = \frac{\sum_{a \in \pi_m} dur(a)}{dur(\pi)} \tag{4.2}$$

For example, the function `(total_moving_time user)` accumulates the time spent in movement actions, so this penalty would be defined in PDDL as: `( / (total_moving_time user)` $dur(\pi)$`)`. The plan metric to be minimised by OPTIC is expressed as the sum of both penalties: $P_{total} = P_{violated} + P_{non\text{-}achiever}$.

4.5 Simulator

In essence, the simulator is meant to execute the plan and monitor that everything works as expected. The first step is to create the structure that will be used to perform the simulation. We use a timed event simulation throughout a structure we call the timeline, where certain events are triggered at particular time instances, possibly provoking world state changes. The simulator checks the fluent conditions during the plan monitoring. The simulator prints the output of the plan simulation steps to a console window, and for the tourism domain, it visualises the plan trace using a specially designed GUI. Should a failure occur during plan simulation preventing a specific plan action from being executed, the simulator activates a replanning mechanism that involves reformulating the planning scenario in light of the new observed state. Next, we describe these tasks in more detail.

4.5.1 Timed Event Simulation: the Timeline

“*Simplicity is the ultimate sophistication.*” (Leonardo da Vinci 1452)

A *timeline* is a simple structure containing a collection of unique timed events in chronological order representing a sequence of world states that need to be monitored. As depicted in Figure 4.1, a simulator timeline is generated based on the plan’s actions, problem information and live events.

A timed event is an event that happens at a particular time instance τ and contains the following information:

1. $cond(a_i) = cond(a_i)_{\vdash} \cup cond(a_i)_{\leftrightarrow} \cup cond(a_i)_{\dashv}$: the start, over all, or end conditions (of one or many actions) to be checked at τ.
2. $eff(a_i) = eff(a_i)_{\vdash} \cup eff(a_i)_{\dashv}$: the start or end effects (of one or many actions) to be applied at τ.
3. TILs represent exogenous events defined as part of the problem information, so they are known upfront and expected to occur at time τ.
4. *Live events* dynamically occur during the execution/monitoring process, so they are unknown a priori.

This way, a timeline encapsulates the information about the plan, TILs and live events and the corresponding world states, irrespective of whether the plan is composed of sequential, parallel or concurrent actions. Given a plan with two actions (`move` and `visit`), of duration 20 and 60, respectively, and a live event that indicates the museum is closed (not open) at time instance $\tau = 90$, the resulting timeline is shown in Figure 4.2.

```
0.00 move person1 hotel museum [20.00]
20.01 visit person1 museum [60.00]
```

```
                90.00 live event (not (open museum1))
```

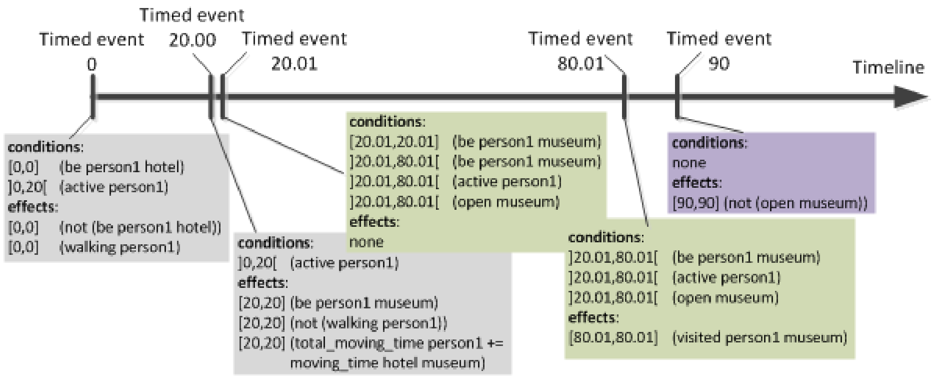

Timed event 0
Timed event 20.00
Timed event 20.01
Timed event 80.01
Timed event 90
Timeline
conditions:
[0,0] (be person1 hotel)
]0,20[(active person1)
effects:
[0,0] (not (be person1 hotel))
[0,0] (walking person1)
conditions:
]0,20[(active person1)
effects:
[20,20] (be person1 museum)
[20,20] (not (walking person1))
[20,20] (total_moving_time person1 += moving_time hotel museum)
conditions:
[20.01,20.01] (be person1 museum)
]20.01,80.01[(be person1 museum)
]20.01,80.01[(active person1)
]20.01,80.01[(open museum)
effects:
none
conditions:
]20.01,80.01[(be person1 museum)
]20.01,80.01[(active person1)
]20.01,80.01[(open museum)
effects:
[80.01,80.01] (visited person1 museum)
conditions:
none
effects:
[90,90] (not (open museum))

Figure 4.2: An Example of a Timeline with Five Timed Events. Note the Closed Interval for the *at start/end* Conditions and Effects, and the Open Interval for the *over all* Conditions

For the development of the simulator, the author used Java and PDDL4J[7], an open-source library that facilitates the development of JAVA tools for automated planning based on the PDDL language. The time scale of the timeline will depend on the granularity of the plan and the periodic steps we want to use for monitoring the timed events. In our implementation, live events can be manually supplied in a live fashion, or they can be retrieved dynamically from a database that keeps real-time information about the state of the world.

[7]More info at https://github.com/pellierd/pddl4j

4.5.2 Plan Execution Simulation

The simulation requires the users to specify how small the execution step will be to simulate the plan execution, thus determining the update frequency of the simulator's internal state with respect to its real-world counterpart. The simulator users may choose a larger step size or set the step size to the granularity of the planner. The smaller the size of an execution step, the more frequently external databases are accessed (Google APIs) to obtain new information and update the real-world state. Domains that exhibit frequent changes can be simulated with more reliable behaviour using a small execution step. Additionally, each timed event updates the simulation state, checking the conditions of the actions and applying the events' effects. Furthermore, the simulator receives live events from the real world during execution, modifying or creating new timed events in the timeline.

The simulation of the plan execution starts at $\tau = 0$, with an initial state equal to the real-world state, and the simulator advances through the timeline in every execution cycle $\tau = \tau + stepsize$ (see Figure 4.2). This process can be printed out to a console window (for domain independence purposes and other application domains) or visually shown in a specially designed GUI (described in Figure 4.5.5) for the smart tourism domain.

The simulator checks that the current state matches the expected state by checking that conditions are satisfied and then updates the current state accordingly. More specifically, every execution cycle involves two main steps:

1. Processing the live events for changes and updating the corresponding timed events

2. For every unprocessed timed event within the current step:

(a) Updating the current state with the TILs and effects of the live events

(b) Verifying that the timed event's conditions are satisfied in the observed state.

(c) Updating the current state with the effects of the actions.

The simulator detects a failure when a condition in a timed event is violated during execution (Step 2b shown above) difference between an expectation and the actual state that resulted in a situation where the plan can no longer be executable. In such a case, the interface (command line or GUI) informs the user about the failure cause: the failed action and the violated condition. For instance, in the example of Figure 4.2, supposing there is a live event at $\tau = 60$ that indicates the museum will no longer be open from 60 onwards. In this case, the *over all* condition `]20.01,80.01[ (open museum)` is violated, which means the `visit` action cannot be successfully executed. Once a failure is detected, the replanning module is invoked, as described in the following section.

4.5.3 Reformulating the Planning Problem

Once a failure is detected, the system performs the reformulation procedure steps shown in Figure 4.3:

Step 1: Creating the new initial state. The new initial state will comprise the information known by the simulator when creating the new problem (the current observed simulation state) and the information about future TILs (known information about forthcoming events). Therefore, the occurrence time of future TILs must also be updated.

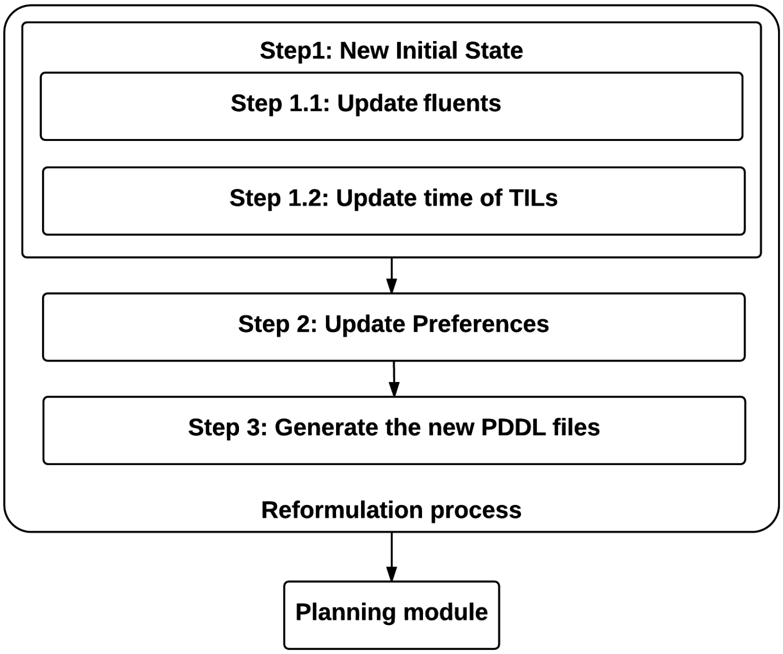

Step1: New Initial State
Step 1.1: Update fluents
Step 1.2: Update time of TILs
Step 2: Update Preferences
Step 3: Generate the new PDDL files
Reformulation process
Planning module

Figure 4.3: Reformulation Steps

Step 1.1: Updating fluents. This step refers to the update of the current state. After the failure, the fluents values are retrieved from the current observed state. However, this might not be accurate depending on how the actions of the domain were modelled, especially when a failure has resulted from a live event which violated an *over all* or *at end* condition. E.g., at the start of a walk action, a person is no longer at location1 and at the end of a walk action, a person is at location2. If a failure occurs due to an over all or at end condition of this action, the state will contain no fluent of the user's location. To tackle this problem, when a failure occurs due to an *over all* or an *at end* condition violation, the simulator will calculate the new initial state by rolling back the *at start* effects of the failing action (if any).

Step 1.2: Updating the time of TILs. When the new problem is reformulated, we invoke OPTIC, which resets the execution time and generates a plan starting from $\tau = 0$. Consequently, we need to update the occurrence time of the TILs to the result of its original time minus the failure time. Let us assume that a failure occurred at $\tau = 100$ and that we have the TIL planned at time $\tau = 235$; therefore, in the new initial state formulation, its time will be $(235 - 100 = 135)$.

Step 2: Updating preferences. When a failure occurs, the simulator distinguishes two cases for preferences or soft goals:

1. Goals already achieved by the successfully executed actions that preceded the failure. The reformulated problem will exclude all of these preference goals and their penalties.

2. Goals that have not yet been met, in turn, are divided into two sets:

   (a) Goals that OPTIC did not include in the original plan. The penalties for these goals are retained as initially defined in the problem file.

   (b) Goals included in the original plan and not yet satisfied due to the failure. The problem reformulation procedure increases the penalties of these goals to potentially enforce these goals in the new plan by assigning a relatively higher priority to these pending goals (twice as much as the maximum penalty among all goals).

Finally, three points are worth mentioning in this step. First, by executing the plan offline without any live events, the simulator learns the soft goals that the planner chose to pursue in the original plan. Second, the hard goals in the new reformulated problem file are kept intact. Third, the problem reformulation procedure applies a *preference stability strategy* that prioritises goals already included in the original plan over those that were not, which can be found as favourable and pleasing to users in various application domains. Other strategies, such as maintaining the plan stability metric concerning the original plan actions, can also be adopted (Fox, Gerevini, et al. 2006).

Step 3: Generating the new PDDL files. The final step of the reformulation process is generating the new PDDL files. The Domain file remains unchanged, except if a certain action needs to be added to the new plan; in such case, a *dummy effect* that triggers the corresponding action is added. Otherwise, only the problem file is created, taking into account the modifications discussed in steps 1 and 2.

4.5.4 Monitoring Preferences and State Trajectory Constraints

In addition to the features mentioned earlier, the simulator supports (as a side feature) monitoring the achievement/violation of the PDDL 3.0 preferences and state trajectory constraints modal operators handled by planners like OPTIC and TempLM (Marzal et al. 2017). The modal operators supported in the simulator are: *at end*, *always*, *sometime*, *within*, *at most once*, *sometime*, *after*, *sometime before*, *always within*, *hold during* and *hold after*.

In every execution cycle (defined in Section 4.5.2), the simulator checks all the preferences and state trajectory constraints except for *within*, *always within* and *hold after* because these can happen outside timed events and may require a complete (or interrupted) plan execution; therefore, they are processed after processing all the timeline timed events.

The simulator returns five sets of preferences: preferences followed by the planner in the original plan, preferences discarded by the planner in the original plan, preferences satisfied in the plan execution, preferences violated in the plan execution and preferences pending due to a failure.

There follows an example of the console output of the simulator. It shows the simulation steps and then prints statistics about goals and state trajectory constraints. More information about how we present and handle PDDL 3.0 preferences and state trajectory constraints modal operators is explained in our work (Marzal et al. 2017).

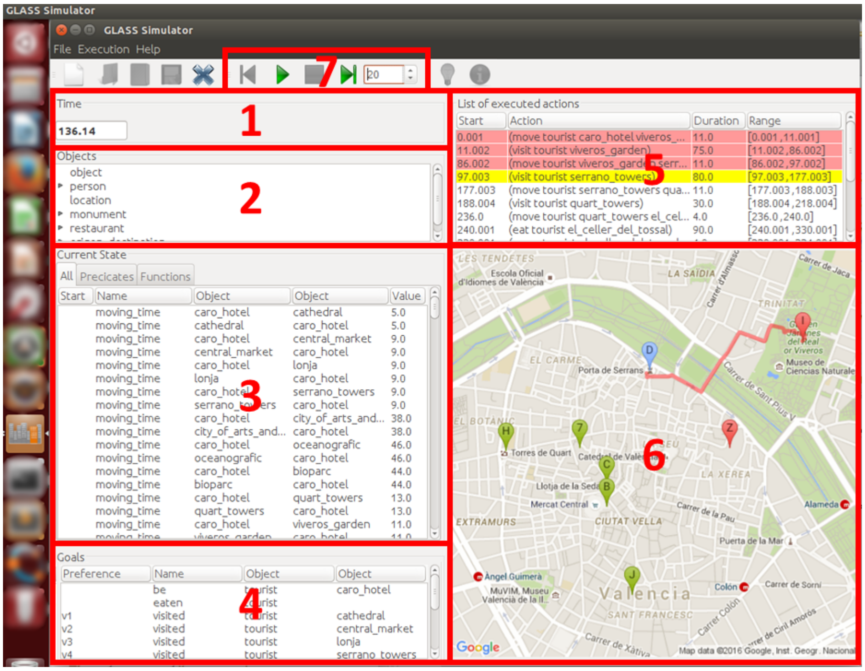
```
Welcome to the Simulator Console Interface!
Building TimeLine Started...
TILs (if any) were Added to the TimeLine...
Actions Conditions and Effects were Added to the TimeLine...
Building TimeLine Finished Successfully...
*************************************************************
              Information before Execution
*************************************************************
Constraints Included by the Planner in the Plan:
Hard Goals: [$g_1$,$g_2$]
Preferences: [$v_1$,$v_2$,$v_3$,$v_6$,$v_7$]

Constraints Discarded by the Planner in the Plan:
Preferences:[$v_4$,$v_5$,$v_8$]

Simulation Started.................
Initial World State at Time=0.0 = {...}.
Simulation World State at Time=Time+step size= {...}.
...
Last Simulation World State at Time=$n$ = {...}.
Simulation Finished Successfully.................
*************************************************************
              Information After Execution
*************************************************************
* Satisfied Hard Goals: [$g_1$,$g_2$]
* Satisfied Constraints
Preference: $v_1$ Satisfied at time= ...
Preference: $v_2$ Satisfied at time= ...
Preference: $v_3$ Satisfied at time= ...
Preference: $v_6$ Satisfied at time= ...
```

```
Preference: v7 Satisfied at time= ...
*Violated Constraints:
Preference: [v4,v5,v8]
```

4.5.5 Graphical User Interface



Time

136.14

Objects

- object
- person
- location
- monument
- restaurant

List of executed actions

Start	Action	Duration	Range
0.001	(move tourist caro_hotel viveros_...	11.0	[0.001 ,11.001]
11.002	(visit tourist viveros_garden)	75.0	[11.002 ,86.002]
86.002	(move tourist viveros_garden serr...	11.0	[86.002 ,97.002]
97.003	(visit tourist serrano_towers)	80.0	[97.003 ,177.003]
177.003	(move tourist serrano_towers qua...	11.0	[177.003 ,188.003]
188.004	(visit tourist quart_towers)	30.0	[188.004 ,218.004]
236.0	(move tourist quart_towers el_cel...	4.0	[236.0 ,240.0]
240.001	(eat tourist el_celler_del_tossal)	90.0	[240.001 ,330.001]

Current State

All | Precicates | Functions

Start	Name	Object	Object	Value
	moving_time	caro_hotel	cathedral	5.0
	moving_time	cathedral	caro_hotel	5.0
	moving_time	caro_hotel	central_market	9.0
	moving_time	central_market	caro_hotel	9.0
	moving_time	caro_hotel	lonja	9.0
	moving_time	lonja	caro_hotel	9.0
	moving_time	caro_hotel	serrano_towers	9.0
	moving_time	serrano_towers	caro_hotel	9.0
	moving_time	caro_hotel	city_of_arts_and...	38.0
	moving_time	city_of_arts_and...	caro_hotel	38.0
	moving_time	caro_hotel	oceanografic	46.0
	moving_time	oceanografic	caro_hotel	46.0
	moving_time	caro_hotel	bioparc	44.0
	moving_time	bioparc	caro_hotel	44.0
	moving_time	caro_hotel	quart_towers	13.0
	moving_time	quart_towers	caro_hotel	13.0
	moving_time	caro_hotel	viveros_garden	11.0

Goals

Preference	Name	Object	Object
	be	tourist	caro_hotel
	eaten	tourist	
v1	visited	tourist	cathedral
v2	visited	tourist	central_market
v3	visited	tourist	lonja
v4	visited	tourist	serrano_towers

Figure 4.4: The Simulator Graphical User Interface

Figure 4.4 shows the six parts of the GUI of the simulator used to provide information about the internal state of the simulator during the whole plan execution simulation. The GUI provides buttons for controlling the next step of the simulation. This GUI is domain-independent and can be used in any application domain except for the map part (Figure 4.4-part 6), specifically designed to offer a smart-city tourism orientation. There are six parts to the GUI, each of which is marked with a number and surrounded by a frame:

1. Figure 4.4-part 1 shows the current simulation time.

2. Figure 4.4-part 2 displays the planning problem objects along with their types. This static information will not change over the simulation process.

3. Figure 4.4-part 3 contains the PDDL description of the current state, which can change after an action starts or ends, when a live event arrives or when a user introduces a manual change (TILs). The fluents of the current state can be separately consulted in two different tabs (boolean and numeric).

4. Figure 4.4-part 4 shows the problem goals.

5. Figure 4.4-part 5 shows the list of plan actions, their start time, the objects involved in the action execution, and the duration of the action. In addition, the actions are shown with a representative colour: actions currently in execution are highlighted in yellow, past or already executed actions in red, and future actions in white.

6. Figure 4.4-part 6 shows the Representative map with icons of the relevant places involved in the plan (places to visit, restaurants and hotels). These location icons change their colour when the corresponding action is executed. The map also displays distances between locations.

7. Figure 4.4-part 7 contains the simulation control buttons. In the middle of the top ribbon, the interface displays four buttons to run the simulator step by step (the user defines the step size), continue the simulation, stop the simulation and reset the simulation.

4.6 Conclusion

In this chapter, the author has presented a domain-independent intelligent system that simulates the execution of a temporal plan in a dynamic environment.

In a nutshell, this intelligent system reads a PDDL domain and problem files as input. It calls a planner to generate a plan and simulates the execution in real-time. The simulation is done by transforming the plan into a timeline consisting of timed events (conditions to be checked and effects to be inserted). The simulator periodically updates its internal state with real-world information, receives sensible environmental changes through live events and creates their corresponding timed events in the timeline. It monitors the execution of the plan. Events are processed in the context of the plan; if a failure occurs due to live events, the simulator reformulates the planning problem. This involves creating the new initial state and updating the time of timed events and the goals. The simulator re-invokes a planner to generate a new plan and resumes the simulation. The simulator is a console application and has a GUI designed specifically for the context of smart tourism. A case study in Appendix A presents a smart tourism application domain and GUI to evaluate the system.

The system presented in this chapter can deal with unexpected situations to a limited extent in the sense that unexpected change is due to the objects and object types already known to the agent in the PDDL domain model. On the other hand, this system can be used side-by-side with the Context-aware Knowledge Acquisition module (the author's second contribution) to adapt to unexpected situations and seize opportunities from objects and object types that are entirely unknown to the agent. This is explained in detail in Chapter 5 and demonstrated in three application domain examples (tourism, repairing agency, and assisted living homes, in Appendices B and C).

The system uses an external planner to generate the original plan and fix failures. The system can be used side-by-side with the Plan Commitment Repair module (the author's third contribution) to ensure a responsible repair policy among multiple agents sharing the same execution environment and minimise the number of failures and the lost time in the community of agents (explained in detail in Chapter 6).

The next step in this PhD thesis is to provide context awareness to autonomous agents. The following chapter presents the knowledge extension through ontologies and provides a context-aware knowledge acquisition module integrated with the intelligent simulator system.

CHAPTER 5

Context-aware Knowledge Acquisition for Planning Applications

FROM the selfish-gene point of view (Dawkins et al. 2017), a metaphor for our behaviour or that of any particular animal species is that we are survival machines-robot vehicles blindly programmed to preserve the selfish molecules known as genes. We may face something that gets in the way or that can be exploited to our benefit to make the best use of the environment. To a certain extent, this metaphor resembles the rational agent that might face unpredicted change in the dynamic environment and must do its best to adapt.

Whilst online planning has demonstrated its usefulness in handling plan failures, unanticipated events that may bring about an opportunity for the task executed by the agent still presents a rich research area to be studied. An agent that relies on the prior knowledge of its designer rather than on its own percepts is an agent that lacks autonomy. Being rational requires the agent to gather information and extend what is already known from what is perceived to compensate for partial or incorrect prior knowledge and achieve the best possible outcome. The agent's initial configuration could reflect some previous knowledge of the environment. Still, the agent should be able to modify and augment this knowledge without being programmed to handle every possible scenario that could stem from encountering objects unknown to the agent or object types unfamiliar to the agent.

In solving this problem, the author had to address two issues: to extend the agent planning knowledge when encountering unforeseen circumstances related to the execution environment. Then, focus on allowing the agent to distinguish related and manageable objects reliably from unmanageable objects. Therefore, allowing the agent to extend its planning knowledge if and only if they are:

- Relevant to the agent (types).
- Manageable by its capabilities (action schemas).

Consequently, the agent becomes:

1. Aware of the dynamic execution environment.
2. Aware of the task it performs.

Consequently, the anticipation of opportunities or recovery from failures is facilitated. This chapter presents the previously mentioned efforts as a single module of context-aware knowledge acquisition for planning applications using ontologies.

It must be noted that the author's approach does not use ontologies to perform planning itself, as the complexity, the performance and the domain dependence become issues, as indicated in the work of (Gréa 2020). As a result of the differences in requirements for each field, the richness of ontological representations and planning representations is evident. A high level of expressivity is important in ontologies to enable more detailed domain descriptions, while a limited level of expressivity is required for effective planning in planning systems. These fields cannot be made more compatible by imposing representational constraints. Ontologists will not tolerate a reduction in ontology richness, and planners will not accept less efficient planning systems. Following the suggestion of (McNeill et al. 2005), the author permits different representations to exist simultaneously to resolve this conflict of interest. When PDDL and OWL are used simultaneously, the agent can use planning techniques for efficient planning and ontologies for context awareness by taking advantage of the additional detailed description of the knowledge domain provided by ontologies.

The rest of the chapter is organised as follows. First, we lead with a brief introduction, followed by three background examples from the author's publications that illustrate the benefits of using context-aware knowledge acquisition for planning applications. Then, a general overview of the approach schema is introduced, as well as a brief description of each component. The details of the ontological operations used in the approach's components are then described. Subsequently, we discuss some implications and future research suggestions, and finally, we conclude.

5.1 Introduction

Thanks to significant developments in automated planning technology, various new applications have been created, such as tourism (Babli, Ibáñez-Ruiz, et al. 2016), ambient assisted living (Babli, Rincon, et al. 2021), travel plan generation (Knoblock et al. 2001), space technology (Guzman et al. 2015; Muscettola et al. 1998), and underwater-installation maintenance (Cashmore et al. 2018). The planning community is keen on designing concise models for an automated agent acting in the real world. Various tools and approaches have been developed that automate the building of domain models, such as (Motta et al. 2004; Vaquero et al. 2013). Nevertheless, designing a planning model that allows an autonomous agent to respond intelligently to unexpected events in a real execution environment proves difficult for a variety of reasons, such as:

- The model design of the execution environment must be comprehensive enough to reflect the execution environment truthfully, yet conversely, concise enough for AP search technologies to compute solutions efficiently.
- Counting for every possible scenario in a non-deterministic execution environment inside the agent's model can be unreasonable, time-consuming and sometimes unfeasible. The result is a partially developed and incomplete model.

The agent is thus deprived of autonomy and is forced to rely on the knowledge of its designers rather than its own percepts (Russell et al. 2020), hence overlooking opportunities that any human executor would have seized or suffering failures that any human executor would have avoided.

Agents monitor the execution of the plan in the environment and are capable of formulating alternative goals on the fly. We briefly revisit some of the approaches explained in Section 2.4. In (Rao et al. 1995), an observed event triggers goals' adjustment in real-time, and the actions selection function is reapplied. (Norman et al. 1995) generates alternative goals when a function exceeds a threshold to avoid reaching a state where a failure occurs and a continuous operation is impossible. INTRO (Cox 2007) generates two kinds of goals that reduce the dissonance between expectations and observations and reduce the likelihood of repeating the same reasoning failure. LUiGi (Dannenhauer and Muñoz-Avila 2013) and LUiGi-H (Dannenhauer and Muñoz-Avila 2015) generate goals using domain-specific monitoring and case-based reasoning, respectively. ARTUE (Klenk et al. 2013), M-ARTUE (Wilson et al. 2013) and T-ARTUE (Powell et al. 2011) generate alternative goals to avoid failures, using manually constructed rule-based goal creation, domain-independent heuristics and goal learning interactively from humans, respectively. Dora the explorer (Hanheide et al. 2010; Hawes et al. 2011) has domain-dependent encoded motivators for generating goals relating to exploring newly detected rooms. Also, in the opportunistic planning approach (Talamadupula, Benton, et al. 2010), discovering new victims triggers goal generation and replanning.

A limitation of the previously mentioned approaches is that goals are mostly formulated based on objects in the agent model or objects of predefined types, such as in the approach of opportunistic planning (Cashmore et al. 2018; Cashmore et al. 2016). Cashmore et al. explain how unexpected events during an AUV's plan execution in autonomous underwater missions might offer opportunities for the vehicle to increase the overall utility of its operations. For example, a partially submerged section of an anchor chain, or other structure from the predefined types of structures, could be spotted during the execution of a mission. This event provides an opportunity to perform an unplanned inspection. In their model, they define an opportunity as a tuple $< t_i, g_o, u >$ where $t_i \in T$ is one of the PDDL types in the domain, g_o is a goal with at least one free argument arg_i of type t_i, and u is a utility value. The goal $g_o[arg_i]$ is called an opportunistic goal. Therefore, opportunities are soft goals associated with a particular object type t_i in the domain. The idea is that instances of a predefined type t_i can be discovered and added to the world during plan execution. Then, they create new soft goals by instantiating the free argument arg_i in the opportunistic goal, g_o. Nevertheless, the new objects must be of the agent's predefined model types, leading the agent to still suffer from limited context awareness. Agents only anticipate opportunities or failures related to existing objects or new objects of types already existing in the agent model. When unexpected events bring about new objects of unknown types, they will mostly pass unnoticed. The monitoring processes of the agent will not detect these unexpected events, nor will they reason about them. However, such events may benefit the agent to either seize an opportunity and increase the plan's utility or repair an execution failure if the agent has the capacity to deal with such new objects.

Our contribution in this chapter is a domain-independent approach that draws upon the richness and expressivity of standard ontology representations, semantic measures and ontology alignment for detecting and accommodating the newly acquired objects into the planning task specification. These new objects may subsequently trigger the formulation of a goal that induces a better-valued plan or can be used to repair failures. More specifically, our approach bolsters the agent autonomy, providing higher context awareness and advances state of the art by:

1. Extending the knowledge of a planning task with objects extracted from a collection of ontologies that describe features of interest for the specific domain and are relevant to the execution environment. Consequently,

higher environmental awareness (Babli, Onaindia, and Marzal 2018; Babli, Marzal, et al. 2018).

2. Enhancing the knowledge extension and tailoring it for planning dynamics to filter out objects unmanageable by the agent. Consequently, providing the agent with task awareness in addition to environmental awareness (Babli and Onaindia 2019).

5.2 Background

This section presents three examples from the author's publications illustrating the approach's benefits.

Consider a repair agency scenario in which a robot in a warehouse has a one-day maintenance task for several large kitchen appliances received by the agency. The warehouse has three areas; a transit area for items that require maintenance, an inspection area where maintenance is performed, and a storage area for things that have already been maintained. The scenario is formulated as a planning task including a specified set of categories of large kitchen appliances, the operations that the agent can perform and their durations (movement between the warehouse areas, a maintenance operation specialised for large kitchen appliances, loading, and unloading). The planner solves this task and returns a plan, which includes a total repairing of three items: two items of type *dishwasher* and one item of type *refrigerator*. During the plan execution, the repair agency receives a new item, *bosch_ID3400*, in the transit area from a different delivery agent operating in the same city. The new item type is requested from that agent and is found to be a *kitchen_range*, not formerly considered in the planning task of the repair agency. This new object may represent an opportunity if the goal of repairing it can be aligned within the modelling of the planning task of the agency and triggers a plan compliant with the current goals resulting in achieving an extra unplanned additional goal, providing better utility. However, suppose the repair agency receives a new object, *iphone_ID7500*, of type *mobile_phone* during execution. Unlike the *bosch_ID3400* case, the autonomous agent is not equipped to repair *iphone_ID7500*. Therefore, the agent should deem the new object irrelevant as it is beyond its operational capabilities and must not erroneously integrate it into its model (the case of *iphone_ID7500*) or generate a goal to repair it. To a human operator, this distinction is simple; however, for an autonomous agent, it is not trivial and requires the agent to be context-aware of the task it can perform. This scenario is detailed in Appendix B.

Consider a tourism scenario where a tourist wishes to make a one-day tour to visit several points of interest (POIs) in a city. The scenario is formulated as a planning task including a specified set of POIs of different categories (predefined types), the opening and closing times of these POIs, the walking time between locations and a recommended list of potentially visitable POIs for the tourist (goals). The planner solves this task by producing a plan which includes a total of four visits: two visits to POIs of type *religious site*, one visit to a POI of type emblematic *architectural building* and one visit to a POI of type *aquarium*. While the plan is being executed, the tourist receives a cellphone notification about opening a nearby exhibition of Picasso's paintings. This external information includes a new object type, *art exhibition*, not formerly considered in the planning task. The new objects may present an opportunity to the tourist if the goal of visiting *Picasso's exhibition* can be aligned with the modelling of the planning task and triggers a plan compliant with the current goals resulting in a better tourism experience for the tourist. This scenario is also detailed in Appendix B.

Consider an example scenario where a robot in a facility such as an assisted living house is tasked with providing target group members with objects to satisfy their needs. The facility has several target group members and several rooms and corridors with different objects that can fulfil the members' needs. The scenario is formulated as a planning task including a specified set of categories of objects, the robot's operations, and their durations (carrying, moving and giving). The planner solves this task and returns a plan to comfort a member by providing a specific object. Given the dynamic nature of the environment, some existing objects might be unavailable during execution (being unexpectedly used by other members, broken, or malfunctioning), leading to execution failures. Additionally, new types not previously introduced in the model may also be encountered (added by staff or caretakers) during the plan's execution and could be used to resolve execution failures. This scenario is detailed as a complete application in Appendix C.

5.3 Overview of the Approach

We first briefly revisit the plan monitoring and execution simulator our approach relies on (Babli, Ibáñez-Ruiz, et al. 2016) detailed in Chapter 4. The simulation system takes $\phi = (Dom, Prob)$ and π as input and encodes them into a timeline as a collection of chronologically ordered timed events encapsulating the changes expected in the subsequent states. During the plan execution simulation process, more specifically, in every execution cycle (Section 4.5.2), the simulator system:

- Receives live events and adds them to the timeline; live events convey information about the execution environment, and exogenous events convey external information from other agents operating in the same environment, dynamically modifying the real-world states.

- Executing the timed events, checking that conditions of the plan actions are satisfied and the effects happen when they should, thus validating and updating the states of the world (timeline).

The system can generate the expected states trivially by running the simulation offline without reading live events with the simulation step size equal to the plan's makespan. Dynamically simulating plan monitoring involves observing the state that results from executing the plan actions in the environment and checking whether the observed state matches the expected state. This operation creates a discrepancy set as the difference between the two sets, which will comprise fluents of the form $(name(v)\ o_{i=1}^{arity(v)})$, where v is a variable that belongs to the finite set of variables V of the domain Dom, $name(v)$ is the variable name (label) and $o_{i=1}^{arity(v)}$ are the objects which v was instantiated with. Two cases can be distinguished:

- $\forall o_i \in o_{i=1}^{arity(v)} : o_i \in O$. All the fluent objects are known to the agent (belong to the predefined finite set of objects of the planning problem $Prob$).
- $\exists o_i \in o_{i=1}^{arity(v)} : o_i \notin O$. There exists an object in the fluent that is unknown to the agent (does not belong to the predefined finite set of objects of the planning problem $Prob$).

The first case is typical, and its discrepancies may lead to failures handled using our simulation system. This chapter focuses on the second case, where a new object exists. The system can retrieve the type of the new object either from:

- Other remote ontologies (Babli, Onaindia, and Marzal 2018) which is helpful for application domains whose objects are proper nouns. E.g., the

Eiffel Tower in a tourism application domain and the Mondeca Tourism Ontology that includes essential concepts of the tourism domain; locations, accommodations and concepts that describe leisure activities and geographic data.

- The source agent of the new object, as in (Babli and Onaindia 2019), when object names are not proper nouns.
- Using other advanced techniques such as image recognition in deep learning (Babli, Rincon, et al. 2021).

The author designed the approach sketched in Figure 5.1.

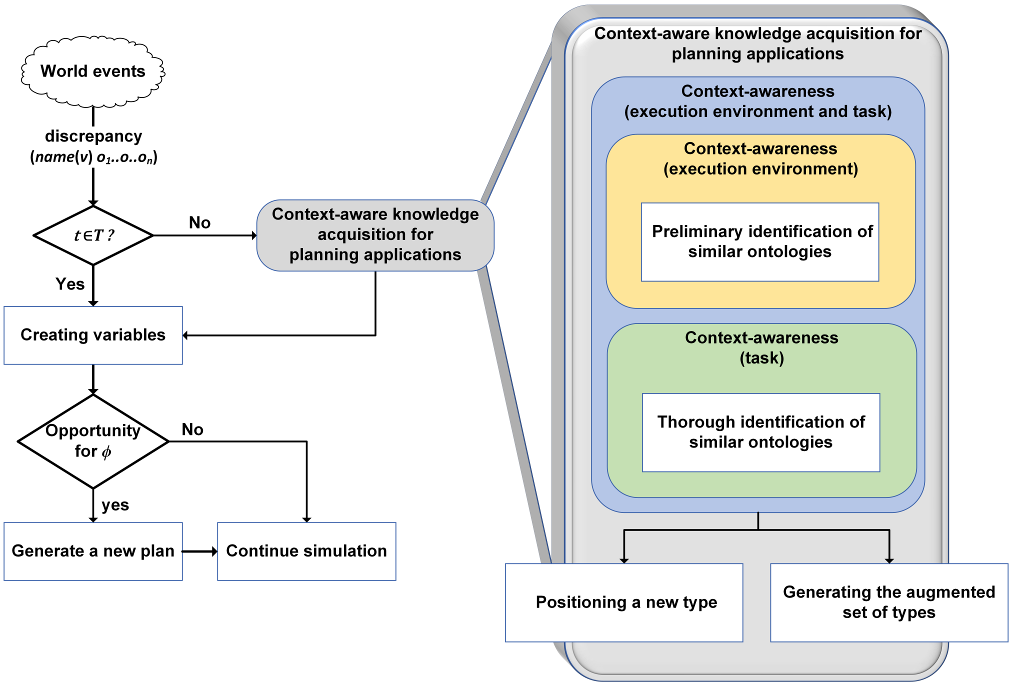

World events
discrepancy
(name(v) o1..o..on)
t∈T ?
No
Yes
Context-aware knowledge acquisition for planning applications
Creating variables
Opportunity for ϕ
No
yes
Generate a new plan
Continue simulation
Context-aware knowledge acquisition for planning applications
Context-awareness (execution environment and task)
Context-awareness (execution environment)
Preliminary identification of similar ontologies
Context-awareness (task)
Thorough identification of similar ontologies
Positioning a new type
Generating the augmented set of types

Figure 5.1: Context-aware Knowledge Acquisition using Ontologies

When a new object is received, two cases can be distinguished:

- $t \in T$, the type of the new object is literally one of the existing types.
- $t \notin T$, the new object type is not one of the predefined types of the planning task.

When $t \in T$, the object is integrated into the planning task with its corresponding variables, which may trigger an opportunity goal and a new plan.

On the other hand, when $t \notin T$, the agent utilises the author's proposed approach, context-aware knowledge acquisition for planning applications using ontologies to:

1. Decide whether t is relevant to the agent's execution environment model. The agent performs a preliminary identification of similar ontologies before positioning the new type t. This corresponds to the orange box in Figure 5.1.
2. Decide whether t is manageable using the agent's variables and operators OP (its actions schemas). The agent thoroughly identifies similar ontologies before positioning the new type t. This corresponds to the green box in Figure 5.1.

The new type of the new object is integrated if and only if it is relevant to the agent's execution environment and is manageable by the agent's capabilities. Otherwise, the agent should deem the object unmanageable (according to the operators and the variables in the agent's model) and thus ignores that object.

This process is helpful whether online during execution for positioning a single new type of a single new object or offline for generating an augmented set of types that are relevant and manageable by the agent to keep an eye on for monitoring when execution starts.

When the new object is integrated, it may subsequently trigger the generation of an opportunity goal, as in the author's work (Babli, Onaindia, and Marzal 2018; Babli and Onaindia 2019) explained in B. On the other hand, it may be used to repair an execution failure when replanning or repairing is invoked, as in (Babli, Rincon, et al. 2021) explained in Appendix C.

5.4 Context-aware Knowledge Acquisition for Planning Applications using Ontologies

The approach's four main phases are depicted in Figure 5.1, each of which is described below.

Phase 1: Preliminary identification of similar ontologies.

This phase provides the agent context awareness of the execution environment and allows it to integrate new objects relevant to the agent. Phase 1 corresponds to the orange box in Figure 5.1 and consists of four stages, as shown in Figure 5.2.

First, the agent creates a preliminary OWL ontological representation of only the types T of the agent's planning task ϕ, the resulting ontology is called η_ϕ. Second, the agent either retrieves a set of remote ontologies describing the application domain (used for applications where objects are proper nouns, such as tourism), or retrieves a set of partial remote planning tasks Δ with PDDL representation, precisely the types T of several semi-cooperative agents from online repositories, and creates their OWL ontological representation R^Δ of only the types. Third, the agent augments the classes representing the PDDL types with OWL annotations to counter the natural complication of lexical alterations. Subsequently, in the fourth stage, the agent applies a preliminary quick similarity measure, a vector space model distance (VSM) to R^Δ, and

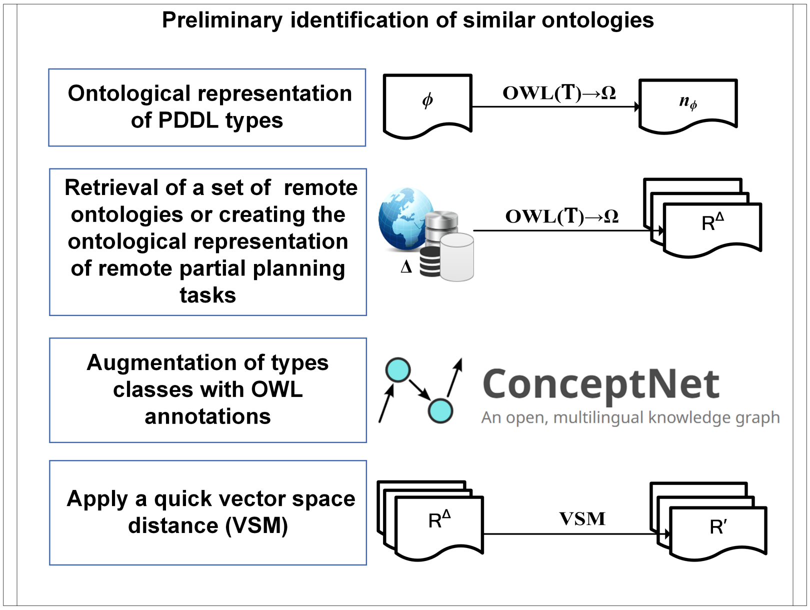

Preliminary identification of similar ontologies
Ontological representation of PDDL types
ϕ
OWL(T)→Ω
n_ϕ
Retrieval of a set of remote ontologies or creating the ontological representation of remote partial planning tasks
Δ
OWL(T)→Ω
R^Δ
Augmentation of types classes with OWL annotations
ConceptNet
An open, multilingual knowledge graph
Apply a quick vector space distance (VSM)
R^Δ
VSM
R'

Figure 5.2: Preliminary Identification of Similar Ontologies

obtains the set R', which contains the potentially most similar ontologies to η_ϕ according to classes (that represent PDDL types).

Phase 2: Thorough identification of similar ontologies.

This phase provides the agent context awareness of the task the agent performs and allows it to integrate new objects that are manageable to the agent's capabilities (variables and operators). Phase 2 corresponds to the green box in Figure 5.1 and consists of four stages, as shown in Figure 5.3.

In the author's publications, this is used both online and offline. It is used online before positioning the new type in the agent model if the type of the new object is retrieved from a remote ontology or from the source agent which delivered the new object, as in the works (Babli and Onaindia 2019; Babli, Marzal, et al. 2018; Babli, Onaindia, and Marzal 2018). On the other hand, it is

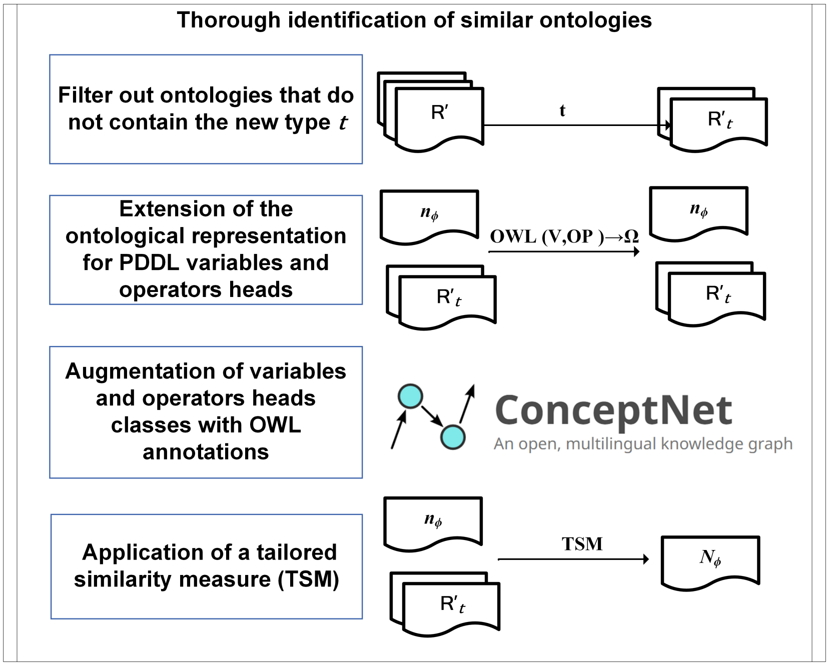

Thorough identification of similar ontologies
Filter out ontologies that do not contain the new type t
R′
t
R′_t
Extension of the ontological representation for PDDL variables and operators heads
n_ϕ
OWL (V,OP)→Ω
R′_t
Augmentation of variables and operators heads classes with OWL annotations
ConceptNet
An open, multilingual knowledge graph
Application of a tailored similarity measure (TSM)
TSM
N_ϕ

Figure 5.3: Thorough identification of similar ontologies

also used offline if the agent needs to be trained to recognise the new types of objects by generating an augmented set of types offline (candidate types) that are potentially useful for the agent later on during execution in case of failure, as in the work of (Babli, Rincon, et al. 2021).

For online usage, the information of a new object $o \notin O$ is received in the form of $(name(v)\ o_1 \ldots o \ldots o_n)$. First, the agent can identify the class/type of the new object o from other remote ontologies or the source agent which delivered o and finds $t \notin T_\phi$. As illustrated in Figure 5.3, among R′, the agent filters out ontologies that do not contain t to get R′$_t$. Then, the agent extends η_ϕ and R′$_t$ to represent not only types T but also variables V and the heads part of actions schemas OP. By doing so, we have represented the relationships (variables) established among the agent's types and capabilities regarding the operations it can perform with these types (operators' heads). It is safe to assume that semi-cooperative agents would share information only related to types T, relationships (representing variables) associated with types V and heads part of operations OP applied to types, with no private details such as objects, state variables (fluents), how the operations are performed (conditions and effects), goals or states. The agent augments the classes representing the PDDL variables and the classes representing heads part of actions schemas OP in η_ϕ and R'_t with OWL annotations to counter the natural complication of lexical alterations. Subsequently, the agent applies a tailored semantic similarity (TSM) measure between η_ϕ and ontologies in R'_t (to consider the planning dynamics of V, OP and obtain N_ϕ). An ontology in N_ϕ with a high similarity value means that the remote agent is equipped with similar capabilities. If the similarity value is higher than a specified threshold, then the new type t is not only relevant but also manageable by the actions' schemas. In contrast, a low similarity value means the agent cannot manage the new class t.

On the other hand, when we do not know the type of the new object, we prepare the agent to be capable of recognising the objects' types that are relevant and manageable. Generating the augmented set of candidate types potentially useful for the agent is done as follows. For each type (class) t_i in the predefined set of object types T of the agent's planning task ϕ, the agent applies the stages mentioned in the previously mentioned paragraph, starting by filtering out the remote ontologies that do not contain t_i, and ending with multiple set of remote ontologies $\{N^\phi_{t_1} \dots N^\phi_{t_i} \dots N^\phi_{t_n}\}$. These ontologies represent the ontologies of the remote agents equipped with similar capabilities to the agent with the planning task ϕ and contain t_i and other classes similar to t_i (t_i siblings). By choosing the most similar ontology from each set, the agent aggregates a specified number of types classes from these ontologies and generates the list of augmented set of types, on which the agent will be trained (offline and only once) to recognise. Type training (learning), where the agent is trained on the augmented set of types, is abstracted from this chapter and explained in Appendix C, where the agent recognises objects of the relevant types through an objects classifier model.

Phase 3: Positioning a new type and integrating a new object.

The agent attempts to position the new class ω representing the new type t of the new object o in η_ϕ, as shown in Figure 5.4.

Among N_ϕ, which are the remote ontologies that represent the most similar agents in terms of the relevance of types and operations and contain the class ω, the agent applies a semantic variance filter to select the ontology with the highest semantic insight, which is referred to as η_ω then it attempts to perform a semantic alignment with a neighbourhood constraint between η_ϕ and η_ω.

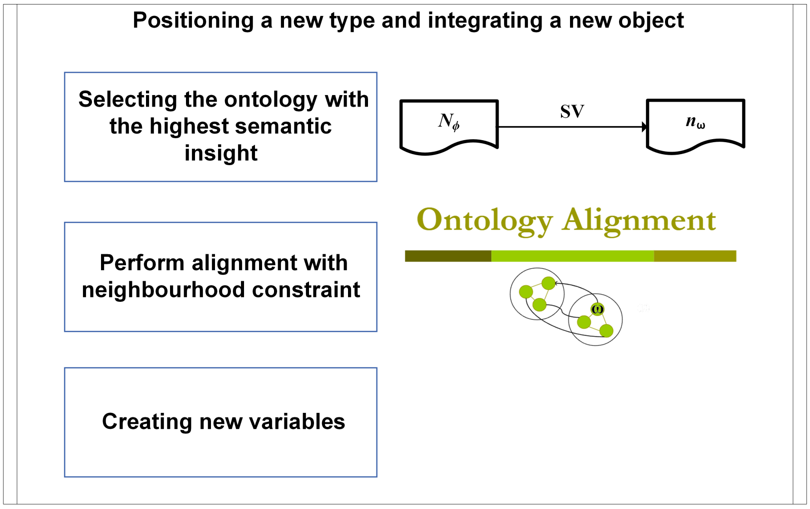

Positioning a new type and integrating a new object
Selecting the ontology with the highest semantic insight
Perform alignment with neighbourhood constraint
Creating new variables
N_ϕ
SV
n_ω
Ontology Alignment
ω

Figure 5.4: Positioning a new type and integrating a new object

If the alignment is successful, ω is positioned in the hierarchy of classes in η_ϕ, and t is added to T as arguments for relating V and as parameters in relating heads of OP. In addition, o is added to O, and the required planning state variables that involve o and their associated values are identified and added to I.

Phase 4: Goal formulation and opportunity identification. Suppose the type t of the new object o that is represented by the class ω appeared to be a class or a sibling of a class (in η_ϕ) that is involved in a goal $g \in G$. In that case, the agent formulates x candidate new goals that involve the newly received object o, where x depends on the possible permutations of objects in the goal variable; $x = 1$ if the goal has only o as a parameter. Once the x candidates are formulated, the system generates $\Phi' = \phi'_1, \ldots, \phi'_x$ (modified versions of ϕ), where the added information includes o, t, the information of o, the discrepancy variables, $G' = g'_i \cup G$, and the new current state. The agent invokes a planner to solve each $\phi'_i \in \Phi'$ to know which g'_i can be considered an opportunity to ϕ in the context of π. g'_i is considered an opportunity goal for ϕ when the planner is able to generate a plan π'_i to solve ϕ'_i that includes the new goal plus the original set of goals.

The following sections detail the ontology-based operations used in our approach. Namely, the OWL ontological representation, augmenting classes using ConceptNet, the VSM distance used in the preliminary identification of similar ontologies, the TSM distance used in the thorough identification of similar ontologies, the SV filter used to select the ontology with the highest semantic insight and the alignment with neighbourhood constraints.

5.5 Ontology-based operations

The author used Java and *The OWL API* to implement the ontology-based operations. *The OWL API*[1] is an open-source efficient Java API for creating, parsing, manipulating and serialising OWL Ontologies. For clarity and conciseness, the ontology-based operations are detailed in this chapter. In contrast, Appendices B and C demonstrate the ontology-based operations and the use of context-aware knowledge acquisition in several application domains, using snapshots from the GUI of Protégé (Musen 2015) to show visual explanations of the ontological representation.

5.5.1 Preliminary OWL Ontological Representation

This subsection corresponds to the first stage in Figure 5.2. Given a planning task of the agent $\phi = (Dom, Prob)$, where the domain is a tuple $Dom = (T, V, OP)$, and the problem is a quadruple $Prob = (O, I, TILs, G)$. The preliminary OWL Ontological Representation η_ϕ consists of a finite set of OWL classes (concepts) Ω used to represent T. PDDL offers the ability to express a type structure for the objects in a domain which allows typing the parameters that appear in variables and operators. Furthermore, PDDL allows the types to be expressed as forming a particular hierarchy. Similarly, when the agent creates the OWL classes representing T, for each type t in T of ϕ, an OWL class ω is created in η_ϕ, abiding by the exact hierarchy defined in ϕ. The set of classes representing the PDDL types is referred to as Ω_T.

As previously mentioned, for application domains whose objects are proper nouns, e.g., Eiffel Tower and smart tourism (Babli, Onaindia, and Marzal 2018), it is useful to represent the set of objects O of the problem instance as

[1] More info at https://github.com/owlcs/owlapi/wiki

individuals Λ of their corresponding classes, so the classes of individuals (the types of the objects) can be identified using remote ontologies. For example, assuming that the definition of a particular tourism problem instance includes a declaration of a set of objects O. For each $o \in O$ of type $t \in T$, an OWL individual λ of the class ω is created in η_ϕ.

5.5.2 Augmentation of Classes using ConceptNet

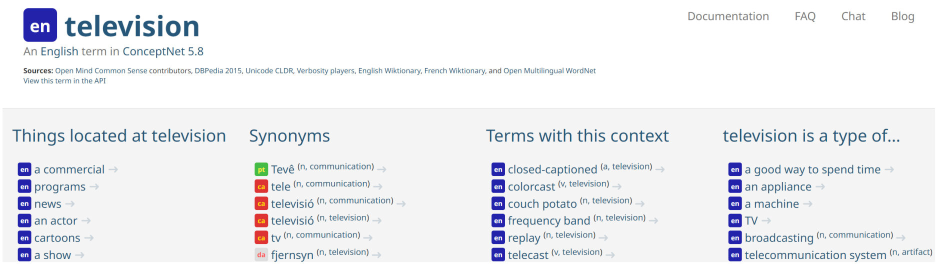
We need to counter the natural complication of lexical alterations and that different people can model the same domain using different terms and languages.

The agent augments the classes of the ontologies with the relations and concepts brought from ConceptNet (modelled as OWL annotation labels) as a standard means to describe concepts. Therefore, η_ϕ will also contain a set of OWL annotations Ξ used to describe Ω, where a class $\omega \in \Omega$ can have one or many annotation labels. ConceptNet is a freely-available semantic network designed to help computers understand the meanings of words that people use. ConceptNet is a knowledge graph that utilises a closed set of 36 selected relations, such as isA, usedFor and hasProperty, to represent relationships between concepts independently of the language or the source of the terms it connects (Speer et al. 2017). As a result, even if the names of the classes are different, classes that refer to the same concept will have annotations in common and will be found similar when measuring semantic distances or when performing the alignment. An example of the facts that can be obtained about a concept from ConceptNet's browsable interface[2] is shown in Figure 5.5.

In our implementation, the agent accesses ConceptNet using its API[3], which allows the agent to query the knowledge about any concept. Figure 5.6 demon-

[2]More info at conceptnet.io

[3]More info at https://github.com/commonsense/conceptnet5/wiki/API



en **television**

An English term in ConceptNet 5.8

Sources: Open Mind Common Sense contributors, DBPedia 2015, Unicode CLDR, Verbosity players, English Wiktionary, French Wiktionary, and Open Multilingual WordNet
View this term in the API

Things located at television	Synonyms	Terms with this context	television is a type of...
en a commercial →	pt Tevê (n, communication) →	en closed-captioned (a, television) →	en a good way to spend time →
en programs →	ca tele (n, communication) →	en colorcast (v, television) →	en an appliance →
en news →	ca televisió (n, communication) →	en couch potato (n, television) →	en a machine →
en an actor →	ca televisió (n, television) →	en frequency band (n, television) →	en TV →
en cartoons →	ca tv (n, communication) →	en replay (n, television) →	en broadcasting (n, communication) →
en a show →	da fjernsyn (n, television) →	en telecast (v, television) →	en telecommunication system (n, artifact)

Figure 5.5: ConceptNet's browsable interface facts about "television"

strates an example of a class television in OWL/XML format, created via The OWL API, which is a subclass of a class major_appliance, with a sample of label annotations (brought from ConceptNet) that a television: isa an appliance, synonym is TV, atlocation a house, capableof show images, relatedto television program and isused for watching a show.

```xml
<!-- http://www.semanticweb.org/repair#television -->
<owl:Class rdf:about="http://www.semanticweb.org/repair#television">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/repair#
        major_appliance"/>
    <rdfs:label>television isa an_appliance</rdfs:label>
    <rdfs:label>television atlocation a_house</rdfs:label>
    <rdfs:label>television capableof show_images</rdfs:label>
    <rdfs:label>television synonym tv</rdfs:label>
</owl:Class>
```

Figure 5.6: OWL/XML format representation of a class television

5.5.3 Vector Space Model Similarity Measure

In ontology engineering, knowing quickly if two ontologies are similar is helpful before deciding to match them. For measuring the distance between ontologies, we decided to look at the ontologies as a bag of terms; consequently, ontology distance measures based on the Vector Space Model (VSM) are applicable. For this purpose, we used a Vector Space Model similarity measure using cosine index with TF (weighted frequency term) to preliminary filter ontologies unrelated to η_ϕ among R^Δ (thus irrelevant classes) and obtain R' as the set of ontologies that may seem most similar to η_ϕ according to classes.

Measuring similarity in the VSM using cosine index with TF has proven to obtain good results. It computes much faster than other distance measures but is not robust to lexical alterations (David et al. 2008). However, lexical alterations do not impact our approach thanks to augmenting the classes with OWL annotations coming from ConceptNet as a standard means to describe concepts. Therefore, the lexical information in each ontology class comes from the local name of the term and the annotations imported from ConceptNet relations and classes. To compute the distance, the author used *OntoSim*[4], an independent open-source Java API that allows computing similarities between ontologies, can be used with *The OWL API* and provides a variety of distance measures. At this point, the ontologies R' (shown in the last stage in Figure 5.2) with the highest types' similarity with respect to η_ϕ are obtained.

[4]More info at https://gitlab.inria.fr/moex/ontosim

5.5.4 Extended OWL representation

This stage aims to extend the ontological representation to account for the planning dynamics by representing the variables V and the head parts of the operators OP.

Similarly to Section 5.5.1, when the agent created the ontological representation of types T, for each $v \in V$ and for each $op \in OP$ a class is created in η_ϕ resulting in Ω_V and Ω_{OP}. As V have typed arguments $args$ and OP heads have typed parameters $pars$, the representation utilises OWL object properties $\mathsf{hasParameter}_{1...n}$ in Ψ to specify $args$ and $pars$ of V and OP, respectively. By specifying the OWL domain and range of the object property $\mathsf{hasParameter}_i$ for the particular class, in addition to using the qualified cardinality "exactly" restriction to describe a class based on class members' relationships within an ontology.

Besides the axioms that the class be is a subclass of the class variable and the label annotations, Figure 5.7 shows the OWL/XML format representation created by The OWL API for specifying arg_1 and arg_2 of the class representing the variable `(be ?locatable - (either dishwasher refrigerator robot television) ?loc - location)` utilising the qualified cardinality restriction on the object properties $\mathsf{hasParameter}_{1...n}$.

5.5.5 A Tailored Semantic Similarity Measure for planning dynamics

The tailored Semantic Similarity Measure (TSM) is shown in Algorithm 1.

TSM is designed to exploit the information from the classes names and the annotations imported from ConceptNet that are attached to these classes. Utilising a hybrid string similarity measure SoftTFIDF (Cohen et al. 2003) to

```
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.semanticweb.org/repair#
             hasParameter1"/>
        <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
             nonNegativeInteger">1</owl:qualifiedCardinality>
        <owl:onClass>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="http://www.semanticweb.org/repair#
                         dishwasher"/>
                    <rdf:Description rdf:about="http://www.semanticweb.org/repair#
                         refrigerator"/>
                    <rdf:Description rdf:about="http://www.semanticweb.org/repair#
                         robot"/>
                    <rdf:Description rdf:about="http://www.semanticweb.org/repair#
                         television"/>
                </owl:unionOf>
            </owl:Class>
        </owl:onClass>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.semanticweb.org/repair#
             hasParameter2"/>
        <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
             nonNegativeInteger">1</owl:qualifiedCardinality>
        <owl:onClass rdf:resource="http://www.semanticweb.org/repair#location"/>
    </owl:Restriction>
</rdfs:subClassOf>
```

Figure 5.7: OWL/XML representation of `(be ?locatable - (either dishwasher refrigerator robot television) ?loc - location)`

Algorithm 1 Tailored similarities for T, V, or OP

```
1: procedure $TSM(c_1, c_2)$
2:     if InflictedForm($c_1$,$c_2$) $\vee$ Synonyms($c_1$,$c_2$) then
3:         SimValue = 1
4:     else if SoftTFIDFdistance($c_1$,$c_2$,synonym)$> 0.7$ then
5:         SimValue = SoftTFIDFdistance($c_1$,$c_2$,synonym)
6:     else
7:         Relations={synonym,isA,usedFor,atLocation,capableOf,relatedTo,antonym,hasA,
8:                 derivedFrom,hasContext,uri}
9:         SimValue = AverageOfNonNullValues($c_1$,$c_2$,Relations))
10:     return $SimValue$
```

measure similarities between classes to support a higher degree of syntactic variation between the terms. SoftTFIDF combines TF-IDF, a token-based similarity widely used in information retrieval (Raghavan et al. 1986), with the Jaro-Winkler edit-based (Winkler 1990) with a threshold of 0.9 for two tokens to be considered similar as used in (Gracia et al. 2013) to promote high precision, and with a threshold of 0.6 for the SoftTFIDF above which two classes are considered similar. There follows an explanation of how the algorithm works on two classes:

- If inflected form or an exact synonym, then `SimValue=1`, as shown in lines $1, 2$ of Algorithm 1.
- Else if the SoftTFIDF distance according to the synonym annotations is greater than 0.7, then `SimValue=`SoftTFIDF distance, as shown in lines $3, 4$ of Algorithm 1.
- Else `SimValue=` the average non-null values of the SoftTFIDF distance for the `Relations`, as shown in lines $5, 10$ of Algorithm 1.

Algorithm 1 is applied to obtain three similarity matrices for types, variables, and heads of operators. Since variables have arguments and heads of operators have parameters, the values of their matrices must be calibrated to consider the semantics of the concepts representing the arguments/parameters. Algorithm 2 shows how the similarity values are calibrated.

Algorithm 2 Calibrate Variables or Op heads matrices

```
1:  procedure $calibrate(op, op')$
2:      $total \leftarrow 0$
3:      $counter1 \leftarrow 0$
4:      $counter2 \leftarrow 0$
5:      for each $c \in params(op)$ do
6:          $counter1++$
7:          $maxValue \leftarrow 0$
8:          for each $c' \in params(op')$ do
9:              $counter2++$
10:             if $TSM(c, c') > maxValue$ then
11:                 $maxValue \leftarrow TSM(c, c')$
12:         $total +=$ maxValue
13:     if $counter1 > counter2$ then
14:         $total = total/counter1$
15:     else
16:         $total = total/counter2$
17:     $SimValue = 0.75 * total + 0.25 * oldSimValue$
18:     return $SimValue$
```

Subsequently, the values are aggregated when we get the calibrated similarity matrices to obtain a final similarity value between two ontologies. A high value means that the newly received object o of the new type t is manageable, and an alignment can be done using the ontology with the high similarity value; otherwise, o is considered unmanageable.

5.6 Selecting the Ontology with the Highest Semantic Insight

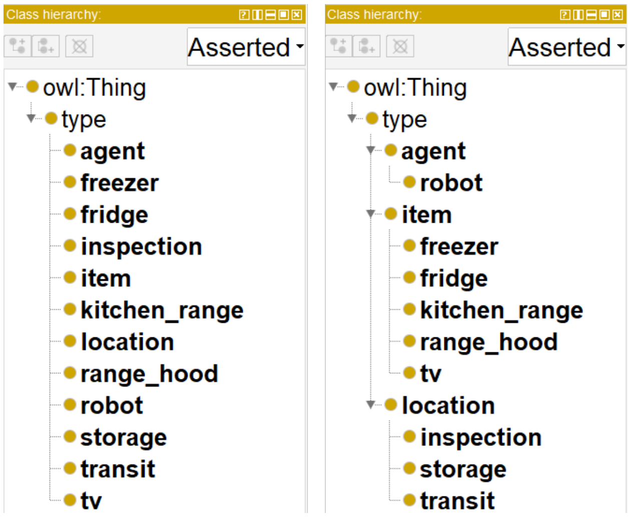
In this stage, the agent decides which ontology (η_ω) amongst the ontologies of N_ϕ (which all cover a certain domain of knowledge and model the same concepts) is the best for the specific task ϕ to perform the alignment. For this purpose, the author decided to use the semantic variance (SV filter in Figure 5.4) introduced in (Sánchez et al. 2015). SV is an intuitive and inherently semantic measure to evaluate the accuracy of ontologies. Unlike ad hoc methods, SV is a mathematically coherent extension of the standard numerical variance to measure the semantic dispersion of the taxonomic structure of ontologies. Here follows the SV formal definition introduced in (Sánchez et al. 2015). Given an ontology n, which models in a taxonomic way a set of concepts $\mathcal{C}$ in n, the SV of n is computed as the average of the squared semantic distance between each concept $\mathsf{c_i} \in \mathcal{C}$ and the taxonomic root node of n. If we denote by $|\mathsf{C}|$ the cardinality of $\mathcal{C}$ excluding root, the mathematical expression of SV of n is shown in Equation 5.1:

$$SV = \frac{\sum_{\mathsf{c_i} \in \mathsf{C}} d(\mathsf{c_i}, \mathsf{root})}{|\mathsf{C}|} \tag{5.1}$$

Where $d(\mathsf{c_i}\ \mathsf{c_j})$ is the semantic distance between two concepts $\mathsf{c_i}$ and $\mathsf{c_j}$ calculated as a function of the number of their non-common taxonomic ancestors divided (for normalisation) by their total number of ancestors. Equation 5.2 shows the semantic distance $d(\mathsf{c_i}\ \mathsf{c_j})$:

$$d(\mathsf{c_i}, \mathsf{c_j}) = \log_2(1 + \frac{|A(\mathsf{c_i})\ \cup\ A(\mathsf{c_j})| - |A(\mathsf{c_i})\ \cap\ A(\mathsf{c_j})|}{|A(\mathsf{c_i})\ \cup\ A(\mathsf{c_j})|}) \tag{5.2}$$

$A(\mathsf{c}_i)$ is the set of taxonomic ancestors of concept c_i in n, including itself. The semantic distance measure $d(\mathsf{c}_i\ \mathsf{c}_j)$ aggregates features in a logarithmic way, which better correlates with the non-linear nature of semantic evidence. More importantly, variance does not depend on the cardinality of the ontology. We calculate the SV for each ontology in N_ϕ, and select η_ω, the one with the highest SV. The goal of the stage can be loosely described as distinguishing the most hierarchical ontology and avoiding non-specialised ontologies where different categories of concepts (types) are flat as siblings.



- owl:Thing
  - type
    - agent
    - freezer
    - fridge
    - inspection
    - item
    - kitchen_range
    - location
    - range_hood
    - robot
    - storage
    - transit
    - tv

- owl:Thing
  - type
    - agent
      - robot
    - item
      - freezer
      - fridge
      - kitchen_range
      - range_hood
      - tv
    - location
      - inspection
      - storage
      - transit

Figure 5.8: Two ontological representations that model the same knowledge domain

Figure 5.8 shows two ontological representations that model the same knowledge domain. Each has the same concepts; However, the semantic variance of the ontology on the left side of the figure is equal to 0.253, whereas it is equal to 0.289 for the ontology on the right side of the figure as it has a deeper hierarchy and it better describes that particular domain of knowledge.

5.6.1 Alignment with neighbourhood constraint

The agent performs an alignment to determine where to position the concept ω representing the new type t within the hierarchy of concepts in η_ϕ. The alignment is performed between:

- On the one hand, Ω_T in η_ϕ, that is the taxonomic branch representing the PDDL types in the ontology.
- And on the other hand, the particular part of the taxonomic branch that includes ω in η_ω (the ontology with the highest semantic insight, that contains ω, and that is most similar to η_ϕ according to the TSM). This taxonomic branch includes the parent concept $\mathsf{parent}(\omega)$, and the sibling concepts $\mathsf{siblings}(\omega)$.

For the alignment, we used the similarity matrices (produced by TSM in Section 5.5.5) with a neighbourhood constraint. Domain-independent constraints convey general knowledge about the interaction between related nodes. Perhaps the most used constraint is the neighbourhood constraint, as suggested by (Doan et al. 2003), where "*two nodes match if nodes in their neighbourhood also match*". Therefore:

- If $\mathsf{parent}(\omega)$ in η_ω matches x in η_ϕ, then the agent establishes $\mathsf{x} :: \omega$ in η_ϕ.
- On the other hand, if no match was achieved with the parent, the neighbourhood constraint procedure matches Ω_T in η_ϕ with $\mathsf{siblings}(\omega)$ in η_ω, if the percentage of matching siblings exceeds a specified threshold, and these matched concepts are found to be under a common parent in η_ϕ, then we list ω as a subconcept of that superconcept in η_ϕ.

5.7 Discussion and Future Work

Extending planning knowledge provides context awareness to the agent. Context awareness of the execution environment in terms of relevant types and context awareness of the agent's capabilities and the task it performs regarding variables and operators.

Recognising unexpected events that bring in new information (new objects) leads to new ways of achieving the existing goals (in case of failure) or new opportunistic goals (increased utility). The work outcomes leave many open lines of research and future developments. There follow some implications and future research suggestions.

The ontological representation can be easily extended to fully represent operators' conditions and effects and the execution state. This means the agent can already represent ground actions as individual instances of their corresponding operators' classes, paving the way for the agent to learn new variables and operators. Learning new variables and operators and importing them into the agent model is very ambitious. Theoretically, it allows the agent to learn new capabilities and perform further new operations. However, practically, agents are often limited by their physical capabilities, their actuators or grippers. Therefore, concerning importing operators, it would be interesting to further extend this research to agents with cognitive tasks, such as speech, conversation or tasks involving natural language processing. In such cases, for example, an agent that is originally modelled to form noun phrases will be able to compose propositional, adjective, verb, or gerund phrases.

Similarly, it would be interesting to represent the HTN formalisation. Since HTN high-level tasks are decomposed into more straightforward tasks and primary actions, an agent that benefits from ontological and HTN representations could extend its model with new high-level tasks if it has the primary actions of that high-level task. Consequently, allowing the agent to import new strategies that were not originally modelled in the agent model, yet the agent would be able to perform.

Our approach has good performance because 1) the semantic distances and operations we use compute largely fast, and 2) that we are not using ontologies to perform planning itself but rather to extend the planning knowledge. The complexity and performance become an issue when using ontologies to perform planning, as indicated in (Gréa 2020). Another possibility is to research using Semantic Web Rule Language (SWRL) to perform planning, or to explore semantic similarity for planning heuristics by computing an overall semantic similarity score between terms that can be used by planning algorithms to guide the searching process if a way is found to achieve good performance.

In the author's work, deciding whether the newly generated goal forms an opportunity is delegated to the planner. This can be considered a rich research problem that requires further research. Some current planners remove from the search space those actions whose conditions are fluents that do not hold in the initial state (Richter and Westphal 2010). Other works, such as (Borrajo et al. 2021), consider that these facts could become true during execution and may allow for reaching the same goals pursued in the original plan more intelligently. Similarly, Goal-Driven Autonomy (GDA) approaches, and goal refinement models of goal reasoning (Aha 2018) can reason about soft goals that were not included in the original plan and become achievable during execution due to the dynamic nature of the environment. However, more research is required for the reachability heuristics when a new object of a new type is integrated into the planning task with its related fluents and a new goal that involves the newly integrated object is generated.

5.8 Conclusion

Context and context awareness are crucial for any intelligent agent that operates in a dynamic environment. In this chapter, the author has presented a domain-independent approach that may be used as a part of a context-aware model in a deliberative context-aware ambient intelligence system, as will be shown in Appendix C. Our approach bolsters an autonomous agent with the capability of extending its planning task to accommodate, on the fly, only relevant and manageable new information that may trigger the formulation of new goal opportunities as shown in Appendix B or may be used in plan repair if a failure occurs Appendix C.

At this stage of this PhD thesis, the agent can simulate and monitor the plan execution and acquire and extend its knowledge in a context-aware manner. The next step in this PhD thesis is to endow the agent with the ability to perform plan repair intelligently to ensure a responsible repairing policy among agents instead of delegating that task to a replanning as typically done.

CHAPTER 6

Plan Commitment Repair

PLAN COMMITMENT (Babli, Sapena, et al. 2023), the third and final contribution of this PhD thesis, stands out as the result of detailed and dedicated research, aimed to offer valuable insights and provoke further thought. While executing its plan in a dynamic environment where multiple agents are operating, an autonomous agent may suffer a failure due to discrepancies between the expected and actual context and thus must replace its obsolete plan. In its endeavour to fix the failure and reach its original goals, the agent may unknowingly disrupt other agents executing their plans in the same environment. We present a property for plan repair called plan commitment to ensure a responsible repair policy among agents that aims to minimise the negative impact on others. We present arguments to support the claim that plan commitment is a valuable property when an agent may have made bookings and commitments to others. We then propose C-TFLAP, an implementation of a plan repair heuristic that allows adapting a failed plan to the new context while committing as much as possible to the original plan. We demonstrate empirically that: (1) our plan repair achieves more committed plans than plan-stability repair when an agent has made bookings and commitments to others, and (2) compared to typical replanning and plan-stability repair, it can reduce the revisions among agents when failures are avoidable and can decrease the time-loss otherwise. In addition, to demonstrate extensibility, we integrate context-aware knowledge extension with committed repairing to increase the agent's chances of repairing.

A Brief Evolutionary Perspective on Commitment

Since AI was born in 1956, it has held great enthusiasm by embracing the idea of replicating human faculties. This unnumbered section pre-motivates the reader on the roots of commitment drawing upon evolution, adaptation and the capacity for commitment. Here follow some of the most influential quotes about the essence of adaptation and commitment.

> "*It is not the most intellectual of the species that survives; it is not the strongest that survives; but the species that survives is the one that is best able to adapt and adjust to the changing environment in which it finds itself.*" – Leon C. Megginson (Megginson 1963)

> "*In the struggle for survival, the fittest win out at the expense of their rivals because they succeed in adapting themselves best to their environment.*" – Ritchie R. Ward (Ward 1972)

> "*In a multiperson situation, one man's goals may be another man's constraints*" – Herb Simon (Simon 1964)

In a TED talk (De Waal 2017) at an official TED conference, Frans de Waal, a distinguished professor, evolutionary biologist and primatologist known for his work on the behaviour, empathy, cooperation and social intelligence of primates (De Waal and Waal 2007), reveals that the terms alpha male and alpha female which mean the highest ranking members goes back to the ’40s and ’50s research on wolves, recently became very popular and are heavily used in mischaracterisation in a way that does not relate to what an alpha male or female actually is – a bully – whereas, in fact, the most critical obligation in alpha members is that they behave responsibly, show empathy and help others.

Regarding our evolutionary origins, social relations, and society’s organisation, Fehr Ernst and Urs Fischbacher in (Fehr et al. 2003) argue that altruism and selfishness are among the most fundamental questions, and experimental evidence indicates that human altruism is an extremely powerful force.

In (Nesse 2007), the author explains that commitment has roots in regular natural selection at the individual level by selection forces that arise from the actions of other individuals, emerging automatically from the dynamics of social groups. In contrast to individuals whose behaviour is predictably self-interested, those who are capable of making and keeping commitments to do things that are not in their direct interests gain advantages through social influence.

The social fabric is woven from promises and threats (commitments) that are not always immediately advantageous to the parties involved. Furthermore, game theorists have shown that players who use commitment strategies achieve greater success than those who rationally calculate every move for immediate reward (Nesse 2001).

6.1 Introduction

Automated planning is the model-based approach for autonomous control devoted to studying the reasoning side of acting (Russell et al. 2010). Planning is a crucial deliberative capability for autonomous agents to synthesise plans of action that achieve their goals in a variety of contexts like e-learning (Garrido et al. 2008), space applications (Chien et al. 2014), robotics (Karpas et al. 2020) and assisted living homes (Babli, Rincon, et al. 2021).

The successful application of automated planning requires looking into the side of acting, that is, examining the results of executing a plan in the real world (Ghallab et al. 2014). Broadly speaking, there are two general ways of addressing planning and execution. The first one is to use models for planning under uncertainty that capture the partial observability and inherent uncertainty of the environment so that the calculated plans account, to some extent, for potential incidents and exogenous events that can occur during execution (Castellini et al. 2021; Spaan et al. 2015; Ong et al. 2010). The second way lies in using classical planning models that assume a static environment only changing when some action is taken, a full observability and determinism of the actions (Goldman, Kuter, and Freedman 2020; Rodríguez et al. 2011; Bajo et al. 2008; Koenig et al. 2005; Vendrell et al. 2001). Under this paradigm, the effects of unexpected situations during the plan execution are tackled at execution time. The first scheme uses more expressive but also more expensive planning models that can only partially account for the uncertainty in the world, whilst the second scheme relies on efficient replanning or plan repair methods. In this work, we adopt the second scheme and develop a novel plan repair strategy among multiple agents sharing the same execution environment.

A planning agent is an entity with the capacity for reasoning and acting. Ideally, a given planning agent would be responsible for synthesising and executing plans and replanning to account for changes to the world state that the agent cannot foresee. However, there are many such agents in the real world, each with different objectives, yet all tied together by a common execution environment they share. We follow the central argument presented in (Talamadupula, Smith, et al. 2013) that the single-agent planning community needs to heed the changes to the world state by generating a new (single-agent) plan that remains consistent with the larger community of agents sharing the world.

We aim to ensure the robust execution of an agent's plan in a community of agents so that it can be adapted to unforeseen situations arising from discrepancies between what an executing agent expects and what it observes. More importantly, ensure that the new adapted plan does not propagate negative consequences to other agents operating in the same environment.

Replanning consists of generating a new plan from scratch without considering the original plan by calling a planner with the new encountered world situation. Replanning is a popular method concerned with allowing an agent that suffered a failure to generate a new plan to reach its goals quickly. However, it derives a new solution without considering the existing plan. By contrast, plan repair fixes the existing plan, tries to retain its parts and structure and adapts it to the new context whilst causing minimal perturbations. In theory, in the worst case, modifying an existing plan is not more efficient than a complete replanning (from the current state) (Nebel et al. 1995). However, in practice, much prior research effort (Bechon et al. 2020; Chen, Xu, et al. 2020; Krogt et al. 2005; Gerevini and Serina 2000) indicates that plan repair can be more effective than replanning when the plan duration (the makespan of the plan) is not the main criterion to optimise.

Since the plan-stability concept was defined in 2006, plan repair has proved that it could provide new plans with equivalent computation speed and with fewer revisions than replanning in the face of disruptions (Goldman, Kuter, and Freedman 2020; Fox, Gerevini, et al. 2006). Albeit there is a wealth of research on plan repair, there is arguably less work on repair strategies that aim to minimise the negative impact among multiple agents sharing the same execution environment, where there is a degree of privacy, reduced communication, or when an agent has bookings and commitments to others that may be costly to undo; for brevity, we will call such a setting the conservative shared setting.

Herein, we identify a new metric, plan commitment, which we claim is important in a community of agents operating in a conservative shared setting. In this situation, if an agent is solely focused on achieving its goals within the minimum time, it may propagate negative consequences to other agents and disrupt the community. Therefore, the repaired plan should be as close as possible to the original plan from the perspective of commitments made to other agents whilst achieving the original goals. When we refer to commitments in the context of a repaired plan, we are specifically referring to reusing the resources that were used in the original plan. The results of this work show that in comparison to replanning and plan-stability repair, an agent that is using plan commitment to repair its plan failures can reduce the adverse effects on other agents, i.e., can cause fewer failures to other agents in a community of agents and can reduce their time-loss if the failures are inevitable.

Our approach builds on the temporal forward partial-order TFLAP planner (Sapena, Marzal, et al. 2018) but exploits the new plan commitment quality as a search strategy by seeding the search with the existing plan and using the new heuristic to guide the search for the most committed repair plan. Our contributions are the plan commitment property, the corresponding heuristic that guides the planner's search to achieve the most committed plan repair, and the resulting planner, which we call C-TFLAP, that repairs failures while trying to minimise the negative impact on others by generating the most committed plan when a failure occurs.

This Chapter is structured as follows. The next section reviews handling plan execution failures in the literature. The plan commitment property is defined in Section 6.3, and in Section 6.4, we show the implementation details of C-TFLAP. Next, we present an experimental evaluation and discuss the results in Section 6.5. Finally, Sections 6.6 and 6.7 finish the chapter by showing future extensions and drawing some conclusions.

6.2 Related Work

The integration of planning and acting is a critical issue in designing deliberation systems, thus resulting in a range of design options for them. The first view relies entirely on planning. The second view depends mostly on execution without much efficient help from planning, whereas the third view is balanced between planning and acting (Ingrand et al. 2017).

For our purposes, this section examines some approaches in two categories. The first category uses predictive models while planning (before execution). The second category deals with failures without predictive models during execution (online interleaved with execution).

For the *first category*, planners need to integrate predictive models with mission specifications and provide a plan or policy as output, transforming perceived states (or beliefs) into lower-level orders, guaranteeing the mission's success and potentially meeting some optimality. Partially Observable Markov Decision Process (POMDP), contingent planning and conformant planning are ways to plan under uncertainty. In POMDP, agents use observations to form a belief of the current state, expressed as a probability distribution over the states, to find a policy prescribing which action is optimal for each belief state (Bai et al. 2014). Contingent planning generates conditional plans given uncertainty about initial state and action effects, where the right branch will be selected during the execution through sensing (Hoffmann and Brafman 2005). The conformant planning problem can be formulated as a path-finding problem in the belief space where a sequence of actions maps a given initial belief state into a target belief (Palacios et al. 2009). Other approaches use a tri-valued logic to account for the unknown items of the world and follow an online anytime planning algorithm that allows combining planning and execution simultaneously (Sapena and Onaindia 2008). Generally, the low predictability of most real-world environments, the immense difficulty in specifying necessary predictive models, and the computational complexity of planning under uncertainty still constrain the effectiveness of such approaches.

The *second category* of systems resulted from addressing the previous concerns by decomposing the deliberation burden into planning and acting phases, using restrictive assumptions in either phase and shifting the focus towards acting. This second category includes various approaches such as replanning, explanation-based plan repair and adaptation, replanning within limited time windows, refinement and unrefinement and iterative repair. In addition, several related works that tackle execution failures for multiple agents are reviewed for the second category. Subsequently, the remainder of this subsection delves into a comprehensive examination of several techniques and methods that fall within the scope of the second category.

Re-invoking a planner (replanning) is straightforward. However, it disregards the resources used in the existing plan and is only eager to generate a new plan to reach the original goals. On the other hand, plan repair involves adapting the existing plan to the actual execution context and making it executable again with minimal perturbation. Plan stability repair has been empirically shown to produce more stable plans than replanning (Fox, Gerevini, et al. 2006). CHEF (Hammond 1990) and GERRY (Zweben et al. 1993) are two early approaches to plan repair. In CHEF, plan failures are described as causal explanations of why they occurred and are used to access abstract replanning strategies, which are then turned into changes to the faulty plans. GERRY starts with a complete but flawed schedule and then uses a scheduling and rescheduling constraint-based iterative repair to improve solutions. The plan adaptation approach (Gerevini and Serina 2000) combines modifying the existing plan by replanning within limited temporal windows and using a local search for action graphs. The refinement and unrefinement approach (Krogt et al. 2005) repairs plans by removing the obstructing actions, adding new actions to achieve the goals, and adapting planning heuristics to remove constraints.

The Hierarchical Task Network (HTN) RepairSHOP (Warfield et al. 2007) and the PANDA hybrid planner (Bidot et al. 2008) consider failed traces during plan adaptation, replay the reasoning that led to the initial decision and select a different path. The forwarding dock state approach (Chen, Xu, et al. 2020) collects the state information about the agent being monitored at different time intervals, including the current state, the near future state, and upcoming actions and tries to find a path from the regressed state to the current state through a breadth-first search. Regression in planning, which dates back to PLANEX (Fikes, Hart, et al. 1972), is a process of finding a path that applies actions backwards from the goals to the current or initial state. A regressed state is a partial expected state which contains the minimal set of conditions that must hold in the world at a specific time instance for the rest of the plan to be executable during execution and for the goals to be satisfied. SHOPFIXER (Goldman, Kuter, and Freedman 2020) uses a graph of causal links and task decompositions to identify a minimal subset of the plan that must be fixed.

The robust plan execution with unexpected observations approach employs a runtime action selection policy that dynamically selects causally-valid actions that are likely to reach the goal (Lima et al. 2020). This effectively moves the causal reasoning online instead of limiting the executor to blindly following the plan's causal structure. Although there are some similarities in purpose, our work addresses a different problem than the robust plan execution with unexpected observations approach. Their approach focuses on minimising the number of failures experienced by a single agent and reducing its need for replanning. In contrast, our work involves a community of agents where each agent must adjust its plan in the event of failure without causing disruptions for the entire community. Our objective is to minimise the need for plan revisions that impact the community as a whole. Another difference is that their approach provides solutions for a single planning problem with multiple robots possibly having concurrent actions. In contrast, in our case, different agents have different planning problems with different private goals but use some shared resources of the environment. In our work, agents operate in the same execution environment and use some shared resources. No collaboration or organisation between agents is needed. Agents devise and execute their plans to achieve their private goals. An agent communicates information about shared resources after it generates its plan, and no further communication is made to assume reduced communication settings. Our approach is independent of how agents communicate (agents can communicate with each other directly, through a mediator or in any other way). If an agent's plan fails (because of a change in the environment or incompatible information received from another agent), it repairs its plan (as a single agent), aiming to minimise the impact on other agents.

It is worth mentioning that there are other approaches, such as the situational evaluation and awareness (SEA) framework (Carreno et al. 2021), which integrate both replanning and plan repair as valid options depending on the characteristics of the failure.

There have been several approaches to solving execution failures for multiple agents, such as centralised or distributed multi-agent planning (MAP) and intra-agent planning. The hybrid planning distributed approach (Bechon et al. 2020) uses refinement and unrefinement. It repairs the plan by iteratively removing actions to amend the global plan of multi-robot missions involving heterogenous robots cooperating autonomously to achieve a common task. The repair approach in (Komenda et al. 2014) either attempts to preserve a suffix of the existing plan and prefixes it with a newly computed plan or preserves the plan's remainder and closes the gap between the state resulting from the failed plan execution and the original goal state. Their decentralised algorithms produce a synchronous multi-agent plan or plan repair that is decomposed among agents, assuming some transparency or collaboration.

In the architecture for iterative plan repair in hierarchical multi-agent systems (HIPR) (Mohalik et al. 2018), the authors base their work on two ideas that agents are organised hierarchically based on attributes like location/administration and that few agents are affected by the failure. The agents are assumed to be collaborating to solve a task by decomposing global plans into local plans for individual agents. There are operational agents at the leaf level to provide operational capabilities in their work. The agents at higher levels, called managing agents, are responsible for deciding the operational agents' actions and coordination among them. This requires additional communication for coordination signalling between agents and extra computations to determine dependencies between agents, the contingency-free plan segments, and the agents affected by a hazard.

The partial satisfaction planning (PSP) intra-agent repair approach (Talamadupula, Smith, et al. 2013; Cushing et al. 2005) describes three replanning paradigms distinguished by the constraints they try to satisfy during the re-planning process. These paradigms are replanning as restart, replanning to reduce computation and replanning for multi-agent scenarios. The authors in that approach refer to these constraints also as commitments. However, in their approach, the agent must publish its plan before putting it to execution, communicating it to other agents, who in turn must inform that agent of the constraints they wish to book. Hence, it knows in advance what constraints it must try to maintain if a failure occurs during execution. Consequently, all these constraints, possibly for multiple agents, must be coded and maintained as a list of new soft goals and will be added to the new problem after the failure.

The closest approach from the literature to our work is plan-stability repair (Fox, Gerevini, et al. 2006), which defines a plan measure called the plan-stability distance to keep the repaired plan as close as possible to the original plan in terms of actions. In contrast, our approach attempts to maintain the repaired plan close to the original plan in terms of resources booked in the original plan.

6.3 Plan Commitment

This section defines a novel distance named *plan commitment*, explains how it differs from action-based measures and provides arguments and examples to support its significance for a community of agents in a conservative shared setting.

Two of the most influential plan measures from the literature were designed to maintain plan stability and foster plan diversity:

- Plan stability distance $d(\pi', \pi)$ (Fox, Gerevini, et al. 2006) between two plans π' and π is defined as the number of actions that appear in a plan π' and not in a plan π, plus the number of actions that appear in π and not in π'. The lower the plan stability distance value is, the more stable and closer it is to the reference plan π. A plan π' achieves better plan stability than a plan π'' if $d(\pi', \pi) < d(\pi'', \pi)$. One of the plan stability arguments is that it is intended to maintain stability between agents, making their interaction potentially predictable and easier. However, that informal argument was not tested empirically in a community of agents.
- Action distance $ad(\pi', \pi)$ (Nguyen, Do, et al. 2012; Srivastava et al. 2007) between two plans π' and π, is defined as the ratio of the number of non-shared actions between the new plan π' and the old plan π to the total number of actions appearing in one of them.

These distances are sensitive to changes in the level of actions and are therefore valuable plan measures. However, we show later in this study that plans that are equally distant from the original plan (actions wise according to these measures) may not lead to the same disturbance levels. Despite the plans being equally distant from the original plan, some of them cause fewer disturbances among agents, leading to fewer failures (revisions) and less wasted time in the community of agents.

We propose a new plan metric we call "plan commitment" distance $c(\pi', \pi)$, which measures the distance of a new plan π' to an original (old) plan π in terms of objects. The lower the plan commitment distance, the closer the new plan is to the original. A plan π' achieves better plan commitment than a plan π'' if $c(\pi', \pi) < c(\pi'', \pi)$. In essence, the plan commitment distance measure is similar to plan stability and action distance measures; however, it differs from the previously mentioned distances measures in two aspects:

- Object-sensitive: plan commitment distance works on the level of objects rather than actions.
- Type-sensitive: plan commitment distance takes into consideration the types of objects that are used in repairing.

6.3.1 Plan Commitment as a Distance

After an execution failure occurs, we do not have a complete repair plan upfront, and we will generate a new repair plan. To get a new plan π' that resembles the original one, π, we use our proposed distance metric based on the commitment-distance concept. First of all, we define two operators (intersection and union) that will allow us to compare how similar two actions are with respect to this metric:

Definition 6.3.1 (Intersection between two actions)**.** *We define the intersection operator $\tilde{\cap}$ between two actions a' and a as a measure of their common characteristics. These characteristics include the objects in the action parameters, the parameter types and the operator names. The algorithm for calculating $a' \ \tilde{\cap} \ a$ can be seen in Algorithm 3.*

Definition 6.3.2 (Union between two actions)**.** *We define the union operator $\tilde{\cup}$ between two actions a' and a as a measure of the total number of distinct characteristics that can be found in these actions. These characteristics include the objects in the action parameters and the operator names. This union function also prevents the distance from being negative when an object appears multiple times as a parameter in an action, as is the case in some domains. The calculation of $a' \ \tilde{\cup} \ a$ is shown in Algorithm 4.*

Our goal now is to be able to assess the appropriateness of an action a', in comparison to the original plan π, before proceeding to add it to the repaired

Algorithm 3 Intersection operator between two actions

Input: the new action a' and the original action a

Output: the intersection value

```
1: procedure $a' \tilde{\cap} a$                                   ▷ The intersection of action $a'$ and $a$
2:     $intersectedObjects \leftarrow 0$
3:     for each $o \in params(a)$ do                              ▷ Parameters of action $a$
4:         $counter \leftarrow 0$
5:         for each $o' \in params(a')$ do                        ▷ Parameters of action $a'$
6:             if $o = o'$ then                                   ▷ Identical objects
7:                 $counter \leftarrow 1$
8:             else if $type(o) = type(o')$ then                  ▷ Identical types
9:                 $counter \leftarrow \max(counter, 0.5)$
10:        $intersectedObjects += counter$                        ▷ $o$ and its closest object in $a'$
11:    if $op(a') = op(a)$ then                                   ▷ Instances of the same operator
12:        $intersectedObjects += 1$
13:    return $intersectedObjects$                                ▷ Returns the value of the intersection
```

Algorithm 4 Union operator between two actions

Input: the new action a' and the original action a

Output: the union value

```
1: procedure $a' \ \tilde{\cup} \ a$                ▷ The union of action $a'$ and $a$
2:     $h \leftarrow \phi$                          ▷ Define a list that allows duplicates
3:     for each $o' \in params(a')$ do              ▷ Parameters of the new action
4:         $h.append(o')$                           ▷ Add the object name
5:     for each $o \in params(a)$ do                ▷ Parameters of the original action
6:         if $o \notin h$ then                     ▷ If name of object $o$ does not exist
7:             $h.append(o)$                        ▷ Add the object name
8:     $h.append(op(a'))$                           ▷ Add the operator of action $a'$
9:     if $op(a) \notin h$ then                     ▷ If operator of $a$ is different
10:        $h.append(op(a))$                        ▷ Add it to the list
11:    return $|h|$                                 ▷ Returns the length of the list
```

plan π'. For this, we define a commitment distance function between an action a' and the original plan π. This function takes into account the actions' parameters (objects) and their types and their operators' names to favour repair actions that use the same objects over those that use different objects of the same type and those that use different objects of different types.

Definition 6.3.3 (Commitment distance between an action and a plan)**.** *We define the commitment distance between an action a' and a plan π, $\delta(a', \pi)$, as the distance that separates a' from the closest action of π, as Equation 6.3.1 shows. shows. $\delta(a', \pi)$, has a value between 0 and 1. Lower values indicate greater similarity.*

$$\delta(a', \pi) = \min_{a_i \in \pi} \left[1 - \frac{a' \ \tilde{\cap} \ a_i}{a' \ \tilde{\cup} \ a_i} \right] \tag{6.1}$$

We consider the commitment distance $\delta(a', \pi)$ to be the key distance in our work as it will be used during heuristic evaluation to find good quality repair plans. Finally, we proceed to the definition of the commitment of a new plan π' (assuming it is either given, generated by plan repair or by replanning) to the original plan π.

Definition 6.3.4 (Commitment distance between two plans)**.** *The commitment distance of a plan π' to a plan π, $c(\pi', \pi)$, is the sum of commitment distances between the actions of π' and π, divided by the number of actions of π' (see Equation 6.2).*

$$c(\pi', \pi) = \frac{\sum\limits_{a'_i \in \pi'} \delta(a'_i, \pi)}{|\pi'|} \tag{6.2}$$

In general, plans that utilise the original plan's objects will have the best commitment value (closest to zero). Plans that do not utilise the original objects but utilise different objects of the same type will have a slightly worse commitment value. Plans with different objects of different types will have a worse commitment value.

6.3.2 The Value of Plan Commitment

We present two informal arguments to demonstrate the significance of the plan commitment metric in autonomous systems. The first argument focuses on the impact of plan commitment on human interaction. The second argument, which we test empirically, focuses on the impact of plan commitment on interactions among agents.

- Significance for human interaction: users are confused by changes to plans in response to upsets. This is subjective to each user; however, trivially, change interferes with autonomy, can confuse users and make them feel they have lost control. The significance of plan commitment in human interaction is related to the negative effects of plan changes on user satisfaction and trust. Such changes can cause confusion and a loss of autonomy, leading to dissatisfaction and mistrust in the autonomous system. By addressing user confusion and preserving autonomy through plan commitment, the design and implementation of autonomous systems can better serve and meet user needs.

- Significance for other agents: in a reduced communication environment, an agent may communicate information to other agents only at the beginning when it generates its plan; these agents, in turn, use that information when generating their own plans to reach their goals. An agent repairing its plan using plan commitment can cause fewer disruptions to other agents. Plan commitment makes the agents more predictable and aims for smooth interactions among agents. Our approach favours repairing plans using the same objects where possible or using different objects of the same types if the same objects are no longer available; otherwise, different objects of different types just to satisfy the goal. We test this informal argument empirically and show that it can reduce the number of revisions among agents and can reduce the time-loss caused by inevitable failures, as the results describe in Section 6.5.

Sub-optimal metrics or heuristics in planning usually do not have properties that can be formally proved; they can work better or worse depending on each specific case. For this reason, like many other metrics in planning, such as stability (Fox, Gerevini, et al. 2006), diversity (Nguyen, Do, et al. 2012), dependency satisfaction (Talamadupula, Smith, et al. 2013), and causal similarity (Joslin et al. 1994), we have opted for an experimental evaluation, from which we can extract statistics and behaviour trends of our approach in comparison with other existing ones.

To prevent bias, 60 trials were performed by randomising 20 problems for two agents in three benchmark domains in a conservative shared setting. The domains are a multi-agent logistics system, a multi-rover system and a multi-robot navigation system. The problems are generated by modifying the initial state, the goals, the planner's starting seeds and the failures. The modifications were generated randomly. To ensure that we were not artificially enhancing the commitments by relying on how a planner explores its search space, the original plans were calculated by a different planner. In our experiments, the original plans were generated by OPTIC, replanning was done via LPG-TD, stability repair was done via LPG-ADAPT and commitment repair was done via C-TFLAP.

We consider a motivating example domain for logistics of a package-delivery problem with drivers, trucks and packages that must be delivered to locations. The actions of the domain allow drivers to walk between locations, load packages to vehicles, board, drive, disembark from vehicles and unload packages. First, we present a simple single-agent base scenario to demonstrate the calculations of the commitment distance. More importantly, to show that two plans can be equally distanced from the original plan according to the plan-stability distance and the action distance (explained previously in this section) yet the commitment distance can distinguish them according to the resources used in each plan.

Fig. 6.1 shows an original plan π_0 for a single-agent simple logistics planning task in which `package0` must be delivered to `location2`, and two candidate plans π'_0 and π''_0 to fix a failure due to `truck0` becoming unavailable.

Table 6.1 shows how the commitment distance $c(\pi'_0, \pi_0) = (\delta(a'_1, \pi) + \delta(a'_2, \pi) + \delta(a'_3, \pi) + \delta(a'_4, \pi) + \delta(a'_5, \pi))/5 = 0.270$ is calculated.

	a_1	a_2	a_3	a_4	a_5	
a'_1	$1-4.5/6$	$1-2.5/7$	$1-3/8$	$1-2/8$	$1-3/8$	$\delta(a'_1, \pi) = 0.25$
a'_2	$1-2.5/7$	$1-3.5/5$	$1-3/7$	$1-2/7$	$1-2/8$	$\delta(a'_2, \pi) = 0.30$
a'_3	$1-2.5/8$	$1-2.5/7$	$1-4.5/6$	$1-2.5/7$	$1-2.5/8$	$\delta(a'_3, \pi) = 0.25$
a'_4	$1-2/8$	$1-2/7$	$1-3/7$	$1-3.5/5$	$1-2.5/7$	$\delta(a'_4, \pi) = 0.30$
a'_5	$1-3/8$	$1-2/8$	$1-3/8$	$1-2.5/7$	$1-4.5/6$	$\delta(a'_5, \pi) = 0.25$

$c(\pi'_0, \pi_0) = 0.270$

Table 6.1: Calculating the commitment distance of π'_0 to π_0

In action-sensitive distances such as the plan stability and the action distances, when one object changes in an action, the whole action is considered different. Consequently, once one object changes, action-sensitive distances cannot further distinguish the quality of the plans, no matter whether the change is small or large. Plan commitment distinguishes cases that the plan stability and action distances cannot distinguish. According to the makespan

a_1 : 0.000 : $(load\ package0\ driver0\ truck0\ location0)$[17.000]
a_2 : 17.001 : $(board\ driver0\ truck0\ location0)$[10.000]
a_3 : 27.002 : $(drive\ driver0\ truck0\ location0\ location2)$[300.000]
a_4 : 327.003 : $(disembark\ driver0\ truck0\ location2)$[10.000]
a_5 : 337.004 : $(unload\ package0\ driver0\ truck0\ location2)$[17.000]

(a) An original plan π_0.

a'_1 : 0.000 : $(load\ package0\ driver0$ **truck2** $location0)$[17.000]
a'_2 : 17.001 : $(board\ driver0$ **truck2** $location0)$[10.000]
a'_3 : 27.002 : $(drive\ driver0$ **truck2** $location0\ location2)$[300.000]
a'_4 : 327.003 : $(disembark\ driver0$ **truck2** $location2)$[10.000]
a'_5 : 337.004 : $(unload\ package0\ driver0$ **truck2** $location2)$[17.000]

(b) Candidate plan π'_0.

a''_1 : 0.000 : $(load\ package0$ **driver2 truck2** $location0)$[17.000]
a''_2 : 17.001 : $(board$ **driver2 truck2** $location0)$[10.000]
a''_3 : 27.002 : $(drive$ **driver2 truck2** $location0\ location2)$[300.000]
a''_4 : 327.003 : $(disembark$ **driver2 truck2** $location2)$[10.000]
a''_5 : 337.004 : $(unload\ package0$ **driver2 truck2** $location2)$[17.000]

(c) Candidate plan π''_0.

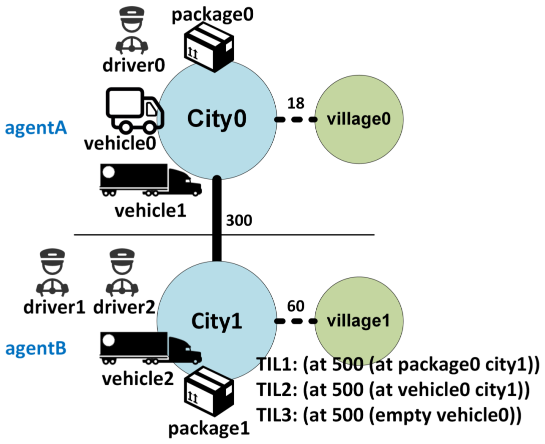
Figure 6.1: The temporal plans π_0,π'_0,π''_0 in a simple single-agent logistics planning task. The number preceding the name of each action is its start time, and the number at the end (in square brackets) is its duration.

metric, the stability and action distances measures π_0' and π_0'' are equally good and equally distanced from π_0. Both makespans are equal to 354. Stability distances are $d(\pi_0', \pi_0) = d(\pi_0'', \pi_0) = 10$ and action distances are $ad(\pi_0', \pi_0) = ad(\pi_0'', \pi_0) = 1$. On the other hand, according to plan commitment $c(\pi_0', \pi_0) = 0.270$, $c(\pi_0'', \pi_0) = 0.457$, as π_0'' has two objects different than π_0 in several actions (`truck2` and `driver2`), whereas π_0' has only one object different (`truck2`).

Such plan distances (including ours) are to make the interaction between the agents potentially predictable and more manageable and promote minimum perturbation. Therefore, they must be looked at in a community of agents' context and not solely in one agent. Our approach is independent of the way agents communicate and share information. However, for the purpose of explaining the example, we will assume that an external entity tracks the movements of the packages, keeping a repository where the agents can consult this information. The rest of the information is private unless an agent decides to share it with another agent. Initially, each agent has private goals and builds a private plan to achieve them. If they do not have enough information to generate the plan, they wait until this information is available in the repository or received from other agents. If this information represents facts expected to occur in the future, they will be modelled as Timed Initial Literals (TILs). There follows a representative yet a straightforward example of the importance of plan commitment in preventing an agent, repairing its plan, from disrupting another agent in a more elaborate logistics problem.

Fig. 6.2 shows a community of two agents and their plans for the logistics task.

In Fig. 6.2a, we have two agents (`agentA`, `agentB`), two cities (`city0,city1`), two villages (`village0`, `village1`), a fleet of vehicles that can be shared between agents, and contains two types: vans (`vehicle0`) and trucks (`vehicle1`,



(a) Logistics problem illustration.

a_1 : 0.000 : $(load\ package0\ driver0\ vehicle0\ city0)[17.000]$

a_2 : 17.001 : $(board\ driver0\ vehicle0\ city0)[10.000]$

a_3 : 27.002 : $(drive\text{-}van\ driver0\ vehicle0\ city0\ city1)[300.000]$

a_4 : 327.003 : $(disembark\ driver0\ vehicle0\ city1)[10.000]$

a_5 : 337.004 : $(unload\ package0\ driver0\ vehicle0\ city1)[17.000]$

(b) `agentA` original plan π_a.

a_1 : 0.000 : $(load\ package1\ driver1\ vehicle2\ city1)[17.000]$

a_2 : 17.001 : $(board\ driver1\ vehicle2\ city1)[10.000]$

a_3 : 27.002 : $(drive\text{-}truck\ driver1\ vehicle2\ city1\ city0)[300.000]$

a_4 : 327.003 : $(disembark\ driver1\ vehicle2\ city0)[10.000]$

a_5 : 337.004 : $(unload\ package1\ driver1\ vehicle2\ city0)[17.000]$

a_6 : 500.001 : $(load\ package0\ driver2\ vehicle0\ city1)[17.000]$

a_7 : 517.002 : $(board\ driver2\ vehicle0\ city1)[10.000]$

a_8 : 527.003 : $(drive\text{-}van\ driver2\ vehicle0\ city1\ village1)[60.000]$

a_9 : 587.004 : $(disembark\ driver2\ vehicle0\ village1)[10.000]$

a_{10} : 597.005 : $(unload\ package0\ driver2\ vehicle0\ village1)[17.000]$

(c) `agentB` original plan π_b.

Figure 6.2: A community of two agents and their plans for the logistics task.

vehicle2). driver0 belongs to agentA, and driver1, driver2 both belong to agentB. An essential restriction in this domain that may cause a disturbance between agents is that a van can traverse any road (cities and villages), whereas a truck can only traverse between cities; Fig. 6.2a shows this restriction as bold and dotted connections, respectively.

agentA has a private goal (at package0 city1). agentA constructs a private plan π_a (shown in Fig. 6.2b) and shares a piece of public information that the end location of vehicle0 at time instance 500 is city1. agentB knows that, at time instance 500, package0 will be at city1, vehicle0 will be empty and at city1. These facts are represented through Timed Initial Literals (TILs).

In addition, agentB has two private goals (at package1 city0) and (at package0 village1). Therefore, agentB constructs its private plan π_b (shown in Fig. 6.2c), making use of vehicle2 to deliver package1 to city0 and using vehicle0 (the vehicle of type van agentA used) to deliver package0 to village1. As seen in these settings, agentA constructed its plan to reach its own private goals and communicated the shared information (in this case about vehicle0) after planning. No further communication is made to assume reduced communication settings, as explained in Section 6.1. agentB used that information to build its plan to reach its own private goals.

We are interested in showing two scenarios as examples of the advantages of plan commitment in comparison with replanning and plan-stability repair:

1. The first scenario shows that plan commitment can reduce the number of failures (revisions) among agents.
2. The second scenario shows that plan commitment can decrease the time-loss resulting from inevitable failures.

Scenario 1: agentA suffers a failure due to the discrepancies of driver0 is no longer being available, vehicle0 being at a close location village0 instead of city0, and a new driver, driver3 is available at city0. Whether agentA performs replanning or plan-stability repair (same result in this case), it will generate π''_a, which uses vehicle1 (shown in Fig. 6.3b) to achieve the original goal. Consequently, agentA will cause a failure to agentB at time instance 500.001 as it can no longer execute the a_6 of π_b resulting in the situation shown in Fig. 6.4b. agentB will perform replanning or repairing, which will cause it to suffer a time-loss of 368 time units, in addition to a negligible processing time for invoking repairing or replanning (or even worse as a 708 time units if driver1 is no longer available and driver2 is used). This 368 lost time is due to the fact that agentB was unable to use the truck (vehicle1) on the road between city1 and village0, and had to retrieve the van when it was discovered to be in the wrong location at time instance 500.001. On the other hand, if agentA had chosen to repair its plan respecting plan commitment using π'_a, which utilises the van vehicle0 (shown in Fig. 6.3a) agentB would have found everything as expected as shown in Fig. 6.4a, and would have succeeded in executing π_b. The plan commitment $c(\pi'_a, \pi_a) = 0.40$ whereas $c(\pi''_a, \pi_a) = 0.57$, the commitment repaired plan π'_a is more committed. Although plan-stability repair (through LPG-ADAPT) has not generated a plan like π'_a, let us assume that it did; still, it is going to favour plan π''_a since $d(\pi'_a, \pi_a) = 9 + 5 = 14$ and $d(\pi''_a, \pi_a) = 5 + 5 = 10$ and agentB will suffer a failure.

Scenario 2: agentA suffers a failure due to the discrepancies driver0 and vehicle0 are no longer available, a new van vehicle3 is at a close location village0, and a new driver, driver3, is available at city0. Similarly, whether agentA performs replanning or plan-stability repair (same result), it will generate π''_a, which uses vehicle1 (shown in Fig. 6.3b) to achieve the original goal. Consequently, agentA will cause a failure to agentB at time instance 500.001

a'_1 : 0.002 : ($walk$ **driver3** $city0$ $village0$)[50.000]

a'_2 : 50.010 : ($board$ **driver3** $vehicle0$ $village0$)[10.000]

a'_3 : 60.020 : ($drive\text{-}van$ **driver3** $vehicle0$ $village0$ $city0$)[18.000]

a'_4 : 78.030 : ($disembark$ **driver3** $vehicle0$ $city0$)[10.000]

a'_5 : 88.040 : ($load$ $package0$ **driver3** $vehicle0$ $city0$)[17.000]

a'_6 : 105.050 : ($board$ **driver3** $vehicle0$ $city0$)[10.000]

a'_7 : 115.060 : ($drive\text{-}van$ **driver3** $vehicle0$ $city0city1$)[300.000]

a'_8 : 415.070 : ($disembark$ **driver3** $vehicle0$ $city1$)[10.000]

a'_9 : 425.080 : ($unload$ $package0$ **driver3** $vehicle0$ $city1$)[17.000]

(a) π'_a result of commitment plan repair for `agentA`.

a''_1 : 0.003 : ($load$ $package0$ **driver3** **vehicle1** $city0$)[17.000]

a''_2 : 17.005 : ($board$ **driver3** **vehicle1** $city0$)[10.0000]

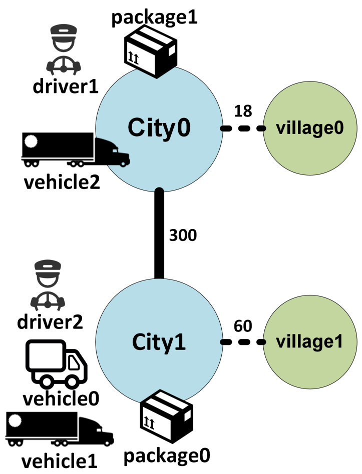
a''_3 : 27.008 : ($drive\text{-}truck$ **driver3** **vehicle1** $city0$ $city1$)[300.000]

a''_4 : 327.010 : ($disembark$ **driver3** **vehicle1** $city1$)[10.000]

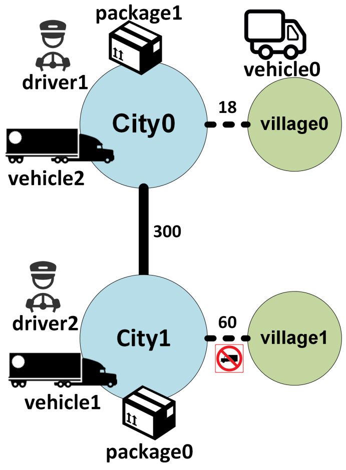
a''_5 : 337.013 : ($unload$ $package0$ **driver3** **vehicle1** $city1$)[17.000]

(b) π''_a result of replanning or plan-stability repair for `agentA`.

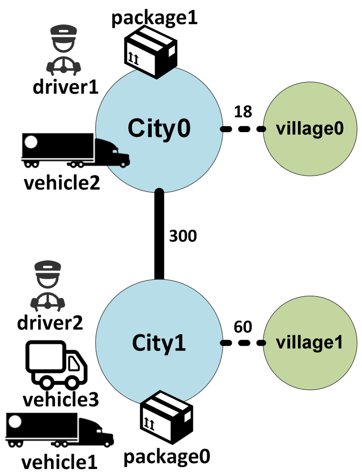
Figure 6.3: The candidate plans π'_a and π''_a.



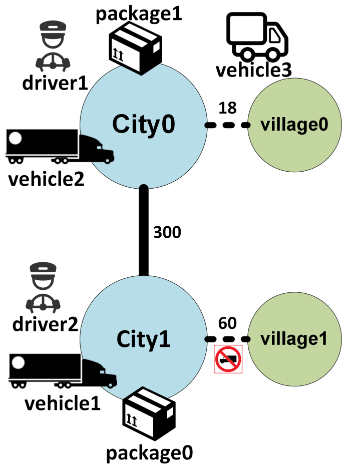
(a) `agentA` used π'_a of `scenario1`



(b) `agentA` used π''_a of `scenario1`



(c) `agentA` used π'_a of `scenario2`



(d) `agentA` used π''_a of `scenario2`

Figure 6.4: Snapshots of `agentB` state at time instance 500

as it can no longer execute the a_6 of π_b resulting in the situation shown in Fig. 6.4d. agentB will perform replanning or repairing, which will cause it to suffer a time-loss of 368 time units, in addition to a negligible processing time for invoking repairing or replanning. On the other hand, if agentA had chosen to repair its plan respecting plan commitment using π'_a (shown in Fig. 6.5), the resulting situation for agentB is demonstrated in Fig. 6.4c.

a'_1 : 0.002 : ($walk$ **driver3** $city0$ $village0$)[50.000]
a'_2 : 50.010 : ($board$ **driver3 vehicle3** $village0$)[10.000]
a'_3 : 60.020 : ($drive\text{-}van$ **driver3 vehicle3** $village0$ $city0$)[18.000]
a'_4 : 78.030 : ($disembark$ **driver3 vehicle3** $city0$)[10.000]
a'_5 : 88.040 : ($load\ package0$ **driver3 vehicle3** $city0$)[17.000]
a'_6 : 105.050 : ($board$ **driver3 vehicle3** $city0$)[10.000]
a'_7 : 115.060 : ($drive\text{-}van$ **driver3 vehicle3** $city0city1$)[300.000]
a'_8 : 415.070 : ($disembark$ **driver3 vehicle3** $city1$)[10.000]
a'_9 : 425.080 : ($unload\ package0$ **driver3 vehicle3** $city1$)[17.000]

Figure 6.5: The commitment repair plan π'_a.

In that case, agentB would suffer a failure; however, its time-loss would have been only the negligible processing time for invoking repairing or replanning, as it can utilise vehicle3 on the road to village1 and vehicle3 is already at city1 (if agentA has used π'_a for repairing its failure). The plan commitments $c(\pi'_a, \pi_a) = 0.55$, $c(\pi''_a, \pi_a) = 0.57$, thus π'_a is more committed.

6.4 Plan Repair through Plan Commitment

General temporal planners are called "general" because they are able to handle a wide range of planning problems that involve temporal constraints, rather than being specialised for a particular domain or application. General temporal planners aim to find good quality plans, where the quality is considered in most cases as the plan makespan, i.e., plan duration. However, it is possible to define a cost function, called *metric* function, to be optimised in many of them. Unfortunately, it is not possible to define a metric using information not defined in the PDDL input files (domain and problem). In our case, we want the cost function to be the plan commitment function, which depends on the objects of the actions of the original plan π. Therefore, we need to introduce certain modifications to an existing planner to use the plan commitment to measure plan quality. We have chosen TFLAP (Sapena, Marzal, et al. 2018), a temporal forward-chaining partial-order planner, for the following reasons:

- It has a good performance. It took third place in the temporal track of the last planning competition (IPC'2018[1]). Two planners, TemPorAl (Cenamor et al. 2018) and CP4TP (Furelos-Blanco et al. 2018), exhibited slightly better performance, but they followed a portfolio approach. This means that they integrate different temporary planners within, making it difficult to introduce modifications (each change would have to be implemented for all inner planners).
- The source code of TFLAP is publicly available [2].
- TFLAP implements a standard A* search, so it is only necessary to make changes in the calculation of the evaluation function, that is, the cost of the path and the heuristic function.
- TFLAP supports planning in temporal settings.

We will call C-TFLAP, the planner resulting after the changes made in TFLAP. In the following subsections, we will describe the working scheme of C-TFLAP and the new heuristic function.

[1] https://ipc2018.bitbucket.io
[2] https://bitbucket.org/osapena/tflap

6.4.1 Working Scheme of C-TFLAP

Fig. 6.6 shows C-TFLAP's working scheme. The main modules of TFLAP are kept practically intact:

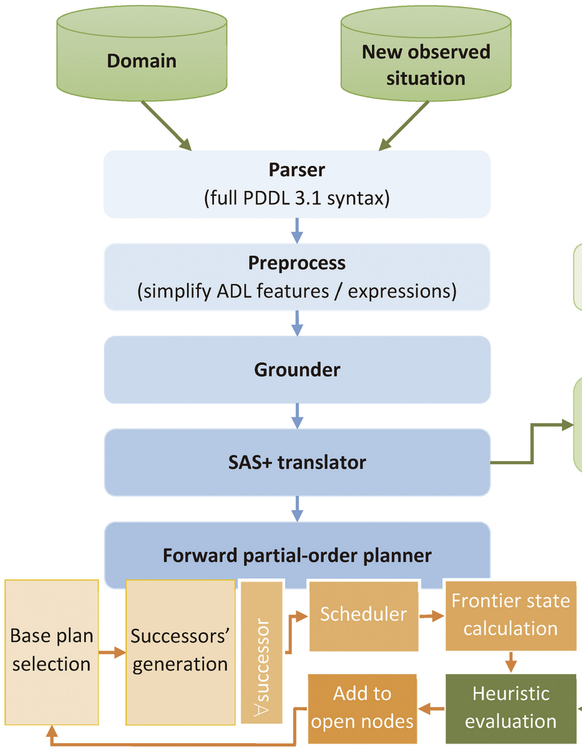


Figure 6.6: Working scheme of C-TFLAP.

- Initially, the domain and problem files of the planning task are sent to the parser module. Then, the information is preprocessed, grounded and translated to the SAS+ formalism (Bäckström et al. 1995), which is a formalism that is used in automated planning and scheduling systems to represent planning problems using multi-valued state variables instead of propositional facts.

- Then, TFLAP uses A* search guided by an evaluation function, with search nodes represented by partial-order plans. The search tree starts with a plan containing one fictitious dummy action representing the initial state. This fictitious action has no conditions and its effects correspond to the fluents of the initial state. The planner applies the following six steps at each iteration of the search process until a solution plan is found:

  1. The evaluation function selects the best node, called the base plan, from the set of open nodes. Open nodes refer to a set of nodes representing potential plans or actions that have not yet been fully evaluated or expanded.

  2. It generates all possible successors of the base plan. A successor plan adds a new action to the base plan by supporting its conditions and solving all threats. Threats arise if there is a resource conflict which occurs when two actions that can potentially be executed in parallel have inconsistent effects (an effect of one negates an effect of the other), interference (one deletes a precondition of the other), or competing needs (they have inconsistent preconditions).

  3. Actions in each successor node are scheduled, determining their start and end times.

  4. The node's schedule is used to compute the frontier state, i.e. the state resulting from executing the plan comprised in the node.

  5. Successor nodes are evaluated using state-based heuristics computed on their frontier states.

  6. Finally, the successor plans are added to the set of open nodes.

The first modification to make in TFLAP is to add the original plan π as a planner input. We have developed a parser for this task. Then we proceed to compute the commitment distance of all the possible actions in the planning task to the original plan. For the commitment distance of a single action a' to a plan π, $\delta(a', \pi)$ see Def. 6.3.3, Eq. 6.3.1. In the next section, we describe in detail how the commitment distance is used during the heuristic evaluation of the nodes to find good-quality repair plans.

6.4.2 Heuristic Evaluation Directed by the Commitment Distance

As we mentioned before, once we generate a new search node, the planner calculates its frontier state, i.e. the state resulting from executing the plan comprised in that node. Formally, we define the frontier state of a plan π', $S(\pi')$, as a set of fluents (variable, value) that hold after the execution of π'.

The heuristic evaluation of a plan π', $h_{cd}(\pi')$, is an estimate of the commitment distances of the actions required to reach the goal G from $S(\pi')$. For this, we follow two steps:

1. A relaxed planning graph (C-RPG) is calculated. This graph is based on a *Graphplan*-like expansion where delete effects are ignored.

2. A relaxed plan is drawn by traversing the C-RPG backwards, i.e. starting from the goals. We compute $h_{cd}(\pi')$ as the sum of the commitment distances of the relaxed plan actions.

This heuristic function is calculated similarly to that proposed in (Hoffmann 2003), but modified to get estimates based on commitment distances.

RPG based on Commitment Distances (C-RPG)

C-RPG is a graph where levels organise fluents and actions. The first level (level 0) is composed of the fluents held in the frontier state $S(\pi')$ of the evaluated node. The following level is action level 0, where the actions applicable in that state, i.e. whose preconditions are true in $S(\pi')$, will be located. An action will be placed at a certain level if the conditions required for it to happen are present at that level or any level before it. When an action is placed at a certain level, its outcome will be recorded at a level determined by its commitment distance from the current level.

The graph expansion continues until all the task goals are achieved. The algorithm can be observed in Algorithm 5.

Extraction of the Relaxed Plan

Once the C-RPG is built, a relaxed plan is computed by traversing the graph backwards. Algorithm 6 shows how the actions of the relaxed plan are chosen. The heuristic value of plan π', $h_{cd}(\pi')$, is then calculated as the sum of the commitment distances of these actions.

6.4.3 Two Queues based Search

The search in C-TFLAP uses two priority queues:

- `pqCD`: this queue sorts the search nodes according to the evaluation function $f_{cd}(\pi) = g_{cd}(\pi) + h_{cd}(\pi)$, where $g_{cd}(\pi)$ is the sum of the commitment distances of the actions in π, and $h_{cd}(\pi)$ is the heuristic value of π (described in Section 6.4.2). This function guides the search effectively towards solutions with small commitment distances. However, these solutions are sometimes too long because they include many redundant

Algorithm 5 *C-RPG* expansion algorithm for a plan π', being π the original plan to repair.

Input: the plan to evaluate π', and the dynamic planning problem $(G, \pi, O', I', TILs')$

Output: the fluent and action levels

```
1: procedure C-RPG
2:     ▷ Initialising first fluent level with the frontier state
3:     $fl \leftarrow S(\pi')$                                ▷ Fluents scheduled to be inserted
4:     $level(f) \leftarrow \begin{cases} 0 & \texttt{, if } f = (variable, value) \in S(\pi') \\ \infty & \texttt{, otherwise} \end{cases}$
5:     $level(a) \leftarrow \infty, \forall a \in O'$          ▷ Initially all actions are unreached
6:     ▷ New expansion stage while there are unachieved goals
7:     while $\exists g \in G / level(g) = \infty$ do
8:         if $fl = \emptyset$ then
9:             fail                                         ▷ Goals are not reachable
10:        ▷ Get the scheduled fluent with smallest level value
11:        $f \leftarrow argmin(level(f_i)), \forall f_i \in fl$
12:        $fl \leftarrow fl - \{f\}$                         ▷ Remove it not to be added again
13:        ▷ Unreached actions with $f$ as precondition
14:        for each $a \in O' / level(a) = \infty \wedge f \in prec(a)$ do
15:            ▷ Check if all action preconditions are met
16:            if $level(f_i) <= level(f), \forall f_i \in prec(a)$ then
17:                $level(a) \leftarrow level(f)$             ▷ Action level updated
18:                $e\_level \leftarrow level(a) + \delta(a, \pi)$    ▷ Level for the action effects
19:                for each $f_j \in eff(a) / level(f_j) > e\_level$ do
20:                    ▷ Schedule the action effects
21:                    $fl \leftarrow fl \cup \{f_j\}$
22:                    $level(f_j) \leftarrow e\_level$
```

Algorithm 6 Calculation of a relaxed plan to estimate the commitment distance to reach the goals from $S(\pi')$.

Input: the plan to evaluate, π', the dynamic planning problem $(G, \pi, O', I', TILs')$ and the fluent and action levels

Output: commitment distance estimate

```
 1: procedure $evaluate$
 2:     $h_{cd} \leftarrow 0$                                ▷ Heuristic value initialisation
 3:     ▷ Queue the problem goals that do not hold in $S(\pi')$
 4:     $cond \leftarrow \{g \in G$ / level(g) > 0}
 5:     while $cond \neq \emptyset$ do
 6:         ▷ Extract the fluent with the highest level value
 7:         $f \leftarrow argmax(level(f_i)), \forall f_i \in open\_cond$
 8:         $cond \leftarrow open\_cond - \{f\}$
 9:         if $level(f) = \infty$ then
10:             return $\infty$                              ▷ Unreachable goals
11:         ▷ Lower cost action that produces $f$
12:         $a \leftarrow argmin(level(a_i) + \delta(a_i, \pi))$,
                $\forall a_i \in O' / f \in eff(a_i)$
13:         ▷ Add the action commit. dist. to the heuristic value
14:         $h_{cd} \leftarrow h_{cd} + \delta(a, \pi)$
15:         ▷ Queue the action preconditions
16:         $cond \leftarrow cond \bigcup \{f_i \in prec(a) / level(f_i) > 0\}$
    return $h_{cd}$
```

actions that, due to their considerable similarity with the original plan's actions, help reduce the average commitment distance. For this reason, we use a second queue.

- `pqFF`: this queue sorts the nodes according to the evaluation function used in the FF planner (Hoffmann 2003): $f_{ff}(\pi) = g_{ff}(\pi) + h_{ff}(\pi)$, where $g_{ff}(\pi)$ is the number of actions in π and $h_{ff}(\pi)$ is the length of a relaxed plan extracted from a traditional relaxed planning graph. This function leads the planner to short solution plans.

C-TFLAP uses `pqCD` and `pqFF` alternatively. Initially, both queues contain only the empty initial plan, and the active queue is `pqCD`. Then, the planner repeats the following steps until a solution is found:

1. The plan π with the lower value according to its corresponding evaluation function is extracted from the active priority queue.

2. All successor plans of π are computed, evaluated and inserted in both queues.

3. The other queue now becomes the active priority queue.

Combining these queues provides a good compromise between the commitment distance and the plan length. We have chosen this method as it is a common approach in planning used in the state-of-the-art planner LAMA (Richter et al. 2010) and Fast Downward (Helmert 2006). Two heuristics are used with separate queues, thus exploiting the strengths of the utilised heuristics in an orthogonal way. We could try other methods to improve our planner's performance in future work.

6.5 Experimental Evaluation and Discussion

First, we describe the setup of the experiments, then provide a brief description of the test domains, and finally show the experiments, followed by a discussion of the results.

6.5.1 Setup of Experiments

To validate the proposed approach for commitment plan repair, we selected three application domains: a multi-agent logistics system, a multi-rover system and a multi-robot navigation system. The domains were modelled in the PDDL 2.2 and are tailored carefully to exhibit the interactions between agents and the possible perturbation one agent can cause to others. For each domain, 20 trials were generated randomly for two agents in a conservative shared setting. In these 60 trials, the initial locations, the goals, the planner's starting seeds (for LPG-TD and LPG-ADAPT), and the failures were generated randomly. The tests were based on a pre-calculated plan given by the planner OPTIC. Then replanning, plan-stability repair and commitment plan repair are done using LPG-TD, LPG-ADAPT and C-TFLAP, respectively. This ensures that we are not artificially enhancing the results by relying on how a planner explores its search space. Plan execution was performed using the simulation system introduced by (Babli, Ibáñez-Ruiz, et al. 2016). All the tests were run on an Intel(R) Core(TM) i7-6700HQ @2.60GHz CPU and 16.0 GB RAM.

6.5.2 Brief Description of the Test Domains

Logistics

A community of two agents for the logistics planning task was explained in Section 6.3.2, where each agent has a goal to deliver a package to a specified location. The logistics planning task has drivers, cities, villages, and vans that can traverse any road, and trucks only travel between cities.

A multi-rover system

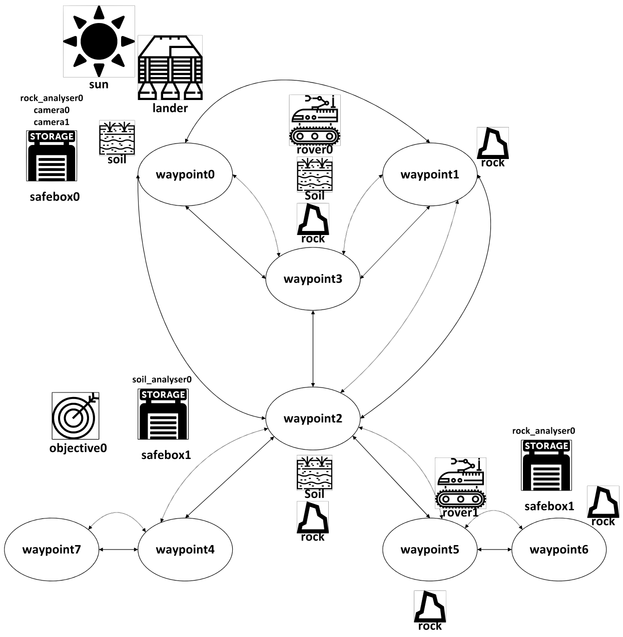
We designed a community of two agents (`agentA` and `agentB`) for the test problems from multi-rover systems in IPC. There are two rovers in the system. Each can independently traverse the surface of Mars from one waypoint to another visible waypoint, take pictures in different modes such as colour, high and low resolution, collect soil and rock samples and transmit all the exploration data back to the lander. The domain is tailored to exhibit interactions between rovers that may cause perturbation. More specifically, the cameras and the samplers are not built-in rovers but tools that can be equipped, used and stored inside safeboxes at waypoints. The negative interactions occur when the first agent, `agentA`, generates a plan and publishes the end location of the tools, such as the camera. The second agent, `agentB`, generates its plan depending on that information. During the execution, `agentA` suffers a failure and fixes its plan. Our interest is whether or not the failure of the first agent will also be propagated to `agentB`. The left side of Figure 6.7 shows an abstraction of a multi-robot rover system.

A multi-robot navigation system with locked doors

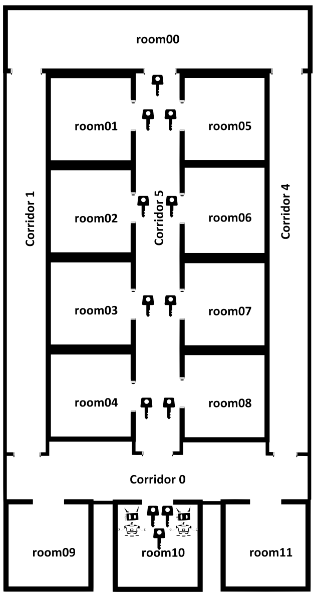
We designed a community of two agents (agentA and agentB) for the test problems from a multi-robot navigation system inspired by multi-robot planning with conflicts and synergies (Jiang et al. 2019). There are two robots in the system; each can independently move between locations (rooms and corridors) to navigate from their initial locations to their goal positions. Resource sharing collaborations among robots can be easily observed as the agents share the same environment and constrained resources, such as locked doors that need unlocking with the possibility of action synergy (e.g., multiple robots going through a door after a single expensive door-opening action). The domain is tailored to exhibit interactions between robots that may cause perturbation. More specifically, we added batteries and keys that the robots could equip to move and unlock doors, respectively. The negative interactions occur when the first agent, agentA, generates a plan and publishes the doors it will unlock, then other agents (in our case agentB) build their plans accordingly. During the execution, agentA suffers a failure and fixes its plan and our interest is whether or not agentB will suffer a failure. The right side of Figure 6.7 shows an abstraction of a multi-robot navigation system with locked doors.

6.5.3 The Experiments

Fig. 6.8 shows the average makespan of the new plans when the agents used replanning (LPG-TD), plan-stability repair (LPG-ADAPT), and commitment plan repair (C-TFLAP), for the selected test domains, compared to the original plans. In Fig. 6.8a, there are three clusters of columns. The first cluster of columns depicts the average makespan of plans for agentA in the logistics domain. In contrast, the second and the third clusters are for the rovers and the navigation domains, respectively. Within each cluster, there are four columns. The first



(a) Multi-robot rover system.

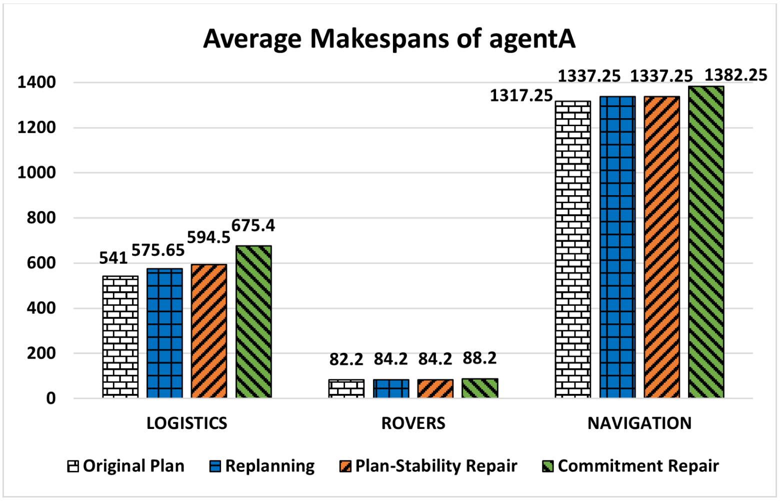


(b) Multi-robot navigation.

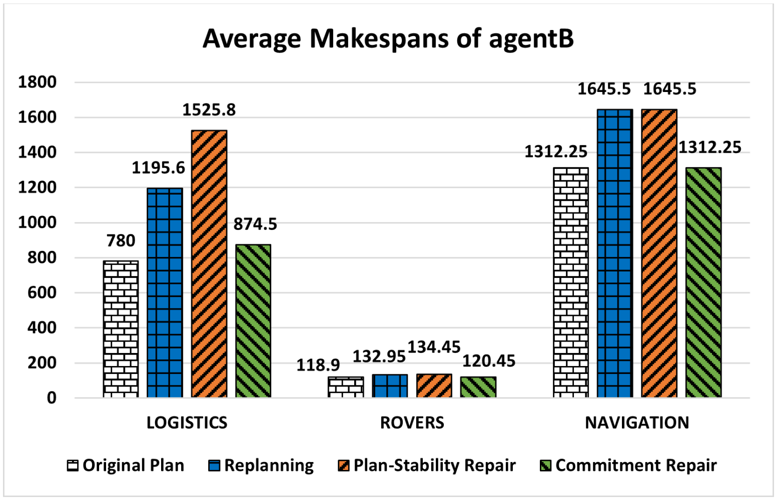
Figure 6.7: Abstraction of the domains environments.

bricks-style column represents `agentA` original plans' average makespan. The second, third and fourth columns represent the average makespan when `agentA` uses replanning, plan stability repair and commitment repair, respectively. On the other hand, Fig. 6.8b depicts the average makespan of `agentB`'s plans corresponding to how `agentA` fixed its plans in Fig. 6.8a. For example, the second column in Fig. 6.8b represents the average makespan of `agentB`'s plans in the logistics domain when it used replanning to fix its plan after `agentA` fixed its own plan using replanning.

Tables 6.2, 6.3 and 6.4 compare whether or not the new plans generated by `agentA` (when it performed replanning using LPG-TD, plan-stability repair using LPG-ADAPT and commitment repair using C-TFLAP) caused the second agent,



(a) `agentA`.



(b) `agentB`.

Figure 6.8: The test domains' average makespan of the three approaches.

`agentB`, to suffer failures. In addition to time lost by the agents due to failures and new plans for the logistics, rovers and navigation domains, respectively. We also compare the saved time when considering the plans of both agents.

The totals and the averages for the number of failures, the agents' time-loss and the overall saved time for the experiments are calculated and shown in the last two rows of each table.

For brevity, to express time-loss and saved time, we define the following notation:

- π_{ag} is the original plan of an agent ag.
- π_{ag}^{m} is the new plan of agent ag using method $m \in \{r, d, c\}$, where r stands for replanning, d for plan-stability repair, and c for commitment plan repair. The agents use the same method within an experiment, e.g., when using replanning, if we have two agents, then both agents will use replanning to fix failures.
- $F_{ag_j}|\pi_{ag_i}^{m}$ is the number of failures suffered by an agent ag_j due to $\pi_{ag_i}^{m}$.
- $LT_{ag}^{m} = makespan(\pi_{ag}^{m}) - makespan(\pi_{ag})$ is the time-loss of an agent ag when using a particular method m. A negative time-loss implies saved time.
- $ST_{m1}^{m2} = \sum_{ag} makespan(\pi_{ag}^{m2}) - \sum_{ag} makespan(\pi_{ag}^{m1})$ is the saved time for all agents, defined as the makespan difference in the new plans of the agents when using method $m2$ compared to method $m1$. A negative saved time implies a time-loss.

6.5.4 Discussion

First, it is worth mentioning that C-TFLAP average runtime for the experiments was 0.5 seconds. The running times are not compared since the three approaches can solve problems in a matter of seconds. The planning time is not a

LOGISTICS	**Replanning**			**Plan-Stability**				**Commitment**				
Nº	$F_b\|\pi_a^r$	LT_a^r	LT_b^r	$F_b\|\pi_a^d$	LT_a^d	LT_b^d	ST_d^r	$F_b\|\pi_a^c$	LT_a^c	LT_b^c	ST_c^r	ST_c^d
0	1	0.00	450.00	1	0	530	-80.00	0	170	0	280.00	360
1	1	70.00	780.00	1	70	1663	-883.00	0	170	0	680.00	1563
2	1	0.00	470.00	1	0	530	-60.00	1	0	460	10.00	70
3	1	-137.00	0.00	0	240	0	-377.00	0	240	0	-377.00	0
4	1	0.00	780.00	1	0	780	0.00	0	300	0	480.00	480
5	1	240.00	630.00	1	240	1513	-883.00	0	240	0	630.00	1513
6	0	0.00	0.00	0	0	0	0.00	0	0	0	0.00	0
7	1	240.00	780.00	1	240	1790	-1010.00	1	290	790	-60.00	950
8	1	0.00	780.00	1	0	780	0.00	0	300	0	480.00	480
9	0	0.00	0.00	0	0	0	0.00	0	0	0	0.00	0
10	0	0.00	0.00	0	0	0	0.00	0	0	0	0.00	0
11	1	70.00	780.00	1	70	1663	-883.00	0	170	0	680.00	1563
12	1	0.00	577.00	1	0	577	0.00	0	170	0	407.00	407
13	1	240.00	630.00	1	240	1513	-883.00	0	240	0	630.00	1513
14	1	0.00	0.00	1	0	0	0.00	1	0	0	0.00	0
15	1	120.00	630.00	1	120	1500	-870.00	1	290	640	-180.00	690
16	1	20.00	-7.00	1	20	80	-87.00	1	20	0	-7.00	80
17	1	0.00	455.00	1	0	1150	-695.00	1	88	0	367.00	1062
18	1	0.00	577.00	1	0	577	0.00	0	170	0	407.00	407
19	1	-170.00	0.00	1	-170	270	-270.00	1	-170	0	0.00	270
TOTAL:	17	693	8312	16	1070	14916	-6981.00	7	2688	1890	4427	11408
AVERAGE:	0.85	34.65	415.6	0.8	53.5	745.8	-349.05	0.35	134.4	94.5	221.35	570.4

Table 6.2: Comparison of the failures, time-loss and saved time in the logistics domain.

ROVERS	**Replanning**			**Plan-Stability**				**Commitment**				
Nº	$F_b\|\pi_a^r$	LT_a^r	LT_b^r	$F_b\|\pi_a^d$	LT_a^d	LT_b^d	ST_d^r	$F_b\|\pi_a^c$	LT_a^c	LT_b^c	ST_c^r	ST_c^d
0	1	0	0	1	0	0	0	1	0	0	0	0
1	1	10	0	1	10	0	0	1	10	0	0	0
2	1	0	50	1	0	50	0	1	15	0	35	35
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	20	30	0	20	30	0	0	20	0	30	30
5	0	0	30	0	0	30	0	0	0	30	0	0
6	1	0	50	1	0	50	0	1	0	0	50	50
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	20	0	-20	-20
9	0	0	0	0	0	0	0	0	0	0	0	0
10	1	0	30	1	0	30	0	1	0	0	30	30
11	1	0	30	1	0	30	0	1	0	0	30	30
12	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	30	1	0	30	0	1	0	0	30	30
14	1	0	0	1	0	0	0	1	15	0	-15	-15
15	1	10	1	1	10	1	0	1	10	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0
17	1	0	0	1	0	30	-30	1	0	0	0	30
18	1	0	0	1	0	0	0	1	0	0	0	0
19	1	0	30	1	0	30	0	1	30	0	0	0
TOTAL:	12	40	281	12	40	311	-30	12	120	31	170	200
AVERAGE:	0.6	2	14.05	0.6	2	15.55	-1.5	0.6	6	1.55	8.5	10

Table 6.3: Comparison of the failures, time-loss and saved time in the rovers domain.

NAVIGATION	**Replanning**			**Plan-Stability**				**Commitment**				
Nº	$F_b\|\pi_a^r$	LT_a^r	LT_b^r	$F_b\|\pi_a^d$	LT_a^d	LT_b^d	ST_d^r	$F_b\|\pi_a^c$	LT_a^c	LT_b^c	ST_c^r	ST_c^d
0	1	-100	615	1	-100	615	0	0	0	0	515	515
1	1	100	615	1	100	615	0	0	100	0	615	615
2	2	0	-100	2	0	-100	0	0	100	0	-200	-200
3	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	-100	2	0	-100	0	0	100	0	-200	-200
5	1	0	615	1	0	615	0	0	100	0	515	515
6	0	0	0	0	0	0	0	0	0	0	0	0
7	1	100	615	1	100	615	0	0	100	0	615	615
8	0	-100	0	0	-100	0	0	0	-100	0	0	0
9	1	0	615	1	0	615	0	0	100	0	515	515
10	0	100	0	0	100	0	0	0	100	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0
12	1	0	615	1	0	615	0	0	100	0	515	515
13	1	0	615	1	0	615	0	0	100	0	515	515
14	1	100	615	1	100	615	0	0	100	0	615	615
15	1	100	615	1	100	615	0	0	100	0	615	615
16	1	100	615	1	100	615	0	0	100	0	615	615
17	0	0	0	0	0	0	0	0	100	0	-100	-100
18	0	0	0	0	0	0	0	0	0	0	0	0
19	1	0	715	1	0	715	0	0	100	0	615	615
TOTAL:	15	400	6665	15	400	6665	0	0	1300	0	5765	5765
AVERAGE:	0.75	20	333.25	0.75	20	333.25	0	0	65	0	288.25	288.25

Table 6.4: Comparison of the failures, time-loss and saved time in the navigation domain.

determining factor in choosing a concrete approach, and for our purposes, we will focus on the disruptions among agents.

We compared the stability and commitment of plans generated by the first agent, agentA, when using LPG-ADAPT for plan-stability repair and C-TFLAP for commitment repair, across different test domains in our experiments. As expected, mostly the use of one metric leads to plans that are better according to such metric. The commitment repair has achieved more committed plans than plan-stability repair, as the average commitment distances for commitment repair were 0.362, 0.108 and 0.275 compared to 0.450, 0.132 and 0.405, in the logistics, rovers, and navigation domains, respectively.

For the logistics test domain, commitment repair has avoided causing a failure to agentB in 65% of the experiments, compared to 15% and 20% in replanning and plan-stability repair, respectively. agentB average time-loss when agentA used commitment repair equals 23% of the average time-loss when it used replanning and 13% of the average time-loss when it used plan-stability repair.

For the rovers test domain, even though each of the three approaches has avoided causing failures in agentB in 40% of the experiments, this does not mean that they are equally good. Since agentB average time-loss when agentA used commitment repair equals 11% of the average time-loss when it used replanning and 10% of the average time-loss when it used plan-stability repair. In addition, using commitment repair did not result in time-loss for agentB in 11 out of 12 times when agentB failures were inevitable, in comparison with 5 out of 12 when it used replanning, and 4 out of 12 when it used plan-stability repair.

For the navigation test domain, commitment repair has avoided causing a failure to agentB in 100% (all) of the experiments compared to 35% in replanning and plan-stability repair. agentB average time-loss when agentA used commitment repair equals 0% of the average time-loss when it used replanning or plan-stability, as the average time-loss when `agentA` used commitment repair equals 0.00, compared to 333.25 when `agentA` used either replanning or plan-stability repair.

We also find it interesting to compare the saved time when considering implications for all agents. When we compared the saved time when the agents used commitment repair against the results of when they used plan-stability repair, we found that:

- For the logistics domain, commitment repair has saved time in 15 experiments and neither saved nor wasted time in 5 experiments, with 570.4 average saved time.
- For the rovers domain, commitment repair has saved time in 7 experiments, wasted time in 2 experiments and neither saved nor wasted in 11 experiments, with 10 average saved time.
- For the navigation domain, commitment repair has saved time in 11 experiments, wasted time in 3 experiments and neither saved nor wasted in 6 experiments, with 288.25 average saved time.

When we compared the saved time when the agents used plan-stability against the results of when they used replanning, we found that:

- For the logistics domain, plan-stability repair has wasted time in 12 experiments and neither saved nor wasted time in 8 experiments, with 349 average wasted time.
- For the rovers domain, plan-stability repair has wasted time in 1 experiment and neither saved nor wasted time in 19 experiments, with 1.5 average wasted time.

- For the navigation domain, plan-stability repair has neither saved nor wasted time in 20 experiments.

When we compared the saved time when the agents used commitment repair against the results of when they used replanning, we found that:

- For the logistics domain, commitment repair has saved time in 11 experiments, wasted time in 4 experiments and neither saved nor wasted time in 5 experiments, with 221 average saved time.
- For the rovers domain, commitment repair has saved time in 6 experiments, wasted time in 2 experiments and neither saved nor wasted time in 12 experiments, with 8.5 average saved time.
- For the navigation domain, commitment repair has saved time in 11 experiments, wasted time in 3 experiments and neither saved nor wasted time in 6 experiments, with 288 average saved time.

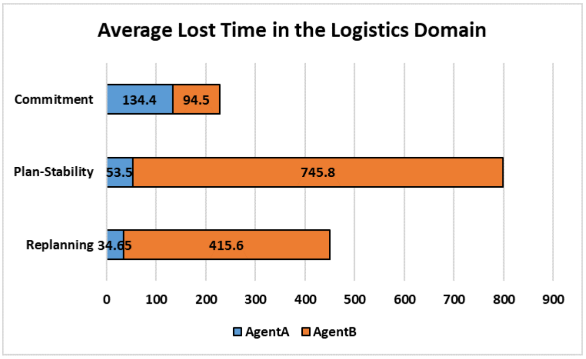
The results described above support our claim of the significance of plan commitment for other agents. It reduced the number of failures and the time-loss when failures were avoidable and decreased the time-loss when failures were inevitable. In addition, we found that commitment repair has achieved better saved time than plan-stability repair when considering implications for all agents and when contrasting each approach against replanning.

It is worth noting that there are situations when both commitment repair and plan-stability repair have wasted time compared to replanning, such as experiments 3 and 15 in the logistics domain. This is natural and was expected as the time waste occurs when there is no dependency between the agents; as the first agent suffers a failure, the second agent's plan is unaffected. When the first agent uses replanning to fix its failure, it generates the shortest makespan plan. In contrast, it generates longer plans when it uses commitment repair and plan-stability repair, leading to a loss of time.

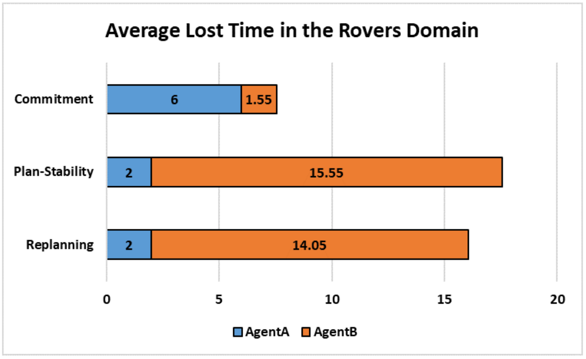
According to Fig. 6.8, when comparing the average makespan of the agents' new plans when they used replanning, plan-stability and commitment repair in the three test domains, we found that plan commitment generated longer average makespan plans for `agentA` and that it generated shorter average makespan plans for `agentB`. In addition, based on our experiments, it may be observed that: 1) in a few cases when LPG-ADAPT seeks to generate a more stable plan, it adds many instances of the original plan's actions, which increases the stability but results in longer makespans and causes a time-loss for the agent compared to replanning which only seeks the shortest makespan plan. 2) Plan stability focuses on generating a higher stability plan (within the agent itself of its new plan compared to its original plan). However, plan stability does not correlate with makespan, i.e., higher stability plans may have longer or shorter makespans than lower stability plans. More importantly, a higher stability plan for an agent does not necessarily mean it will cause less disturbance to other agents.

Fig. 6.9 compares the agents' average time-loss when using replanning, plan stability, and commitment repair in the three test domains to highlight further the additional costs (increased time-loss) incurred by the agent that suffered the failure (`agentA`) and the resulting decreased time-loss experienced by other agents (`agentB`). For brevity, there follows a detailed comparison of the values for the Logistics domain (Fig. 6.9a), whereas the values' comparison for the other domains in Fig. 6.9b and Fig. 6.9c are shown in Table 6.5.

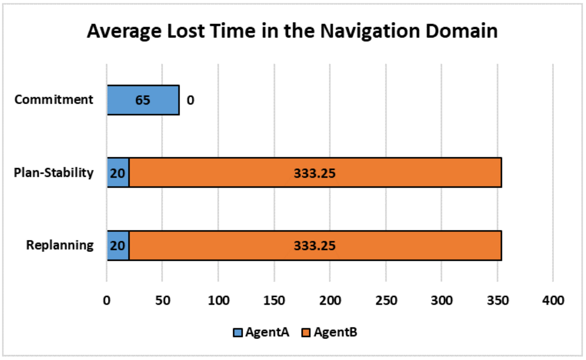
In Fig. 6.9a, the comparison between the commitment column (the top column) and the plan-stability column (the middle column) demonstrates that for the Logistics domain, `agentA`'s average time-loss using commitment plan repair



(a) Logistics.



(b) Rovers.



(c) Navigation.

Figure 6.9: Agents' average time-loss using the three approaches in the test domains.

(134.4) exceeds its value when it used plan stability (53.5) by approximately 1.512 times. Correspondingly, the resulting `agentB`'s average time-loss using commitment plan repair (94.5) is approximately 6.89 times less than its value when it used plan stability (745.8). Similarly, the comparison between the commitment column (the top column) and the replanning column (the bottom column) demonstrates that for the Logistics domain, `agentA`'s average time-loss using commitment plan repair (134.4) exceeds its value when it used replanning (34.65) by approximately 2.875 times. Correspondingly, the resulting `agentB`'s average time-loss using commitment plan repair (94.5) is approximately 3.397 times less than its value when it used replanning (415.6).

Change in Average Time-Loss						
	Logistics		**ROVERS**		**NAVIGATION**	
	$AVG.\ LT_c^r$	$AVG.\ LT_c^d$	$AVG.\ LT_c^r$	$AVG.\ LT_c^d$	$AVG.\ LT_c^r$	$AVG.\ LT_c^d$
agentA	2.88 times more	1.51 times more	2.00 times more	22.00 times more	2.25 times more	2.25 times more
agentB	3.4 times less	6.89 times less	8.06 times less	9.03 times less	-333.25	-333.25

Table 6.5: The change in each agent's average time-loss in the three application domains. Where $AVG.\ LT_c^r$ is the average time-loss when an agent used commitment repair compared to replanning, and $AVG.\ LT_c^d$ is the average time-loss when an agent used commitment repair compared to plan stability.

As can be observed in Table 6.5, the plan commitment metric has generated longer plans for the agent that suffered the failure (increased time-loss) and that it generated shorter plans for other agents (decreased time-loss) compared to replanning and plan stability repair. In addition, plan stability and replanning tend to optimise the agent's plan without considering the negative effects they will cause on other agents. Commitment repair, on the contrary, generates longer plans but significantly minimises the damage it causes to other agents.

In light of this finding and the positive results obtained when comparing the overall saved time for all agents, the results suggest that commitment repair by an agent can have a significant positive impact on other agents (indicated by the reduced time-loss of the second agent, the reduced number of propagated failures and the total saved time) if that agent is willing to endure some loss of utility (indicated by the increased makespan for the agent that originally suffered a failure).

This study acknowledges that the plan commitment metric is a more sensitive metric than the plan stability metric for plan repairing among multiple agents sharing the same execution environment in a conservative shared setting. Once one object changes inside an action, the plan stability metric (or other action-based metrics) cannot further distinguish the quality of the plans, no matter whether the change is small or large. Therefore, they cannot distinguish plans that disrupt other agents from those that do not, unlike the plan commitment metric, which accounts for and is sensitive to every change.

On the one hand, the plan commitment metric produces better results than other metrics whenever the agents use shared resources for generating their plans, one agent suffers a failure, multiple possible plans exist to fix the failure and reach the agent's goals, and some of these plans utilise the shared resources in a way that do not disrupt other agents while other plans do not. In agents that use the plan commitment metric, plans that utilise the same shared resources of the original plan are prioritised over plans that utilise alternative equivalents or resources of different types.

- Plans that utilise the same shared resources that were used in the original plan lead to meeting other agents' expectations, avoiding causing them failures and preventing time-loss.

- Plans that utilise alternative equivalent resources also meet other agents' expectations, despite causing them failures but preventing time-loss since the resource is equivalent and can be used as an alternative.
- Finally, plans that utilise resources of different types discard other agents' expectations; thus, they cause failures and time-loss to other agents.

Action-based metric distances do not distinguish the previously mentioned types of plans; thus, they are indifferent about selecting plans that disturb other agents.

On the other hand, if there is only one way for the agent to fix the failure and achieve its original goals, then an agent that uses plan commitment would have an equal performance as an agent that uses any other metric. Furthermore, in situations where there is no agent dependency (no shared resources) and multiple ways to fix a failure, the additional plan length resulting from generating a more stable or committed plan by the agent that suffered the failure does not lead to saving the other agents' time. In contrast, optimising other metrics of value to the agent, such as makespan, is more beneficial for the community of independent agents.

In the experiments, some failures were inevitable in both the logistics and rovers domains. In contrast, all the failures were avoidable in the navigation domain. The different nature of the test domains and the variety of the experiments made it interesting to examine the agents' behaviour when using the three approaches (replanning, plan-stability repair or commitment plan repair).

6.6 Extensibility and Future Work

Extensibility

C-TFLAP can be modified by manipulating the heuristic function; therefore, it can be easily extended to include new functionalities. There follows a brief description of how we extend its usage with a cognitive semantic layer to increase the agents' chances of a successful repair.

Automated planning knowledge extension is a cognitive semantic approach that involves adding new object types to the predefined list of object types for a planning task in a dynamic way (Babli and Onaindia 2019). This approach is used to make the system more context-aware and better able to reason with incomplete knowledge, which can improve its autonomy and chances of successful repairing.

Using the approach presented in (Babli and Onaindia 2019), we utilised the context-aware knowledge acquisition that was designed initially to formulate new goals. The ontological details are abstracted out in this study, and a detailed description can be found in (Babli and Onaindia 2019). For illustration, the output for the logistics application domain presented in this study is shown in Fig. 6.10.

When using automated planning knowledge extension, the domain is augmented with new types, and therefore it must be passed to C-TFLAP. In addition, an important requirement for this extension to work is to have the domain types modelled in a reasonable hierarchy in which the types that must be used differently (such as `van` and `truck` in the logistics domain explained in Section 6.3.2) must not be modelled as siblings under the same parent. It is worth mentioning that typing in PDDL is domain-independent as PDDL offers the

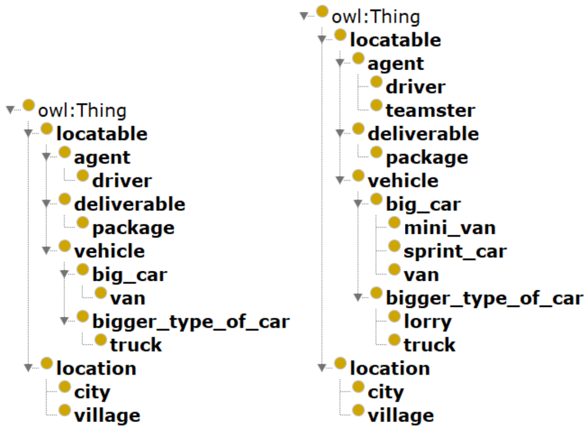


Figure 6.10: OWL Representation of the hierarchy of the predefined set of object types and the augmented hierarchy of types.

ability to express a type structure for the objects in a domain which allows typing of the parameters that appear in variables and operators. Furthermore, PDDL allows the types to be expressed as forming a particular hierarchy.

Fig. 6.10 demonstrates how `van` and `truck` are not sibling types, and they are modelled under different parent types, `big_car` and `bigger_type_of_car`, respectively. When this module is used, The condition of Line 8 in Algorithm 3 becomes:

```
else if $type(o) = type(o')$ or $parentType(o) = parentType(o')$ then $counter \leftarrow \max(counter, 0.5)$
```

Therefore, for the logistics problem shown in Section 6.3.2, the algorithm would favour plans that use the same object used in the original plan (the same van, `vehicle0` as in Scenario 1), otherwise a different object from the same type (a different van, `vehicle3` as in Scenario 2) or a different object from a different type that is compatible with the original object type (`vehicle4` of a new type sprinter that is compatible with the type van as the extended scenario shown below), otherwise, a different object from a non-compatible type (vehicle1 of a type truck) just to satisfy the goal.

Scenario (extended): agentA that was shown in Fig. 6.2a, suffers a failure due to the discrepancies driver0 and vehicle0 are no longer available, a new driver driver3 is available at city0 and vehicle4 is at a close location village0, which is a new object of a new type sprinter unknown to the agent. Whether agentA performs replanning or plan-stability repair, it will discard vehicle4. Similarly to scenarios 1 and 2, it will generate the plan π''_a which uses vehicle1 of type truck (shown in Fig. 6.3b) and thus, agentB will suffer a failure. On the other hand, if agentA and agentB utilise automated planning knowledge extension agentA would have realised that the type sprinter is equivalent to van. Then, agentA would have performed commitment plan repair which considers objects types, generating the plan π'_a shown in Fig. 6.11, which utilises vehicle4.

a'_1 : 0.002 : (*walk* **driver3** *city*0 *village*0)[50.000]
a'_2 : 50.010 : (*board* **driver3** **vehicle4** *village*0)[10.000]
a'_3 : 60.020 : (*drive-van* **driver3** **vehicle4** *village*0 *city*0)[18.000]
a'_4 : 78.030 : (*disembark* **driver3** **vehicle4** *city*0)[10.000]
a'_5 : 88.040 : (*load package*0 **driver3** **vehicle4** *city*0)[17.000]
a'_6 : 105.050 : (*board* **driver3** **vehicle4** *city*0)[10.000]
a'_7 : 115.060 : (*drive-van* **driver3** **vehicle4** *city*0*city*1)[300.000]
a'_8 : 415.070 : (*disembark* **driver3** **vehicle4** *city*1)[10.000]
a'_9 : 425.080 : (*unload package*0 **driver3** **vehicle4** *city*1)[17.000]

Figure 6.11: The commitment repair plan π'_a.

In that case, agentB would have suffered a failure and fixed it (with negligible processing time for invoking repairing or replanning) using vehicle4, thus avoiding time-loss of 368 time units. The plan commitments $c(\pi'_a, \pi_a) = 0.55$,

$c(\pi''_a, \pi_a) = 0.57$, thus π'_a is more committed. The authors did not use this extension in the evaluation against replanning and stability repair to avoid bias, as those methods cannot handle new objects of different but equivalent types.

Future Work

Multiple sound mathematical formulations may exist to define a plan metric distance; while all are mathematically correct, their applicability may vary depending on the specific context of the problem at hand. The plan commitment metric is designed to focus on the resources used within an action. In the vast majority of cases, two actions of the same operator that use the same resources are opposite actions. For example, drive from city A to city B and drive from city B to city A, or stack block A on block B and stack block B on block A. Although the plan commitment metric values both actions in the same way, the planner will select only those that lead to the objective. Additionally, in the future, if we want to compare plans of different agents, who probably use different models, our approximation will be more general and robust than an approximation that focuses on the ordering of objects within actions. This work outcome leaves many open lines of research and future development. There follow some of the implications and suggested future work.

It would be interesting to test with different types of agents (self-interested and collaborative) in the same community for future work. Currently, there is no limit on time/cost for the newly generated plan, and it may be interesting to specify or discover a threshold according to a library of plans. The current implementation is domain-independent, which may be considered an advantage. All objects are treated as equally important; however, for enhanced performance in specific applications, the plan commitment equation (shown in Section 6.3) can be easily modified to give some objects more weight. E.g., the objects that matter to other agents, instead of trying to commit to all the objects used in the original plan, possibly resulting in longer plans that discourage self-interested agents and may be considered a limitation in some cases. On the other hand, autonomously identifying the objects that matter more for each community of agents would require further investigation and research. C-TFLAP can continue to search for solutions with better commitment value after finding the first solution (as with LPG-ADAPT). Studying the utility of stability and commitment for an agent to autonomously decide when better stability or better commitment plan is worth the computation and the additional plan length after finding the first solution can be interesting. Furthermore, it may also be interesting to use the plan commitment in single-agent settings inversely, use the least committed plans for plan diversity purposes, and explore different solutions with states not explored in the original plan.

The notion of “responsibly repairing” a plan refers to the process of making changes or adjustments to a plan in a way that minimises the negative impact on other agents or systems. In other words, an agent that repairs its plan responsibly is one that takes into consideration the potential consequences of its actions on other agents and aims to minimise any disruption or adverse effects. This could involve, for example, consulting with other agents or systems to openly book commitments in advance such as intra-agent repair approach (Talamadupula, Smith, et al. 2013), or using a specific metric or algorithm to determine the repair to make as a single agent (no booking or coordination needed) such as the metric presented in our work. Coining the notion of “responsibly repairing“ as a formal definition would require further investigation and would be an important area for future research. Additionally, a formal definition would benefit from establishing clear criteria of what is considered responsible in different scenarios and settings.

6.7 Conclusion

Plan repair is crucial for any intelligent agent that operates in a dynamic environment. In action-sensitive distances such as the plan stability and the action distances, when one object changes in an action, the whole action is considered different. Consequently, once one object changes, action-sensitive distances cannot further distinguish the quality of the plans, no matter whether the change is small or large. This study presents a property called plan commitment to ensure a responsible repairing policy among agents that aims to minimise the negative impact on others and presents the arguments to support the claim that plan commitment is a valuable property when an agent may have made bookings to others. An implementation is presented based on TFLAP resulting in C-TFLAP planner. The empirical evaluation results support our claim as commitment repair can reduce the number of failures and the time-loss among agents. The results demonstrate that the suggested approach outperforms typical replanning and plan-stability repair for the mentioned purposes. The approach is domain-independent and agile and can be easily modified and extended.

Determining an appropriate threshold for the delay caused by a repair method in a particular agent, beyond which it would negatively impact the overall system performance (other agents), is a complex task that requires further research and investigation. In traditional multi-agent systems where agents coordinate and communicate their plans and goals openly, the threshold is determined by simple trade-offs such as the extra time spent by an agent versus the resulting time loss for other agents or the overall system's performance (total saved time in the community of agents). However, determining the threshold is more challenging in a community of agents with reduced communication and privacy. As we mentioned previously, the advantages of our method are that no coordination is required among agents, no communication is required after an agent generates its plan, and goals and plans are private and not shared among agents. In future work, we will continue our investigation for single-agent planning and repairing in a community of agents with reduced communication and privacy settings by estimating the threshold through historical data (case-based) or simulating the existence of another agent in the execution environment. By doing so, we can determine the expected impact of a delay on the overall system's performance and estimate the threshold of a disruption occurring in the worst case or on average.

Part III

Conclusions and Future Work

CHAPTER 7

Conclusions and Future Work

THIS chapter presents the general conclusions of this PhD thesis. The content of this chapter is organised as follows. The first section describes the characteristics and the necessity of autonomy techniques to deal with the dynamic environment change efficiently. The second section briefly describes the proposed approach and lists the most relevant contributions and results. The third and last section provides several lines of future work derived directly from this thesis work.

7.1 Introduction

Autonomy is more of a necessity than convenience in AI, robotics and multi-agent systems and is indeed a matter of power (Castelfranchi et al. 2003). This power allows the agent to utilise its available resources, knowledge and skills to maintain an adaptive behaviour while pursuing its goals through an architecture that makes it able. Lack of power produces dependence; therefore, autonomy and lack of autonomy are due to the agent’s architecture since it defines what resources an agent can use and provides its various capabilities. The problem is how to engineer and design a good, trustworthy architecture that strives to boost agents’ autonomy and allow a responsible interaction between agents. Planning, acting, perceiving, monitoring, learning, reasoning with dynamic environments, and reasoning with time and human interaction are the deliberative functions which constitute the driving forces of autonomous behaviour and are the pillars that one aims for in an autonomous agent’s architecture. In this PhD thesis, we target execution monitoring, context-aware knowledge acquisition and plan repairing.

Execution monitoring is crucial for the success of an autonomous agent that is executing a plan in a dynamic environment as it influences its ability to react to changes quickly. Due attention must be paid to changes; otherwise, the agent might suffer a plan execution failure or lose opportunities. Changes result from noisy observation, partial observability, non-determinism, dynamic environment or other agents operating in the same environment.

Partial or incomplete knowledge leads to unanticipated events that may bring about an opportunity for the task executed by the agent. Accounting for all possibilities in the agent’s model results in a lack of autonomy and is sometimes unfeasible. An agent that relies on the prior knowledge of its designer rather than on its own percepts is an agent that lacks autonomy. Instead, being rational requires the agent to be context-aware, gather information and extend what is already known from what is perceived to compensate for partial or incorrect prior knowledge and achieve the best possible outcome in various novel situations.

While executing its plan in a dynamic environment where multiple agents are operating, an autonomous agent may suffer a failure due to discrepancies between the expected and actual context and thus must replace its obsolete plan. In its endeavour to fix the failure and reach its original goals, the agent may unknowingly disrupt other agents executing their plans in the same environment.

7.2 Conclusions

Human beings are endowed with the ability to adapt and cope with a plethora of challenges encountered in daily life. Even when unexpected events and change occurs, we can handle them intuitively and solve problems effectively. When faced with a challenge, we look around, think outside the box, and utilise all available means to find solutions. We strive to increase our skills and capabilities to capitalise on opportunities when they arise and achieve more. Moreover, we are civilised; we treat each other with sympathy and respect as we embark on life's journey towards reaching our goals.

Similarly, the work carried out in this PhD thesis was initiated to endow the autonomous agents executing a plan in a dynamic environment with the ability to adapt to unexpected events and unfamiliar circumstances and utilise their own perception of context, enabling them to provide context-aware deliberative responses for seizing an opportunity or repairing a failure without disrupting other agents that are working in the same execution environment. This work focused on developing a domain-independent architecture capable of handling the requirements of such autonomous behaviour, namely, execution in a dynamic environment, context-aware knowledge acquisition and responsible plan repairing.

7.2.1 Brief Summary of the Proposed Approach

The proposed solution starts from an architecture (detailed in Chapter 3) based on integrating three fundamental modules. The first module is the intelligent system for execution simulation in a dynamic environment (Chapter 4). The second module is the context-aware knowledge acquisition for planning applications module (Chapter 5). The third module is the plan commitment repair module (Chapter 6). In this architecture, an agent can simulate the execution of its plan in a dynamic environment, integrate new objects, whether of existing or brand new types not previously considered in the agent module and use them when an execution failure occurs or to seize an opportunity. The novel plan repair strategy that uses plan commitment allows the agent to repair its plan without adverse effects on other agents. The architecture modules were investigated and evaluated in several applications in the appendices of this PhD thesis, a tourist assistant (Appendix A), a repairing agency application (Appendix B) and an assisted living home (Appendix C).

The three main contributions of this work are related to the developed architecture modules from which the following results have been drawn:

1. The agent uses the intelligent simulation system, which reads a PDDL domain and problem files as input. It calls a planner to generate a plan and simulates the execution in real-time. The simulation is done by transforming the plan into a timeline. The simulator periodically updates its internal state with real-world information, receives sensible environmental changes through live events and creates their corresponding timed events in the timeline. It monitors the execution of the plan. Events are processed in the context of the plan; if a failure occurs due to live events, the simulator reformulates the planning problem. This involves creating the new initial state and updating the time of timed events and the goals.

The simulator re-invokes a planner to generate a new plan and resumes the simulation. The simulator is a console application and has a GUI designed specifically for the context of smart tourism.

2. The agent uses a domain-independent approach that may be used as a part of a context-aware model in a deliberative context-aware ambient intelligence system. It bolsters an agent with the capability of autonomously extending its planning task to accommodate on the fly, only relevant (according to the agent's objects' types) and manageable new information (according to the agent's actions' schemas) that may trigger the formulation of new goal opportunities or may be used when a failure occurs.

3. The agent utilises a property called plan commitment that is very valuable when it may have made bookings to others as it ensures a responsible repairing policy among agents. An implementation is presented based on TFLAP resulting in C-TFLAP planner. The empirical evaluation results support our claim as commitment repair can reduce the number of failures and the lost time among agents. The results demonstrate that the suggested approach outperforms typical replanning and plan-stability repair for the mentioned purposes. The approach is domain-independent and agile and can be easily modified and extended.

7.2.2 Brief Summary of the Thesis Contributions

In this PhD thesis, we joined two fields in the realm of AI, the field of AP and the field of KR. The aims of this thesis have been directed towards allowing autonomous agents to deal with the dynamic environment change, not depending solely on the prior knowledge of designers but instead relying on their own percepts to provide context-aware deliberative responses for seizing an opportunity or repairing a failure that occurs during the execution of the plan.

In our opinion, the aims of this thesis and the five objectives to achieve the aims of the thesis (discussed in Section 1.2.2 of Chapter 1) have been favourably fulfilled. Follows a brief summary of the most general contributions of the thesis:

- Design and development of a domain-independent intelligent system for execution monitoring simulation in a dynamic environment. The system is responsible for the following deliberative functions; perception, acting and monitoring and triggering goal generation, replanning or plan repairing.

  – Perception: by sensing the IoT objects' values characterising the environment's actual state. If an opportunity is spotted or a failure occurs, perception provides reactivity. During the plan's execution, the agent may encounter new objects of existing types, integrates them into the planning problem, and uses them to repair failures or seize an opportunity.

  – Acting and Monitoring: by executing the plan, monitoring actions, and detecting failures that may occur

- Design and development of a domain-independent context-aware knowledge acquisition method. The agent creates an ontological representation of the planning domain and dynamically augments the predefined list of object types of the planning task with relevant new object types. This allows the agent to be context-aware of the environment and the task being performed and reason with incomplete knowledge. Consequently, boosting the system's autonomy and context awareness as the agent will be attentive (during execution simulation) to seize opportunities (for goal generation) or fix failures (replanning using an external planner or plan repairing).

- Design and development of the novel plan commitment repair strategy among multiple agents sharing the same execution environment for repairing an agent's plan when a failure is detected concerning the new observed state and the original plan of the agent. The agent utilises a new metric, plan commitment, as a heuristic to guide the search for the most committed repair plan to the original plan from the perspective of commitments made to other agents (the resources used in the original plan) whilst achieving the original goals. Consequently, the community of agents will suffer fewer failures due to the sudden changes (reduced revisions) or will have less lost time if the failure is inevitable.

7.3 Future Lines of Research

This section introduces some extensions that could be made based on this PhD thesis work. Despite the bolstered autonomy offered by the contributions that have been developed, they are but humble steps on the long road ahead to autonomy. Numerous extensions can significantly improve the autonomy provided by the system's architecture and the features it supports.

Concerning the intelligent system for execution simulation in a dynamic environment contribution, our simulation system can receive live information from the dynamic execution environment, monitor the execution of the plan, detect discrepancies and reformulate the planning problem upon failures to invoke replanning or plan repairing. A possible extension that could produce very important improvement in the autonomous behaviour of the agent is to investigate and derive techniques that decide what information is relevant to the agent and how relevance is decided to allow filtering information received from open-data platforms and other agents.

Currently, the dynamic change in the simulation system is received through live events during the execution; another possible extension is to allow simulating with uncertainty in action effects during execution.

The simulation system currently accepts a general PDDL domain+problem description as an input description of the planning task; a third possible extension is the HTN description to be compatible with HTN planning. Especially since HTN planning allows informed compound high-level expectations (Dannenhauer 2016), which goes very well with the use of ontologies, further research is required for autonomously devising the rules of high-level expectations for domain-independent problems instead of hard coding them according to experts in each domain.

On the other hand, concerning the context-aware knowledge acquisition for planning applications contribution, one key point of using knowledge sources (such as ConceptNet Open Mind Common Sense, OpenCyc, Verbosity players, LinkedDBTour and Open Multilingual WordNet) is the quality of the content within these knowledge sources. When human-out-of-the-loop autonomy performance is what we seek, autonomous systems can propagate biases learned from human data inside knowledge sources and reinforce misinformation, inaccurate classifications or any systematic discrimination found in society. Therefore besides filtering the new knowledge based on relevance according to the types in the agent's model and the compatibility with the agent's capabilities (as we achieved in our second contribution in Chapter 5), it may be interesting to extend the scope of work to investigate new techniques to filter new knowledge based on quality and credibility.

In the context-aware knowledge acquisition for planning applications module, after formulating a new goal on the fly that concerns a new object of a new type not previously considered in the agent's model, we delegate the task of finding whether this new goal constitutes an opportunity for the agent to an external planner. Another possibility is to investigate and develop heuristics (planning-based, regression-based or ontology-based) that could tell if a newly generated goal that involves a new object of a new type is actually an opportunity for the agent rather than delegating this task to the planner. Deriving regression-based heuristics to estimate opportunities is possible for pending goals (involving existing objects) that become suddenly possible during execution; however, heuristics to estimate opportunities that involve new objects of new types are yet to be investigated. Another idea is to investigate performing local task repairing through ontologies within a planning-based system and try to overcome the performance issues inherited in that problem.

As mentioned in Chapter 5, it would be interesting to ontologically represent the HTN formalisation. Since HTN high-level tasks are decomposed into more straightforward tasks and primary actions, an agent that benefits from ontological and HTN representations could extend its model with new high-level tasks if it has the primary actions of that high-level task. Consequently, allowing the agent to import new strategies that were not originally modelled in the agent model, yet the agent would be able to perform. Further research is required to enable agents to continuously acquire new behaviour, as in the work of (Vachtsevanou et al. 2020).

In this PhD thesis, we have taken the initiative to empower agents exhibiting a higher level of autonomy. One essential extension that must be further investigated is responsible AI allowing humans to enforce several levels of autonomy for accountability and responsibility, as suggested in the work of (Methnani et al. 2021). Although these are baby steps toward high-level autonomy, multiple scholars have raised concerns over an ongoing accountability gap. Further investigation is required to avoid incidents and to maintain the public's trust in the technology.

Concerning the deliberative context-aware ambient intelligence system for assisted living Homes (detailed in Appendix C). Striving for autonomy can divert attention away from the goal of developing human-centric AI, where human agency is supported and never undermined. Future work must conduct live laboratory demonstrations with senior citizens attended by health professionals and further analyse the demonstration data to adjust the performance, avoid unforeseen inconvenient situations, enhance explainability, and for safe human-robot interactions.

Concerning responsible plan commitment repair, it is interesting to study the utility of metrics, such as stability and commitment, for an agent to autonomously decide when better stability or better commitment plan is worth the computation and the additional plan length after finding the first solution. It may also be interesting to use the plan commitment in single-agent settings inversely; the least committed plans may be used for plan diversity purposes. In addition, a parallel alternative possibility is investigating behaviour trees for plan repair, one problem worth further investigation is finding a heuristic for intelligently pruning branches.

Part IV

Appendices

Appendices

APPENDIX A

Case Study: Tourist Assistant

CONSIDERING the first contribution of this PhD thesis, presented in Chapter 4, this appendix aims to demonstrate the intelligent simulation system's behaviour and validity in a complete smart tourism application domain context. In addition, to provide an example of a tourist making a one-day tour of Valencia city, the jewel of the Mediterranean on Spain's southeastern coast. This chapter uses the system as a smart tourism application that plans a tourist agenda and keeps track of the plan execution. A RS returns the list of places that best fit the individual taste of the tourist. For tourism, we use a method similar to the e-tourism approach introduced in the work of (Ibáñez-Ruiz et al. 2016) to retrieve the list of goals tailored to different tourists' travelling styles. The planner creates a personalised agenda with indications of times and durations of visits. The simulator monitors the execution, periodically updates its internal state with information from open data platforms, and maintains a snapshot of the real-world scenario through live events that communicate sensible environmental changes. The simulator builds a new planning problem when an unexpected change affects the plan execution. The planner arranges the tourist agenda by calculating a new plan to be executed again by the simulator.

A.1 Planning Module

A.1.1 Initial State

The initial state of a planning problem describes the state of the world when the plan starts its execution. Such as reflecting the opening hours of the places to visit, the distances between them, and the user's initial location. The representation depends on the nature of the information. The information whose value is true or known from the beginning of the execution is expressed as fluent state variables, either boolean or numerical. Whereas TILs are used to denote time-stamped changes to the world as the information that is known to happen at a future time.

The variable `(be ?per ?loc)` is used to represent the location of the tourist, e.g., the fluent `(be tourist caro_hotel)`. The pair of TILs `(at 0 (active tourist))` and `(at` τ `(not (active tourist)))` determine the available time of the user for the tour as the difference between the time when the tour starts and finishes. The time indicated in the TILs is relative to the starting time of the plan; that is, `(at 540 (not (active tourist)))` refers to 7pm if the plan starts at 10am. Another pair of TILs is used to define the time window in which the tourist prefers to have lunch. For example, if the preference is between 2pm and 4pm, the TILs are `(at 240 (time_for_eat tourist))` and `(at 360 (not (time_for_eat tourist)))`.

The duration of a particular visit to a monument ?mon for a tourist ?per is defined through the variable (visit_time ?mon ?per), e.g., the fluent (= (visit_time Lonja tourist) 80).

The list of available restaurants is given through the variable (free_table ?r), e.g., the fluent (free_table ricard_camarena). For each restaurant, the author defines the time slot it serves meals, which may depend on the type of restaurant, the kitchen closing time or other factors. Both places to visit and restaurants have an opening hour and a closing hour that are specified by TILs: (at τ (open ?loc)) and (at τ (not (open ?loc))), to indicate when the place/restaurant is no longer available. For example, (at 0 (open Lonja)), (at 540 (not (open Lonja))).

The distance between two locations (?from) and (?to) is defined by the variable (moving_time ?from ?to), which returns the time in minutes needed to travel between (?from) and (?to) by using the travel mode preferred by the user. The time to move between two places is represented through the variable (moving_time ?from ?to), e.g., fluent (= (moving_time caro_hotel Lonja) 9), where the value nine is taken from Google Maps.

A.1.2 Goals and Preferences

The simulator deals with two types of goals: *Hard goals* and *Soft goals (preferences)*. The hard goals are obligatory goals, such as the final destination at which the tourist wants to finish up the tour (be tourist caro_hotel) and that at the end of the tour, the tourist must have eaten at any restaurant (eaten tourist). On the other hand, the preferences are visiting attractions, such as visiting the 15th-century Gothic masterpiece (Lonja), (preference v3 (visited tourist Lonja)).

The objective is to find a plan that achieves all the hard goals while minimising the plan metric to maximise preference satisfaction. A penalty is added to the metric when a preference is not fulfilled. Specifically, the authors define penalties for non-visited Points Of Interest (POIs) and travelling times. Non-visited places are penalised for assisting OPTIC in selecting activities (tourist visits) that have a higher priority for the user. For example, if the priority for visiting `Lonja` is 290, and the sum of the priorities of all the visits is 2530, the penalty for not visiting `Lonja` would be expressed in PDDL as: `( / (* 290 (is-violated v3)) 2530)` as explained in Eq. 4.1. On the other hand for penalising travelling time, the function `(total_moving_time tourist)` accumulates the time spent in movement actions, so if the plan's total makespan is 540, this penalty would be defined in PDDL as: `( / (total_moving_time tourist) 540)`, as explained in Eq. 4.2.

A.1.3 Actions

The author defines three operators in the tourism domain: *move*, *eat* and *visit*. Figure A.1 illustrates the *move* operator from one location to another.

Follows a description of the *move* operator:

- Parameters: the tourist `?per`, the initial location `?from` and the destination `?to`.
- The duration of the operator is set to the estimated/actual time to go from `?from` to `?to`, which is stored in the database.
- The preconditions for this operator to be applicable are:
    - $cond(move)_{\vdash}$: the tourist must be at location `?from`.

```
(:durative−action move
    :parameters (?per − person ?from − location ?to − location)
    :duration (= ?duration (moving_time ?to ?from) )
    :condition (and (at start (be ?per ?from))
    (over all (active ?per)))
    :effect (and (at start (not (be ?per ?from)))
    (at start (walking ?per))
    (at end (be ?per ?to))
    (at end (not (walking ?per)))
    (at end (increase (total_moving_time ?per)
                              (moving_time ?from ?to)))))
```

Figure A.1: Operator `move` of the Tourism Domain

  - $cond(move)_{\leftrightarrow}$: the time window for the available time of the tourist is active during the whole execution of the action instantiated from this operator.

- The effects assert that:

  - $eff(move)_{\vdash}$: the tourist is no longer at the initial location.
  - $eff(move)_{\dashv}$: the tourist is at the new location.
  - $eff(move)_{\dashv}$ the time spent in *move* actions is modified according to the movement duration.
  - The fluent `(walking ?per)` is asserted at the start and deleted at the end to indicate the tourist's position in case of failure.

In this particular example, we only consider walking; however, other transport modes according to the tourist's preferences can be included; e.g., cycling, driving, and public transport.

```
(:durative—action visit
    :parameters (?per — person ?mon — monument)
    :duration (= ?duration (visit_time ?mon ?per))
    :condition
    (and (at start (be ?per ?mon))
    (over all (be ?per ?mon))
    (over all (active ?per))
    (over all (open ?mon))
    :effect (and (at end (visited ?per ?mon))))
```

Figure A.2: Operator `visit` of the Tourism Domain

The operator *v*isit for visiting a monument is defined in Figure A.2. The parameters are the monument to visit `?mon` and the tourist `?per`. The duration of the operator is defined by the variable `(visit_time ?mon ?per)`. The conditions for this operator to be applicable are: (1) the tourist is at the monument during the whole execution of the action instantiated from this operator; (2) the monument is open during the whole execution of the action instantiated from this operator and (3) the time window for the available time of the tourist is active. The effect is that the monument is visited.

The Operator *e*at represents the eating activity and is defined in Figure A.3.

The parameters in the `eat` operator are the tourist and the restaurant. The duration of the action is defined by the variable `(time_for_eat ?pers)` and specified by the user. The following conditions must hold to apply this action: (1) the tourist is at the restaurant during the whole execution of the action;

```
(:durative−action eat
    :parameters (?per − person ?loc − restaurant)
    :duration   (= ?duration (eat_time ?per ?loc))
    :condition (and (at start (free_table ?loc))
    (at start (be ?per ?loc))
    (over all (be ?per ?loc))
    (over all (active ?per))
    (over all (open ?loc))
    (over all (time_for_eat ?per)))
    :effect (and (at end (eaten ?per))))
```

Figure A.3: Operator `eat` of the Tourism Domain

(2) the restaurant is open during the whole execution of the action; (3) the restaurant has a free table and (4) both the time window for the time to have lunch defined by the user and the available time are active. The effects of the operator assert that the tourist has had lunch.

Finally, some implementation design decisions are worth mentioning. Such as the decision to assert the fluent `(walking ?per)` as a start effect of the *move* operator. This is not necessary for the rest of the tourism domain model operators (such as visiting a POI or eating in a restaurant) as they do not manipulate the location of this tourist. In addition, an alternative design choice would have been to insert `(walking ?per ?loc1 ?loc2)`; this would help a human operator that is debugging the execution to know the location of the tourist. However, it will not be beneficial to the simulator itself, as the simulator can know the tourist's location after the failure by rolling back the start effects of the failed action (as explained in Step 1.1: Updating fluents in Section 4.5.3). Other alternative approaches can be used, such as sensing actions.

A.2 Valencia Tour

This section aims to demonstrate the behaviour of the simulation system with a representative example of a tourist making a one-day tour of Valencia city.

Table A.1: Recommended Places, Initial and Adapted penalties, RV, RV’ and RV”.

Initialised before execution		**Adapted during execution**	
Places	**RV**	**RV’**	**RV”**
Cathedral	280	280	280
Central market	270	**600**	**600**
Lonja	290	**600**	**600**
Serrano towers	250	—	—
City of arts and sciences	280	280	280
Oceanografic	300	300	300
Bioparc	210	210	210
Quart towers	200	—	—
Viveros garden	250	—	—
Town hall	200	**600**	**600**

Initialised by a set of tourist profiles and restaurants, the system retrieves a set of recommended places (Table A.1, column 1). Utilising tourist profiles, an RS identifies recommended activities. Places on this list are accompanied by a recommendation value RV (Table A.1, column 2) based on the tourists’ level of interest. Based on these values, the planning module generates a plan that suits the tourist’s tastes. On the other hand, RV’ and RV” are the adapted penalties after execution failures. As will be seen in the following example, when the simulator reformulates the planning problem upon failures, it increases the penalties for the pending preferences by assigning a relatively higher priority to these pending goals (twice as much as the maximum penalty among all goals). This encourages the planner to prioritise goals already included in the original plan over those that were not (As explained in Step 2: Updating preferences of Section 4.5.3).

Figure A.4 shows a snapshot of the tour (plan) calculated by OPTIC for the tourist. In the snapshot, the red location icons indicate the visits included in the plan. The tour begins at the hotel in which the tourist is staying (green location icon) and consists of six visits to monuments (red icons) and one meal at a restaurant (orange icon).

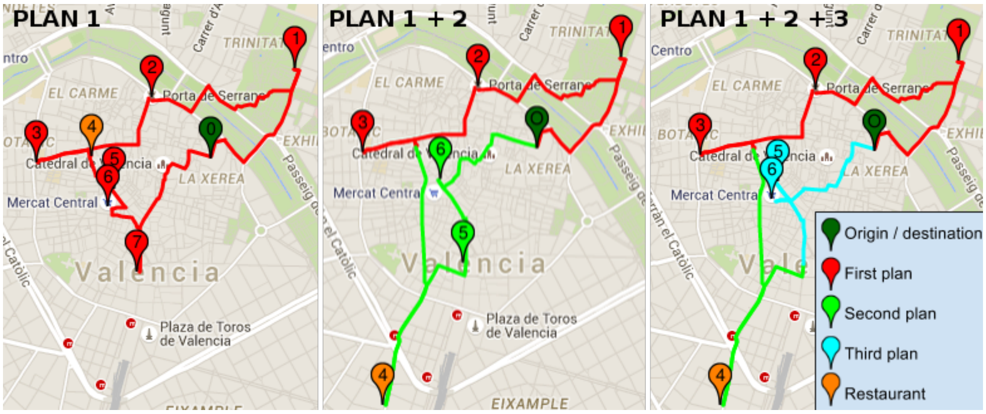


Figure A.4: The Three Simulated Plans. Icons in PLAN1: (0) Caro hotel, (1) Viveros Garden, (2) Serrano towers, (3) Quart towers, (4) El Celler del Tossal (RESTAURANT), (5) Lonja, (6) Central market, (7) Town hall. Icons in PLAN2: 1,2,3 are the same as PLAN1, (4) Pederniz (RESTAURANT), (5) Town hall, (6) Lonja. Icons in PLAN3: 1,2,3 and 4 are the same as PLAN2, (5) Lonja, (6) Central market

The simulator starts the plan execution simulation of PLAN1 (shown in Fig. A.5) with the above information. Let us assume that at the time 1:55 pm, a live event is received `(at 235 (not (free_table el_celler_del_tossal)))`, indicating that the restaurant chosen by OPTIC *el celler del tossal* becomes unexpectedly full and has no available table. When the live event arrives, the

a_1 : 0.001 : (*move tourist caro_hotel viveros_garden*)[11.000]
a_2 : 11.002 : (*visit tourist viveros_garden*)[75.000]
a_3 : 86.002 : (*move tourist viveros_garden serrano_towers*)[11.000]
a_4 : 97.003 : (*visit tourist serrano_towers*)[80.000]
a_5 : 177.003 : (*move tourist serrano_towers quart_towers*)[11.000]
a_6 : 188.004 : (*visit tourist quart_towers*)[30.000]
a_7 : 236.000 : (*move tourist quart_towers el_celler_del_tossal*)[4.000]
$\mathbf{a_8}$: 240.001 : (eat tourist el_celler_del_tossal)[90.000]
a_9 : 330.001 : (*move tourist el_celler_del_tossal lonja*)[4.000]
a_{10} : 334.002 : (*visit tourist lonja*)[80.000]
a_{11} : 414.002 : (*move tourist lonja central_market*)[1.000]
a_{12} : 415.003 : (*visit tourist central_market*)[70.000]
a_{13} : 485.003 : (*move tourist central_market town_hall*)[6.000]
a_{14} : 491.004 : (*visit tourist town_hall*)[35.000]
a_{15} : 526.004 : (*move tourist town_hall caro_hotel*)[13.000]

Figure A.5: The temporal plan: PLAN1

tourist has already visited the first three monuments (1. Viveros garden; 2. Serrano towers; 3. Quart towers). At time instance 240.001 the simulator detects a failure because the action `(eat tourist el_celler_del_tossal)` is not executable. Therefore, the simulator reformulates a new planning problem as follows:

1. *Initial state*:

   - The current location of the tourist is the point at which the previous plan failed, i.e., the restaurant *el celler del tossal*.
   - Since the new simulation time is reset to zero, the simulator updates the time of the TILs in the current state. Namely, the opening and closing time of places, the time slot for having lunch and the TIL

`(at` τ `(not (active tourist)))`, where τ is set to the new time the tourist must get back to the hotel from time zero.

- Additionally, the fluent `(total_moving_time tourist)` is updated with the total time the tourist has spent in moving around the city.

2. *Goals*: Places already visited (the first three monuments) are removed from the list of goals. The new set of goals includes two lists: (a) The *pending goals* of the failed plan; that is, (4. have lunch) and three remaining monuments that have not yet been visited by the tourist (5. Lonja, 6. Central market, 7. Town Hall). (b) The original problem's goals that were not included in the plan.

   Regarding penalties of the goals, the list of goals in (b) is included in the new planning problem with their original recommended values (see the non-bold values RV' in the 2nd column of Table A.1). For the pending goals of (a) in the previous Goals list, the penalty for these goals is increased (the bold values RV' in the third and fourth columns of Table A.1 for the three pending monuments) according to the stability concept explained in Section 4.5.3.

Figure A.4 (middle part in green colour) shows a demonstrative of a new plan obtained due to the simulator invoking OPTIC and obtaining the new plan, PLAN2 (shown in Figure A.6).

A few things must be noted in this new plan, PLAN2:

1. OPTIC suggests a new restaurant (orange icon, labelled with number 4) relatively far away from the previous restaurant. The reason is that we have only included in the planning problem the 10-top restaurants in Valencia suggested by *Trip Advisor*, and the closest one to the prior restaurant is the one shown in the second map.

a_8 : 240.002 : (*move tourist el_celler_del_tossal el_pedernil*)[22.000]

a_9 : 262.003 : (*eat tourist el_pedernil*)[90.000]

a_{10} : 352.004 : (*move tourist el_pedernil town_hall*)[16.000]

$\mathbf{a_{11}}$: **368.005** : (**visit tourist town_hall**)[**35.000**]

a_{12} : 403.006 : (*move tourist town_hall lonja*)[6.000]

a_{13} : 409.007 : (*visit tourist lonja*)[80.000]

a_{14} : 489.008 : (*move tourist lonja caro_hotel*)[9.000]

Figure A.6: The temporal plan: PLAN2

2. The places included in the new plan and the paths between them are marked with green. The new plan maintains the visit to Town Hall (now represented by the green icon numbered 5) and the visit to Lonja (now represented by the green icon with the number 6). However, this new plan has discarded the Central Market visit, likely due to the long distance to the new restaurant.

The simulation continues. During the tourist visit to the Town Hall, it is announced that it will close 140 minutes earlier than the currently scheduled closing time; a live event `(at 380 (not (open town_hall)))` is received. A new failure is detected in the middle of the execution of `(visit tourist town_hall)` due to a violation of an *overall* condition. The simulator reverts to the previous state (rolling back the last visit action) but preserves the simulation time during the process. Then, in the new reformulated problem, the tourist is located at the Town Hall, has not visited the Town Hall, and the live event causes the fluent `(open town_hall)` to be removed from the initial state. The goal `(visited tourist town_hall)` will be included in the new problem to attempt to maintain preferences stability. However, since the Town Hall is no longer open for visits, it will not be included in the new plan generated by OPTIC. The penalties of the goals for this third problem are shown in column 3 of Table A.1.

The third plan, PLAN3, is highlighted in light blue colour on the right side of Figure A.4). This new plan suggests visiting (5. Lonja) and then (6. Central market) (that was eliminated from the second plan), and finally, the tour ends with the tourist returning to the hotel (as shown in Figure A.7).

a_{11} : 368.006 : $(move\ tourist\ town\_hall\ lonja)$[6.000]
a_{12} : 374.007 : $(visit\ tourist\ lonja)$[80.000]
a_{13} : 454.008 : $(move\ tourist\ lonja\ central\_market)$[1.000]
a_{14} : 455.009 : $(visit\ tourist\ central\_market)$[70.000]
a_{15} : 525.010 : $(move\ tourist\ central\_market\ caro\_hotel)$[9.000]

Figure A.7: The temporal plan: PLAN3

A.3 Conclusion

In this chapter, the author has demonstrated how the intelligent system for execution simulation in a dynamic environment (introduced in Chapter 4) is used as a tourist assistant for the e-tourism application domain.

It offers a personalised plan of POIs visits suited to the tourist's interest. It also handles the soft preferences regarding the POIs the tourist wishes to visit. More importantly, this intelligent system gives the advantage of providing the tourist with an enhanced tourism experience in a way not typically introduced by trip generators. Specifically, not only for generating the trip itself but also online during the trip; after generating a plan tailored to the tourist visiting preferences, the plan execution is monitored in real-time, thus dynamically reacting to failures due to the changes in the environment, such as when a restaurant or a POI gets closed or fully booked. The system generates a new plan for the tourist taking into consideration the following:

- The tourist's hard goals (activities) that have already been performed from the original plan and the remaining hard goals.
- The tourist's soft goals in the new plan when a failure occurs, the system gives more weight to the pending visits included in the original plan to meet the tourist's expectations as much as possible and allow some sense of stability and control.
- The tourist's constraint restrictions, such as the remaining time for the tourist's one-day tour and lunchtime.

The recommendation values of the POIs used as preferences' penalties to be satisfied by the planner are supplied using a recommendation system according to the tourists' profiles, similarly to the approach used in the work of (Ibáñez-Ruiz et al. 2016). On the other hand, for other domains, the preferences penalties can be read from a data source or supplied explicitly in the PDDL problem file, which the simulator reads as input. In other words, there is no need to use a recommendation system in other application domains. Although a specialised GUI was developed for the tourism application domain, the system is domain-independent. The output is shown in a console window (as demonstrated in Section 4.5.4) and can be written into a file for logging.

A point worth mentioning is that the simulator can scale for execution in large cities with many POIs. POIs are modelled in the PDDL domain model as instance objects of locations, and the distances between locations are modelled as numeric fluents. Therefore, a high number of POIs or restaurants defined in the problem instance does not increase the complexity of finding a solution to the planning problem; on the contrary, it makes it easier for the planner to find solutions and alternative solutions when a failure occurs.

APPENDIX B

Case Study: Context-aware Knowledge Acquisition for Planning Applications

BASED on the second contribution of this PhD thesis, presented in Chapter 5, this appendix aims to demonstrate the behaviour and the validity of the context-aware knowledge acquisition approach for planning applications in two different application domains, namely, a repairing agency application and a smart tourist assistant application. We demonstrate in a step-by-step fashion how an autonomous agent draws upon the richness and expressivity of a standard ontology representation, semantic measures and ontology alignment to be able to accommodate newly acquired objects of new types into the planning task specification. Consequently, the anticipation of opportunities is facilitated as these new objects may subsequently trigger the formulation of a goal that induces a better-valued plan, as shown in this appendix or can be used to repair failures, as shown in Appendix C.

B.1 A Repairing Agency Application

Consider a repair agency scenario in which a robot, `av`, in a warehouse has a one-day maintenance task for several large kitchen appliances received by the agency.

B.1.1 The Planning Task

The robot `av` can work on one appliance at a time. The warehouse has three areas; `transit` area for items that require maintenance, `inspect` area where maintenance is performed, and `storage` area for items that have already been maintained. Initially, the system has a set of major kitchen appliances categories; `television`, `refrigerator` and `dishwasher`.

The operations that `av` can perform are `move`, `repair`, `load`, and `unload`. `move` is an operator that allows `av` to move between the warehouse areas. `load` is an operator that allows the robot to carry an object at an area, and similarly, `unload` allows the robot to unload the object at an area. `repair` is an operator that allows `av` to perform a maintenance operation on a major kitchen appliance at the `inspect` area.

The information of the initial state includes:

- `av`'s start location `(be av area_storage)`, status as `(empty av)` and operational hours between 10:00 and 22:00 controlled by the *TILs* `(at 0 (active av))` and `(at 720 (not (active av)))`.

- The fluents representing the the durations of movement between the warehouse areas `(= (moving_time area_storage area_inspect) 5)`, `(= (moving_time area_storage area_transit) 5)`, `(= (moving_time area_inspect area_transit) 5)`, `(= (moving_time area_inspect area_-storage) 5) ,(= (moving_time area_transit area_storage) 5)`, `(= (moving_time area_transit area_inspect) 5)` maintenance time `(= (repair_time) 80)`, loading time and unloading time `(= (load_unload_-time) 1)`.

- The appliances' initial locations at the inspection area; `(be washer_ID02 area_transit)`, `(be washer_ID101 area_transit)`, and `(be refrigerator_-ID03 area_transit)`.

- The appliances that require repairing; `(require_repair washer_ID02)`, `(require_repair washer_ID101)` and `(require_repair refrigerator_-ID03)`.

Where `washer_ID02` and `washer_ID101` are two objects of the PDDL type `dishwasher`, and `refrigerator_ID03` is an object of the PDDL type `refrigerator`. The goals are to repair and deliver any appliance that requires repairing and deliver it to the storage area. Since we do not deal with conditional effects or derived predicates, we added a dummy operator with a zero duration that asserts that an appliance is delivered if it is repaired and it is in the storage area.

The scenario is formulated as a planning task ϕ, and a planner is invoked to solve this task. The plan to solve ϕ (π_1 shown in Figure B.1) is calculated by the planner OPTIC and consists of 27 actions; the robot av moves from its start location `(area_storage)` to `(area_transit)`, loads an item, moves to `(area_inspect)`, unloads the item to be repaired, repairs the item, loads the item, moves to (area_storage), unloads the item, and the item is delivered; the previous rotation (8+1 dummy) applies for each of the three appliances that require repairing.

a_1 : 0.0003 : ($move\ av\ area\_storage\ area\_transit$)[5.0000]
a_2 : 5.0005 : ($load\ av\ washer\_id02\ area\_transit$)[1.0000]
a_3 : 6.0008 : ($move\ av\ area\_transit\ area\_inspect$)[5.0000]
a_4 : 11.0010 : ($unload\ av\ washer\_id02\ area\_inspect$)[1.0000]
a_5 : 12.0013 : ($repair\ av\ washer\_id02\ area\_inspect$)[80.0000]
a_6 : 91.0015 : ($load\ av\ washer\_id02\ area\_inspect$)[1.0000]
a_7 : 92.0017 : ($move\ av\ area\_inspect\ area\_storage$)[5.0000]
a_8 : 97.0020 : ($unload\ av\ washer\_id02\ area\_storage$)[1.0000]
a_9 : 98.0023 : ($dummy\ washer\_id02\ area\_storage$)[0.0000]
a_{10} : 98.0025 : ($move\ av\ area\_storage\ area\_transit$)[5.0000]
a_{11} : 103.0027 : ($load\ av\ washer\_id101\ area\_transit$)[1.0000]
a_{12} : 104.0030 : ($move\ av\ area\_transit\ area\_inspect$)[5.0000]
a_{13} : 109.0033 : ($unload\ av\ washer\_id101\ area\_inspect$)[1.0000]
a_{14} : 110.0035 : ($repair\ av\ washer\_id101\ area\_inspect$)[80.0000]
a_{15} : 189.0038 : ($load\ av\ washer\_id101\ area\_inspect$)[1.0000]
a_{16} : 190.0040 : ($move\ av\ area\_inspect\ area\_storage$)[5.0000]
a_{17} : 195.0043 : ($unload\ av\ washer\_id101\ area\_storage$)[1.0000]
a_{18} : 196.0045 : ($dummy\ washer\_id101\ area\_storage$)[0.0000]
a_{19} : 196.0047 : ($move\ av\ area\_storage\ area\_transit$)[5.0000]
a_{20} : 201.0050 : ($load\ av\ refrigerator\_id03\ area\_transit$)[1.0000]
a_{21} : 202.0052 : ($move\ av\ area\_transit\ area\_inspect$)[5.0000]
a_{22} : 207.0055 : ($unload\ av\ refrigerator\_id03\ area\_inspect$)[1.0000]
a_{23} : 208.0058 : ($repair\ av\ refrigerator\_id03\ area\_inspect$)[80.0000]
a_{24} : 287.0060 : ($load\ av\ refrigerator\_id03\ area\_inspect$)[1.0000]
a_{25} : 288.0063 : ($move\ av\ area\_inspect\ area\_storage$)[5.0000]
a_{26} : 293.0065 : ($unload\ av\ refrigerator\_id03\ area\_storage$)[1.0000]
a_{27} : 294.0068 : ($dummy\ refrigerator\_id03\ area\_storage$)[0.0000]

Figure B.1: The temporal plan: π_1

In addition, the repair agency may unexpectedly receive new objects that require maintenance from other agents during π_1's execution. The agent must decide whether to accept or ignore such objects.

B.1.2 Execution Simulation and Context-aware Knowledge Acquisition

The simulation system presented in Chapter 4 is utilised to simulate the execution of π_1. The simulator starts the execution simulation. After repairing and delivering the appliance `washer_ID02` (after a_9 in π_1 shown in Figure B.1), new information is received from a different delivery agent `(be iphone_ID7500 area_transit)` that includes the new object `iphone_ID7500` $\notin O$. The system requests the type of the new object from the agent that delivered it and finds t=`mobile_phone` $\notin T$.

The system creates the preliminary ontological representation η_ϕ (explained in Section 5.5.1) that represents the types of ϕ as shown in Figure B.2.

```
(:types
agent - object
robot - agent
location - object
inspection - location
storage - location
transit - location
major_appliance - object
dishwasher - major_appliance
refrigerator - major_appliance
television - major_appliance)
```

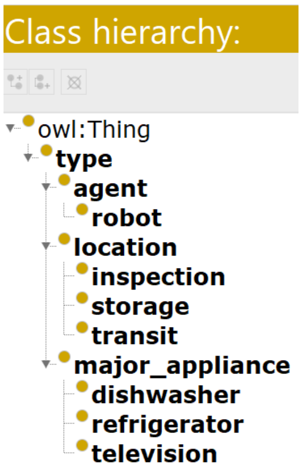


Figure B.2: PDDL types T in ϕ and the corresponding OWL representation Ω_T in η_ϕ

The left part of Figure B.2 shows the types T of the agent's planning task ϕ, and the right part shows the corresponding Ω_T in η_ϕ. For instance, the type `dishwasher` is represented as major_appliance :: dishwasher to denote that the class dishwasher is a subclass of the class major_appliance. It can be observed that the planning task types are arranged in a reasonable hierarchy and that their OWL representation follows this hierarchy truthfully.

Second, the system retrieves the types part of several semi-cooperative agents planning tasks from online repositories $\Delta = \{A, B, C\}$ (shown in Figures B.3, B.4, and B.5, respectively) from online repositories.

```
(:types
agent item location - object
robot - agent
dishwasher fridge kitchen_range mobile_phone tv - item
inspection storage transit - location)
```

Figure B.3: T_{ϕ_A} of a remote planning task ϕ_A

```
(:types
agent item location - object
robot - agent
freezer fridge kitchen_range range_hood tv - item
inspection storage transit - location)
```

Figure B.4: T_{ϕ_B} of a remote planning task ϕ_B

```
(:types
person accommodation attraction restaurant - object
hotel - accommodation
aquarium zoo tower architectural_building park cathedral - attraction)
```

Figure B.5: T_{ϕ_C} of a remote planning task ϕ_C

The system creates their preliminary ontological representation R^{Δ} respecting their corresponding hierarchies of types, as shown in Figure B.6.

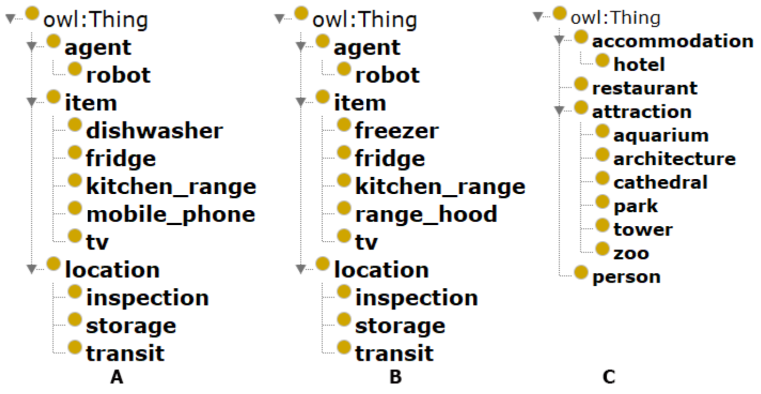
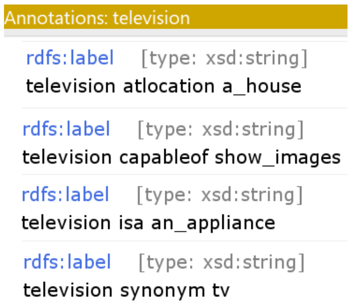


Figure B.6: R^{Δ} preliminary representation

The agent's own ontological representation η_{ϕ} and the remote ontological representations of the remote planning tasks R^{Δ} are augmented using ConceptNet (as explained in Section 5.5.2) as a standard means to describe concepts, so the lexical information in each class of the ontology comes not only from the local name of the term but also from annotations imported from ConceptNet relations and classes. A small portion of the annotations attached to the class television in η_{ϕ} is shown in Figure B.7.



Annotations: television

rdfs:label [type: xsd:string]
television atlocation a_house

rdfs:label [type: xsd:string]
television capableof show_images

rdfs:label [type: xsd:string]
television isa an_appliance

rdfs:label [type: xsd:string]
television synonym tv

Figure B.7: A sample of the annotations describing an OWL class within an ontology

VSM distance (described in Section 5.5.3) is calculated between η_ϕ and each ontology in R^Δ; the distances are 0.79, 0.78, and 0.38, respectively. The system tries to find $t = mobile\_phone$ in R^Δ and creates $\mathsf{R'}_t = \{A\}$, the only remote ontology with the type t and high similarity value (0.79) to η_ϕ. If A were to be recognised as η_ω and used for the alignment, like in (Babli, Marzal, et al. 2018), the result would be erroneously integrating `iphone_ID7500` and generating a goal the agent cannot manage with its operators. Instead, in our approach, we extend the representation of η_ϕ and $\mathsf{R'}_t = \{A\}$, as explained in Section 5.5.4, to account for the planning dynamics by representing the variables V and the head parts of the operators OP. Consequently, allowing the agent to distinguish new objects manageable by the agent's capabilities (variables and operators).

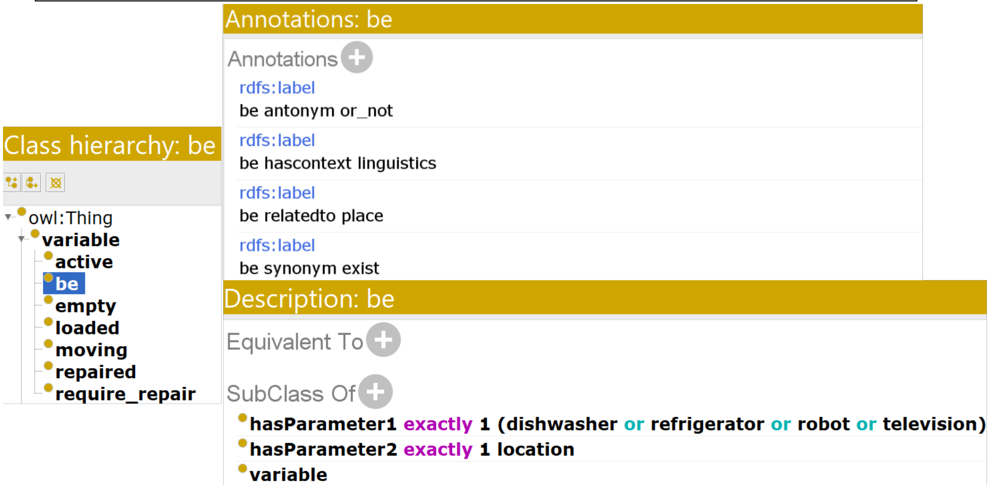
The top part of Figure B.8 shows V of ϕ.

The bottom part shows the corresponding ontological representation of Ω_V and the description for $v =$ `(be ?locatable - (either dishwasher refrigerator robot television) ?loc - location)`, with $\mathsf{J}_{\mathsf{hasParameter}_{1\ldots2}}$ to specify $arg_1 = dishwasher \lor refrigerator \lor robot \lor television$ and $arg_2 = location$, and a sample of the annotation labels attached to the class be, brought from ConceptNet.

Similarly, the top part of Fig B.9 shows the head parts of OP of ϕ.

The bottom part of the figure shows the corresponding ontological representation of Ω_{OP} and the description for the head of $op =$ `(repair ?robot - robot ?item - (either television refrigerator dishwasher)`, with OWL object properties $\mathsf{J}_{\mathsf{hasParameter}_{1\ldots3}}$ to specify $arg_1 = robot$, $arg_2 = dishwasher \lor refrigerator \lor robot \lor television$ and $arg_3 = inspection$, in addition to a sample of the annotation labels attached to the class repair, brought from ConceptNet.

```
(:predicates
(be ?locatable - (either dishwasher refrigerator robot television)
    ?loc - location)
(active ?robot - robot)
(moving ?robot - robot ?area1 - location ?area2 - location)
(repaired ?item - (either dishwasher refrigerator television))
(require_repair ?item - (either dishwasher refrigerator television))
(loaded ?item - (either dishwasher refrigerator television) ?robot - robot)
(empty ?robot - robot))
```


Annotations: be

Annotations

rdfs:label
be antonym or_not

rdfs:label
be hascontext linguistics

rdfs:label
be relatedto place

rdfs:label
be synonym exist

Class hierarchy: be

- owl:Thing
  - variable
    - active
    - be
    - empty
    - loaded
    - moving
    - repaired
    - require_repair

Description: be

Equivalent To

SubClass Of

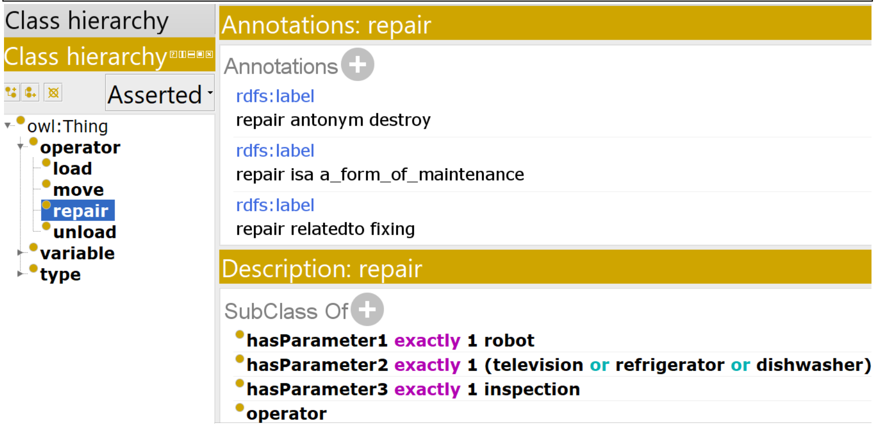
- hasParameter1 exactly 1 (dishwasher or refrigerator or robot or television)
- hasParameter2 exactly 1 location
- variable

Figure B.8: PDDL variables V in ϕ and their OWL representation Ω_V in η_ϕ

On the other hand, Figure B.10 shows the variables of the remote planning ontology V_A and the corresponding ontological representation in A, in addition to the annotations sample from ConceptNet.

One can notice in Figure B.10 that the remote agent has specialised variables `(motherboard ?mobile - mobile_phone)`, `(keyboard ?mobile - mobile_phone)` and `(battery ?mobile - mobile_phone)` associated with the type `mobile_-phone`.

```
(:action schemas heads
(move ?robot - robot ?location1 - location ?location2 - location)
(load ?robot - robot ?item - (either television refrigerator dishwasher)
    ?location - location)
(repair ?robot - robot ?item - (either television refrigerator dishwasher)
    ?inspection_area - inspection)
(unload ?robot - robot ?item - (either television refrigerator dishwasher)
    ?location - location)
```



rdfs:label
repair antonym destroy

rdfs:label
repair isa a_form_of_maintenance

rdfs:label
repair relatedto fixing

hasParameter1 exactly 1 robot
hasParameter2 exactly 1 (television or refrigerator or dishwasher)
hasParameter3 exactly 1 inspection
operator

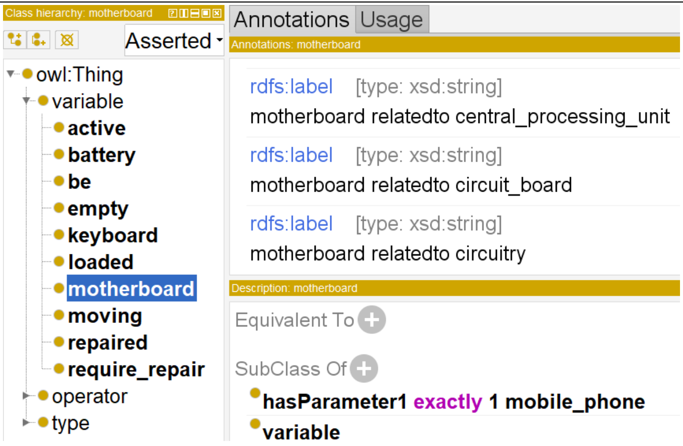
Figure B.9: PDDL Operators heads OP in ϕ and their OWL representation Ω_{OP} in η_ϕ

Figure B.11 shows the head parts OP_A and their corresponding ontological representation in A, in addition to the annotations sample from ConceptNet. Besides the operator `repair1`. One can notice in Figure B.11 that the remote agent has a specialised operator, which head is `(repair2 ?robot - robot ?item - mobile_phone ?inspection_area - inspection)` for repairing mobile phones.

The agent applies the TSM distance (explained in Section 5.5.5), the distance is found to be equal to 0.39, and the agent deems `iphone_ID7500` unmanageable.

The simulation of π_1 continues. After `av` has repaired and delivered the appliance `washer_ID101` (after a_{18} in π_1 shown in Figure B.1), a new information

```
(:predicates
(be ?locatable - (either dishwasher fridge robot kitchen_range tv
    mobile_phone) ?loc - location)
(active ?robot - robot)
(moving ?robot - robot ?area1 - location ?area2 - location)
(repaired ?item - (either dishwasher fridge kitchen_range tv mobile_phone))
(require_repair ?item - (either dishwasher fridge kitchen_range tv
    mobile_phone))
(loaded ?item - (either dishwasher fridge kitchen_range tv
    mobile_phone) ?robot - robot)
(empty ?robot - robot)  (motherboard ?mobile - mobile_phone)
(keyboard ?mobile - mobile_phone) (battery ?mobile - mobile_phone))
```



- owl:Thing
  - variable
    - active
    - battery
    - be
    - empty
    - keyboard
    - loaded
    - motherboard
    - moving
    - repaired
    - require_repair
  - operator
  - type

rdfs:label [type: xsd:string]
motherboard relatedto central_processing_unit

rdfs:label [type: xsd:string]
motherboard relatedto circuit_board

rdfs:label [type: xsd:string]
motherboard relatedto circuitry

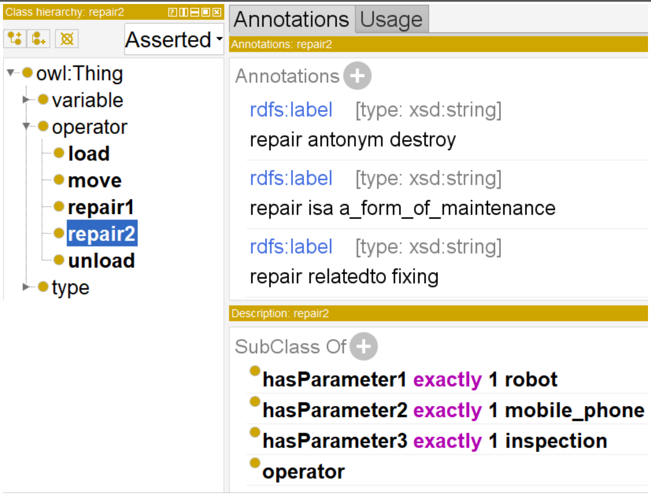
hasParameter1 exactly 1 mobile_phone
variable

Figure B.10: V in ϕ_A and their OWL representation

is received (be bosch bosch_ID3400 area_transit, that includes the new object bosch_ID3400 $\notin \mathcal{O}$ of type t=kitchen_range $\notin T$.

The system creates $\mathrm{R'}_t = \{A, B\}$, and the TSM distances with respect to η_ϕ are respectively: 0.39, 0.64. Subsequently, B is used as η_ω to position kitchen_-range in η_ϕ by applying alignment with neighbourhood constraint (explained in Section 5.6.1). If there were multiple ontologies with high TSM similarity

```
(:action schemas heads
(move ?robot - robot ?location1 - location ?location2 - location)
(load ?robot - robot ?item - (either dishwasher fridge kitchen_range tv
    mobile_phone) ?location - location)
(repair1 ?robot - robot ?item - (either dishwasher fridge kitchen_range tv)
    ?inspection_area - inspection)
(repair2 ?robot - robot ?item - mobile_phone ?inspection_area - inspection)
(unload ?robot - robot ?item - (either dishwasher fridge kitchen_range tv
    mobile_phone) ?location - location))
```



rdfs:label [type: xsd:string]
repair antonym destroy

rdfs:label [type: xsd:string]
repair isa a_form_of_maintenance

rdfs:label [type: xsd:string]
repair relatedto fixing

hasParameter1 exactly 1 robot
hasParameter2 exactly 1 mobile_phone
hasParameter3 exactly 1 inspection
operator

Figure B.11: OP in ϕ_A and their OWL representation

(which is not the case for this example), the agent would apply the semantic variance to select the most specialised ontology (explained in Section 5.6).

When applying the alignment, the system finds that the matching percentage is 66% according to siblings (siblings(kitchen_range) shown in Figure B.7), and therefore positions `kitchen_range` in η_ϕ as major_appliance :: kitchen_range. A new entry `kitchen_range - major_appliance` is added to T, and as $args$ and $pars$ for relating V and OP, respectively, then `bosch_ID3400` is added to O.

The information required for integrating `bosch_ID3400` in ϕ is automatically identified, requested, and added to I.

Since `kitchen_range` is a sibling of a type involved in a goal $g \in G$, thus, the system formulates new g'_1=`(repaired bosch_ID3400)`, g'_2=`(delivered bosch_ID3400)` and $G' = g'_1 \cup g'_2 \cup G$. Finally, the system updates the current state when the new information was received, and the planner is called to generate a new plan (π_2 shown in Figure B.12), allowing the robot to repair and deliver the original set of items plus the new item.

$a_{19} : 196.0047 : (move\ av\ area\_storage\ area\_transit)[5.0000]$
$a_{20} : 201.0050 : (load\ av\ refrigerator\_id03\ area\_transit)[1.0000]$
$a_{21} : 202.0052 : (move\ av\ area\_transit\ area\_inspect)[5.0000]$
$a_{22} : 207.0055 : (unload\ av\ refrigerator\_id03\ area\_inspect)[1.0000]$
$a_{23} : 208.0058 : (repair\ av\ refrigerator\_id03\ area\_inspect)[80.0000]$
$a_{24} : 287.0060 : (load\ av\ refrigerator\_id03\ area\_inspect)[1.0000]$
$a_{25} : 288.0063 : (move\ av\ area\_inspect\ area\_storage)[5.0000]$
$a_{26} : 293.0065 : (unload\ av\ refrigerator\_id03\ area\_storage)[1.0000]$
$a_{27} : 196.0045 : (dummy\ refrigerator\_id03\ area\_storage)[0.0000]$
$a_{28} : 196.0047 : (move\ av\ area\_storage\ area\_transit)[5.0000]$
$a_{29} : 201.0050 : (load\ av\ bosch\_id3400\ area\_transit)[1.0000]$
$a_{30} : 202.0052 : (move\ av\ area\_transit\ area\_inspect)[5.0000]$
$a_{31} : 207.0055 : (unload\ av\ bosch\_id3400\ area\_inspect)[1.0000]$
$a_{32} : 208.0058 : (repair\ av\ bosch\_id3400\ area\_inspect)[80.0000]$
$a_{33} : 287.0060 : (load\ av\ bosch\_id3400\ area\_inspect)[1.0000]$
$a_{34} : 288.0063 : (move\ av\ area\_inspect\ area\_storage)[5.0000]$
$a_{35} : 293.0065 : (unload\ av\ bosch\_id3400\ area\_storage)[1.0000]$
$a_{36} : 294.0067 : (dummy\ bosch\_id3400\ area\_storage)[0.0000]$

Figure B.12: The temporal plan: π_2

Similarly, if new objects of the new types `range_hood`, `fridge`, `freezer` or `tv` are received during the execution, the agent would be able to integrate the new objects of these new types, formulates new goals and generate new plans as long as the planner can generate plans for these opportunistic goals.

At the end of the robot's operational day, the agent can filter out irrelevant or unmanageable objects and extends its planning knowledge autonomously to integrate relevant and manageable objects and generate new opportunistic goals.

B.2 A Tourist Assistant Application

We revisit the *tourism* example, which considered a tourism planning task ϕ, in which a tourist wishes to make a one-day tour to visit several points of interest (POIs) in Valencia city, the jewel of the Mediterranean on Spain's southeastern coast.

B.2.1 The Planning Task

Initially, the system retrieves a set of recommended places (attractions) according to the user profile and a set of restaurants. The predefined categories of types T are shown in Figure B.13. As noted, T is defined in a reasonable hierarchy.

```
(:types
person accommodation attraction restaurant - object
hotel - accommodation
aquarium architecture cathedral park tower zoo - attraction)
```

Figure B.13: PDDL types,T, of the tourist assistant domain Dom

The set of variables in V is shown with comments explaining each variable in Figure B.14.

```
(:predicates
    ; A tourist is at a location
    (be ?per - person ?loc - (either accommodation attraction restaurant))
    ; A tourist has visited an attraction
    (visited ?per - person ?loc - attraction)
    ; A tourist has eaten
    (eaten ?per - person)
    ; An attraction or a restaurant is open
    (open ?loc - (either attraction restaurant))
    ; To specify the tourist eating time window
    (time_for_eat ?per - person)
    ; A restaurant has free tables
    (free_table ?loc - restaurant)
    ; To specify the one-day tour time window
    (active ?per - person)
    ; The tourist is walking
    (walking ?per - person))
(:functions
    ; Travelling time between two locations
    (moving_time ?to - (either accommodation attraction restaurant)
        ?from - (either accommodation attraction restaurant))
    ; Visit duration of an attraction
    (visit_time ?mo - attraction ?per - person)
    ; Eating duration for a tourist
    (eat_time ?per - person ?rest - restaurant)
    ; The total moving time of a person to minimise travelling time
    (total_moving_time ?per - person))
```

Figure B.14: The variables V of the tourism planning task

The operations that a tourist can perform in this example domain are the generic operations `move`, `visit` and `eat`. `move` is a generic operator that allows a tourist to move between locations as long as the time is within the one-day defined time window of the tour. `visit` is a generic operator that allows a tourist to visit an attraction as long as the attraction is open, the tourist is at the attraction, and the time is within the one-day defined time window. `eat` is a generic operator that allows a tourist to eat at a restaurant as the tourist is at the restaurant, the restaurant is not closed or fully booked, and the time is within the one-day defined time window as well. The operators OP heads of the tourism planning task are shown in Figure B.15.

```
; Allows the tourist to move between locations
(:durative-action move :parameters (?per - person
    ?to - (either accommodation attraction restaurant)
    ?from - (either accommodation attraction restaurant))
; Allows the tourist to visit an attraction
(:durative-action visit :parameters (?per - person ?mon - attraction)
; Allows the tourist to eat at a restaurant
(:durative-action eat :parameters (?pers - person ?loc - restaurant)
```

Figure B.15: The operators OP heads of the tourism planning task

The tourist's initial location is at `caro_hotel`. The tourist is active between 10:00 and 23:00, defined as a time window using TILs `(at 0 (active tourist))` and `(at 730 (not (active tourist)))`. The tourist wishes to eat between 13:00 and 19:00, specified using `(at 412 (time_for_eat tourist))` and `(at 604 (not (time_for_eat tourist)))`. The opening/closing times of the POIs and the restaurants (also defined as TILs), the recommended durations of POIs visits, and the duration of movement between locations are imported from Open Data platforms and included in I. The predefined list of objects O is shown in Figure B.16. As noted, O contains the tourist and proper nouns objects of the attractions, restaurants and the hotel.

For each $o \in O$ of type $t \in T$, an OWL individual λ is created in η_ϕ of the class t; e.g., `Lonja` is an object of the PDDL type `architecture`, and the system creates lonja as an individual of the class architecture as shown in Figure B.17.

The list of goals G is shown in Figure B.18. The hard goals are at the end of the day, the tourist has eaten and has returned to `caro_hotel`. On the other hand, the tourist wishes to visit eight attractions; these are modelled as soft goals (preferences) that the planner will try to satisfy according to their proprieties within the limited one-day tour.

```
(:objects
    tourist - person
    caro_hotel - hotel
    valencia_cathedral - cathedral
    lonja - architecture
    serrano_towers - tower
    valencia_oceanografic  - aquarium
    valencia_bioparc - zoo
    quart_towers - tower
    viveros_garden - park
    valencia_town_hall - architecture
    ricard_camarena la_cantinella navarro la_salita  el_bolon_verde
    el_pedernil el_celler_del_tossal gordon mood_food blanqueries - restaurant)
```

Figure B.16: PDDL objects of a particular tourism planning problem *Prob*

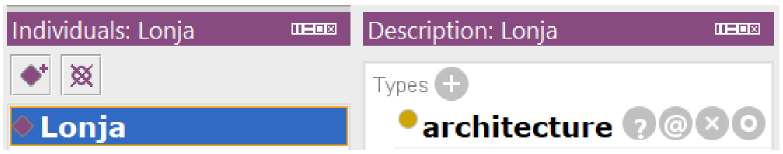


Figure B.17: The ontological representation of the PDDL object `Lonja`

```
(:goals (and (eaten tourist) (be tourist caro_hotel)))
(:constraints (and
    (preference v1 (sometime (visited tourist valencia_cathedral)))
    (preference v2 (sometime (visited tourist lonja)))
    (preference v3 (sometime (visited tourist serrano_towers)))
    (preference v4 (sometime (visited tourist oceanografic)))
    (preference v5 (sometime (visited tourist bioparc)))
    (preference v6 (sometime (visited tourist quart_towers)))
    (preference v7 (sometime (visited tourist viveros_garden)))
    (preference v8 (sometime (visited tourist town_hall)))))
```

Figure B.18: The goals G of a particular tourism planning problem *Prob*

On the other hand, the metric is shown in Figure B.19. The metric consists of the penalties for not visiting the attractions and the penalty for the time spent on moving.

```
(:metric minimize (+
    ; penalties for not visiting the attractions
    ( * ( / (* 280 (is-violated v1)) 1980)  100)
    ( * ( / (* 290 (is-violated v2)) 1980)  100)
    ( * ( / (* 250 (is-violated v3)) 1980)  100)
    ( * ( / (* 300 (is-violated v4)) 1980)  100)
    ( * ( / (* 210 (is-violated v5)) 1980)  100)
    ( * ( / (* 200 (is-violated v6)) 1980)  100)
    ( * ( / (* 250 (is-violated v7)) 1980)  100)
    ( * ( / (* 200 (is-violated v8)) 1980)  100)
    ; penalising the time spent on movement
    ( * ( / (total_moving_time tourist) 730) 100)))
```

Figure B.19: The metric of a particular tourism planning problem $Prob$

The scenario is formulated as a planning task ϕ then a planner is called to solve this task by producing a plan. Figure B.20 shows PLAN1 generated by OPTIC, which includes a total of five attractions visits (`viveros_garden`, `valencia_cathedral`, `lonja`, `quart_towers` and `serrano_towers`), eating at the restaurant (`el_celler_del_tossal`) and returning to the hotel (`caro_-hotel`).

Figure B.21 shows a map demonstrating $PLAN1$. The visits included in $PLAN1$ are marked with red location pins in the snapshot. The tour starts from the original location of the tourist, i.e., `caro_hotel` in which the user is staying (green location pin), and includes visits to the five attractions (red pins) and one stop at a restaurant (orange pin).

While the plan is being executed, the tourist receives a cellphone notification that includes opening a new attraction of a new object type not previously

a_1 : 0.001 : $(move\ tourist\ caro\_hotel\ viveros\_garden)$[11.000]

a_2 : 11.002 : $(visit\ tourist\ viveros\_garden)$[75.000]

a_3 : 86.002 : $(move\ tourist\ viveros\_garden\ valencia\_cathedral)$[11.000]

a_4 : 97.003 : $(visit\ tourist\ valencia\_cathedral)$[120.000]

a_5 : 217.003 : $(move\ tourist\ valencia\_cathedral\ lonja)$[5.000]

a_6 : 222.004 : $(visit\ tourist\ lonja)$[80.000]

a_7 : 302.004 : $(move\ tourist\ lonja\ quart\_towers)$[7.000]

a_8 : 309.005 : $(visit\ tourist\ quart\_towers)$[30.000]

a_9 : 339.005 : $(move\ tourist\ quart\_towers\ el\_celler\_del\_tossal)$[4.000]

a_{10} : 343.000 : $(eat\ tourist\ el\_celler\_del\_tossal)$[60.000]

a_{11} : 403.000 : $(move\ tourist\ el\_celler\_del\_tossal\ serrano\_towers)$[7.000]

a_{12} : 410.001 : $(visit\ tourist\ serrano\_towers)$[80.000]

a_{13} : 490.001 : $(move\ tourist\ serrano\_towers\ caro\_hotel)$[9.000]

Figure B.20: The temporal plan: $PLAN1$

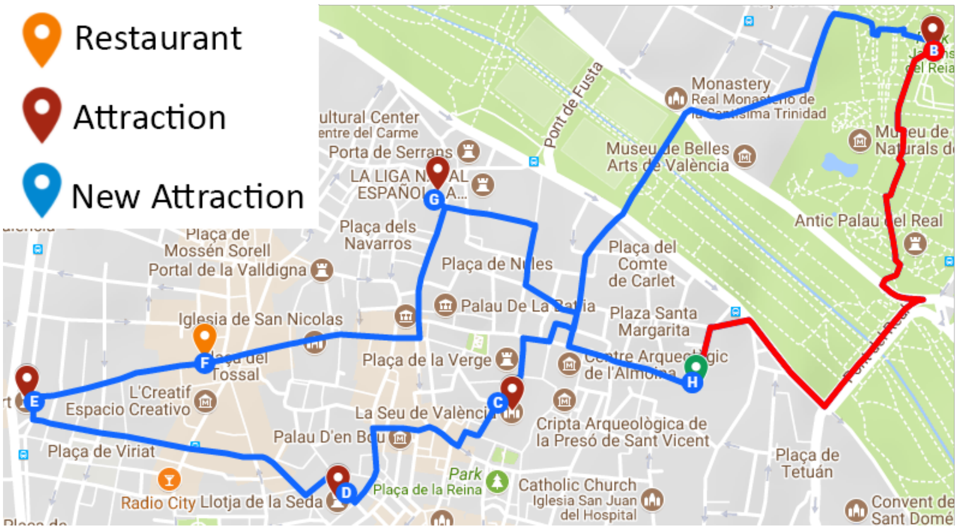


Figure B.21: $PLAN1$: The pins (A/H) Caro hotel, (B) Viveros Garden, (C) Cathedral, (D) Lonja, (E) Quart towers, (F) El Celler del Tossal (restaurant), and (G) Serrano towers.

considered in the planning task. The new objects may present an opportunity to the tourist if the goal can be aligned with the modelling of the planning task and triggers a plan compliant with the current goals resulting in a better tourism experience for the tourist.

B.2.2 Execution Simulation and Context-aware Knowledge Acquisition

The simulation system presented in Chapter 4 is utilised to simulate the execution of $PLAN1$. The simulator starts the execution simulation. After visiting the first attraction `viveros_garden` (a_2 of $PLAN1$ in Figure B.20 and after pin (B) at the end of the red line in Figure B.21), new information is received `(open virgen_plaza)`, which contains a new object `virgen_plaza`.

The system creates the ontological representation of the types of the planning task (explained in Section 5.5.1) Ω_T in η_ϕ, as shown in Figure B.22, respecting the exact hierarchy defined in T.

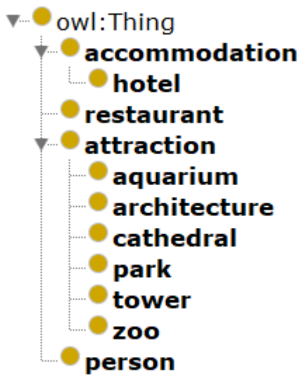


Figure B.22: The OWL representation Ω_T in η_ϕ

The system accesses several remote ontologies available in online-repositories $R^\Delta = \{B, C, D, E, F\}$ (shown in Figure B.23)

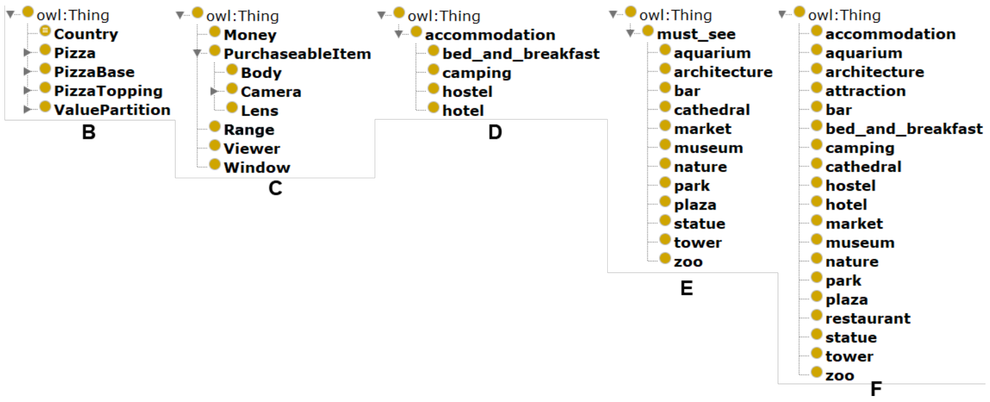
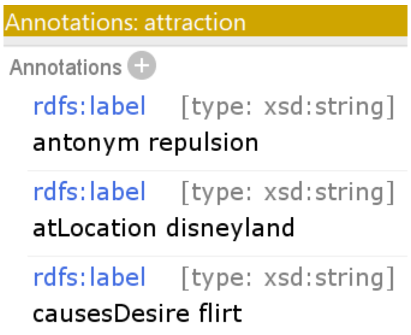


Figure B.23: The remote ontologies $\mathsf{R}^{\Delta} = \{B, C, D, E, F\}$

The ontology η_{ϕ} and the ontologies R^{Δ} are augmented using ConceptNet (explained in Section 5.5.2). Figure B.24 shows a small portion of the twenty-one annotations attached to the class attraction.


Annotations: attraction

Annotations

rdfs:label [type: xsd:string]
antonym repulsion

rdfs:label [type: xsd:string]
atLocation disneyland

rdfs:label [type: xsd:string]
causesDesire flirt

Figure B.24: Annotations sample assigned to Class attraction

Afterwards, the VSM distance (described in Section 5.5.3) is calculated between η_{ϕ} and each ontology in R^{Δ}, the distances are respectively: 0, 0, 0.64, 0.79, and 0.78. Therefore, $\mathsf{R}' = \{D, E, F\}$ is recognised as the set of remote ontologies most similar to η_{ϕ}.

The system tries to find the individual virgen_plaza representing the new object `virgen_plaza` in R' and creates $\mathsf{R}'_\mathsf{o} = \{E, F\}$, the set of ontologies that are most similar to η_ϕ and contain the individual virgen_plaza

The system applies the semantic variance to select the most specialised ontology (explained in Section 5.6). The SV distance is measured to find whether E or F better describes the semantics of the application domain, and the values are 0.25 and 0.16 since E is more hierarchical. Therefore, $\eta_o = E$ is the remote ontology with the highest semantic insight, most similar to η_ϕ, and contains virgen_plaza. Next, the system retrieves the class of the individual virgen_plaza from η_o and finds it to be plaza $\notin T$.

The system attempts to position plaza in η_ϕ by applying an alignment with neighbourhood constraint (explained in Section 5.6.1). By aligning η_ϕ and η_o, the system finds that parent(plaza) in η_o is must_see, and it matches the class attraction in η_ϕ; therefore, the system asserts attraction :: plaza inside η_ϕ. An individual virgen_plaza of the class plaza is created in η_ϕ. Correspondingly, a new entry `plaza - attraction` is added to T, and `virgen_plaza` is added to O. The information required for integrating `virgen_plaza` in ϕ is automatically identified and requested from Open Data platforms; e.g. movement durations between the new location and the existing locations. This information is added to I. Since `plaza` is a sibling of a type that is involved in a goal $g \in G$ thus, the system formulates a new g' =`(visited tourist virgen_plaza)`, and $G' = g' \cup G$. Finally, the system updates I with the current state at the time the new information was received. The planner is called to generate a new plan ($PLAN2$ shown in Figure B.25), allowing the tourist to visit the original POIs plus the new POI `virgen_plaza` of the new type `plaza`.

Figure B.26 shows a map demonstrating $PLAN2$.

a_3 : 86.000 : $(move\ tourist\ viveros\_garden\ virgen\_plaza)$[13.000]

a_4 : 99.001 : $(visit\ tourist\ virgen\_plaza)$[5.000]

a_5 : 104.001 : $(move\ tourist\ virgen\_plaza\ valencia\_cathedral)$[3.000]

a_6 : 107.002 : $(visit\ tourist\ valencia\_cathedral)$[120.000]

a_7 : 227.002 : $(move\ tourist\ valencia\_cathedral\ lonja)$[5.000]

a_8 : 232.003 : $(visit\ tourist\ lonja)$[80.000]

a_9 : 312.003 : $(move\ tourist\ lonja\ quart\_towers)$[7.000]

a_{10} : 319.004 : $(visit\ tourist\ quart\_towers)$[30.000]

a_{11} : 349.004 : $(move\ tourist\ quart\_towers\ el\_celler\_del\_tossal)$[4.000]

a_{12} : 353.000 : $(eat\ tourist\ el\_celler\_del\_tossal)$[90.000]

a_{13} : 443.000 : $(move\ tourist\ el\_celler\_del\_tossal\ serrano\_towers)$[7.000]

a_{14} : 450.001 : $(visit\ tourist\ serrano\_towers)$[80.000]

a_{15} : 530.001 : $(move\ tourist\ serrano\_towers\ caro\_hotel)$[9.000]

Figure B.25: The temporal plan: $PLAN2$.

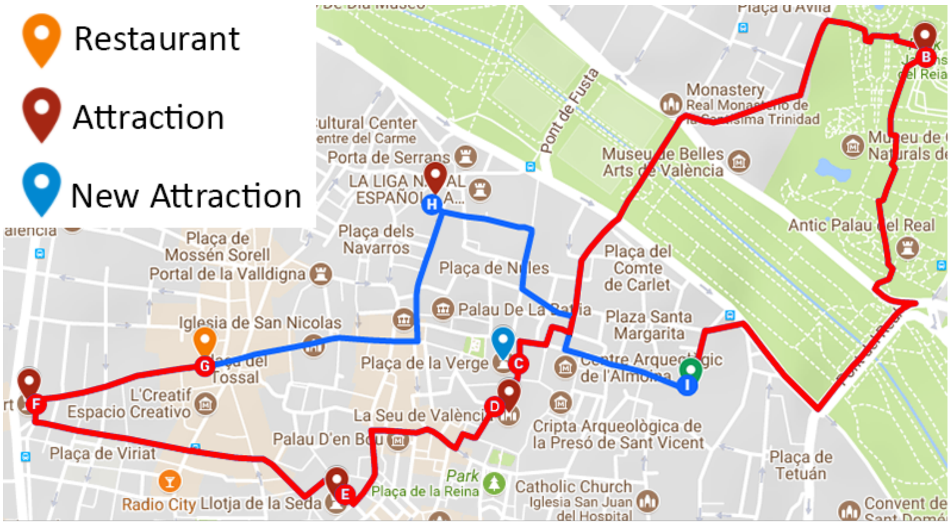


Figure B.26: $PLAN2$: The pins (A/I) Caro hotel, (B) Viveros Garden, (C) Virgen plaza, (D) Cathedral, (E) Lonja, (F) Quart towers, (G) El Celler del Tossal (restaurant), and (H) Serrano towers.

The simulation continues. After the tourist has eaten in the restaurant `el_celler_del_tossal` (after a_{12} of $PLAN2$ in Figure B.25 and after pin (G) at the end of the red line in Figure B.26), new information is received `open jimmy_glass_jazz_bar`. Similarly, the system deals with the new information and extends the knowledge of the planning task, a new plan is obtained ($PLAN3$ shown in Figure B.27) and demonstrated in the map of Figure B.28.

a_{13} : 443.000 : ($move\ tourist\ el\_celler\_del\_tossal\ serrano\_towers$)[7.000]
a_{14} : 450.001 : ($visit\ tourist\ serrano\_towers$)[80.000]
a_{15} : 530.001 : ($move\ tourist\ serrano\_towers\ jimmy\_glass\_jazz\_bar$)[5.000]
a_{16} : 535.002 : ($visit\ tourist\ jimmy\_glass\_jazz\_bar$)[45.000]
a_{17} : 580.002 : ($move\ tourist\ jimmy\_glass\_jazz\_bar\ caro\_hotel$)[9.000]

Figure B.27: The temporal plan: $PLAN3$.

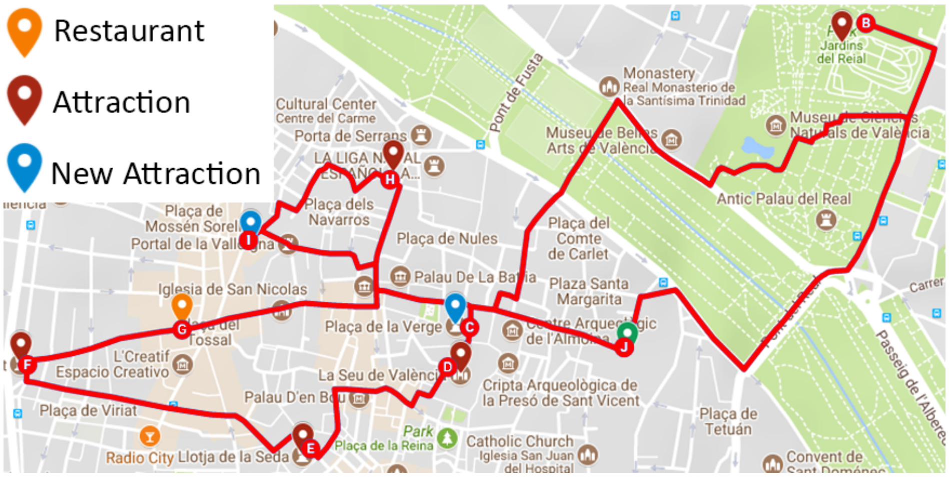


Figure B.28: $PLAN3$: The pins (A/J), B, C, D, E, F, G, H are the same as $PLAN2$ and (I) Jimmy Glass Jazz bar

The execution simulation continues until $PLAN3$ ends. The tourist ends up visiting seven attractions instead of five attractions, eating in the restaurant, and returning to the hotel.

B.3 Conclusion

In this appendix, the author has demonstrated in a step-by-step fashion the behaviour and validity of the context-aware knowledge acquisition approach for planning applications in two different application domains: a repairing agency application domain and a tourist assistant application domain. Drawing upon the richness and expressivity of standard ontologies, semantic measures and operations, the agent can integrate new objects of new types not previously considered in the agent's model into the planning task. Consequently facilitates the anticipation of opportunities.

It can be noted that the tourist assistant application domain, detailed in Section B.2, have generic (non-specialised) actions such as `move`, `visit` and `eat` actions. In addition, the objects in this application domain are proper nouns, such as Caro Hotel and Lonja. Therefore, we did not need to apply the TSM and working on the level of the types was sufficient. On the other hand, applying a TSM similarity measure would have been a better option if the scenario were to include specialised actions requiring specialised conditions. Such as hiking and canoeing actions that require hiking or canoeing types of equipment, as the case with the repairing agency application detailed in Section B.1, in which the agent is equipped to repair a mobile phone, is distinguished from that agent that is not using TSM.

Our approach is domain-independent, only requires the planning task of the agent as input, and does not require prior knowledge of exogenous events or possible opportunities since the agent can extend its knowledge autonomously without being programmed to handle every potential situation. To do so, the system uses remote ontologies or the public planning information shared by semi-cooperative agents, translates this information into ontologies, and applies various semantic operations. Furthermore, the agents would share information only about types, relationships between types, and heads of operations applied to types, not private details such as objects, state variables, state goals, state states, or how operations are performed.

Having an ontological representation of their public planning information would assist agents in communicating, collaborating, and autonomously extending their knowledge without being programmed to handle every possible situation. However, that is a gap between the planning community and the knowledge community, which is being progressively bridged by recent research efforts.

APPENDIX C

Case Study: Assisted Living Homes

DERIVED from the contributions presented in this PhD thesis, namely: the intelligent system for execution simulation in a dynamic environment (presented in Chapter 4) and the context-aware knowledge acquisition approach for planning applications (presented in Chapter 5), and other AI techniques that will be presented in this appendix, this appendix presents an integrated deliberation architecture for Ambient Intelligence (AmI) healthcare application (Babli, Rincon, et al. 2021).

Monitoring wellbeing and stress is one of the problems covered by ambient intelligence, as stress is a significant cause of human illnesses directly affecting our emotional state. The proposed architecture provides a plan for comforting stressed seniors suffering from negative emotions in an assisted living home and executes the plan considering the environment's dynamic nature. Literature was reviewed to identify the convergence between deliberation and ambient intelligence and the latter's latest healthcare trends. A deliberation function was designed to achieve context-aware dynamic human-robot interaction, perception, planning capabilities, reactivity, and context awareness concerning the environment.

C.1 Introduction

Population ageing has become a global phenomenon, bringing challenges and increasing the demand for healthcare solutions. According to the UN (United Nations Department of Economic and Social Affairs 2020), the population aged 80 years or older nearly tripled between 1990 and 2019 and is projected to triple between 2019 and 2050 to reach 426 million. In addition, according to the American Medical Association, stress is the cause of more than 60 per cent of all human illnesses (Rabin 2002).

AmI refers to intelligent software that supports people in their daily lives by sensibly assisting them (Augusto, Callaghan, et al. 2013). AmI has proven helpful for healthcare and senior citizens. It has been used in a broad spectrum of related applications, such as monitoring seniors and alerting caregivers, recognising seniors' activities, Alzheimer detection, managing stress, and empowering people living with the early stages of dementia with autonomy (Bozan et al. 2019; Schmidt et al. 2018; Giménez-Manuel et al. 2022).

The ability to reason and the inclusion of cognitive tasks are increasingly demanded in AmI to deal with stress-related disorders and other health issues. At least one of the following features is highlighted for the successful application of AmI in healthcare: context awareness, learning, reactivity and proactivity, reasoning with dynamic environments and reasoning with time. These five features and human interaction converge toward the intelligent deliberation notion.

Context awareness is the capability of a system to understand the current situation (environment context), track changes, and relate this knowledge to the system to produce proactive reactions (Augusto, Callaghan, et al. 2013). Learning how the senior feels (personal context), planning what the senior needs (Ramoly et al. 2018), and reasoning about time to maintain an orderly schedule of the senior activities are also crucial functionalities (Pollack et al. 2003).

Whether in open or closed well-structured environments such as Assisted Living (AL), autonomously performing tasks and interacting with humans require deliberation capabilities to adapt to circumstances. In this work, we adopt the deliberation definition of Ingrand and Ghallab (Ingrand et al. 2017), which conceptually distinguishes six functions required for successful deliberation: planning, acting, perceiving, monitoring, learning, and goal reasoning.

The convergence between the desired features in AmI and deliberation is natural due to the technologies being used. The combination of learning and perception provides context awareness, and the combination of planning, acting, and monitoring provides reactivity/proactivity to deal with incomplete knowledge and dynamic environments.

Despite promising, current AmI solutions for seniors' healthcare lack a balanced, integral deliberative function. Planning-focused approaches are useful for proactivity and reactivity but fall short of semantic knowledge reasoning and vice versa. On the other hand, methods that focus on cognition, context awareness, learning, activity recognition, and sensors tend to lack planning capabilities (Meditskos et al. 2017; Chen and Nugent 2019; Rocco et al. 2014).

Typically, approaches that emphasise learning and recognising seniors' activities provide real-time responses but fall short of planning, such as in the work of (Miguez et al. 2019), where the automation module is a communication organiser. In other cases, the focus is on sensors, data collection and communication. Such as the work (Wood et al. 2006), which provides simplistic assistance of the form if-then, based on the processing of sensor data, such as, if a senior falls, then the system alerts medics or if the temperature increases, then the system activates the air conditioner. Depending only on sensors can provide imperfect data since assisted living houses' environments are dynamic as other actors, in particular seniors and caretakers, can alter the environment; moreover, sensor readings may be incomplete or even faulty.

This work's scope is on human-centric technologies that require close alignment of the seniors and the AI interacting with the environment to provide context awareness. To tackle the challenges above, this appendix proposes a deliberation architecture for an AmI healthcare application that provides comfort to stressed seniors suffering from negative emotions in an AL environment. The deliberation architecture makes a balanced synthesis of data and knowledge, learning, perception, dynamic goal generation, planning, acting, and monitoring functionalities required to detect, address, and follow up on the senior status evolution.

The rest of the appendix is organised as follows. The related work section reviews the importance of the various components to achieve deliberation in AmI and the most advanced methods for detecting stress and recognising emotions. The concept design section describes our AL environment and its components. We also explain the planning task representing the suggested concept design. We provide the system schema and briefly explain our deliberative function architecture and its five layers in the proposed methodology section. Afterwards, we describe the details of the deliberation function. We demonstrate the behaviour and validity of the approach and the proposed methods in the validation section. Then, we outline areas for improvement and other implications in the discussion section and finally, we conclude.

C.2 Related Work

Context awareness enhances AmI applications for elderly healthcare and stress management in particular (Burns et al. 2011). They enable such systems to recognise seniors, their status, and the surrounding environment, allowing richer reasoning and better-informed decisions and actions. Perception, also known as observing, is a critical component in deliberation (Ingrand et al. 2017). IoT advances and the scarce availability of human assistance have led to AL. Such environments may be equipped with IoT sensors for determining the availability and location of the objects and for sensing anomalies, such as when a senior suffers stress or falls, making them appealing for seniors' healthcare applications.

Planning, acting, monitoring, and reasoning are crucial for deliberation in AmI healthcare systems to handle time as well as the dynamic environment when a plan is executed. The AmI homecare system for diabetic patients (Amigoni et al. 2005) utilises a distributed hierarchical task network to coordinate a plan, monitors its execution and the environment to adjust the room temperature, suggests insulin injection, or alerts the user on the need for medical attention. COACH (Mihailidis et al. 2004) relies on planning and computer vision to support ageing-in-place safety. The framework (Ramoly et al. 2018) for domestic healthcare robots generates a goal dynamically according to a cognition layer and uses a task planner to plan actions. The generalised argumentation framework (Oguego et al. 2018) takes adaptive decisions and reasons with the evolution of inhabitants' preferences over time on lighting, healthy eating, and leisure inside a smart home. The framework in (Augusto and Nugent 2004) utilises temporal reasoning with a rule-based system to recognise hazardous situations in a smart home environment, facilitating decision-making that resolves the anomaly and returns the environment to a safe state. PEAT (Levinson 1997) and Autominder (Pollack et al. 2003) both utilise automated planning to provide daily activities reminders to adjust schedules due to changes in the observed activities.

Ontologies are helpful informally defining the terms and relationships representing the vocabulary of a domain and its context, allowing the reuse of richer, fine-grained knowledge to assist in the performance of everyday tasks and promoting context awareness toward everyday environments (Babli and Onaindia 2019; Daoutis et al. 2009; Nguyen, Loke, et al. 2011).

Machine learning has recently been popular for context recognition, recognising human activities and emotions and observing environment objects (Howard et al. 2017). A variation in stress levels directly affects the emotional state and negatively influences physical and mental health. A recent study (Bozan et al. 2019) shows that negative emotions are the top problem-related category for AL. Stress is detected through a variety of bio-signals such as electroencephalography (EEG), photoplethysmography (PPG) and electrocardiogram (ECG), and facial expressions analysis, which is contactless, allowing detecting a larger spectrum of emotions (Gradl et al. 2019).

C.3 Concept Design of the AL Environment

Helping clients understand self-care is vital in relapse prevention and recovery facilities, AL or nursing homes. Emotions such as terror, sadness, and boredom can also occur in response to trauma. The recovery process requires acknowledging these emotions as they could be signs of stress disorders (Yehuda 2002). If taken into consideration, simple self-care indicators could prevent accidents (Abbas et al. 2017) or improve fall prevention plans by paying attention to hydration (Benelam et al. 2010). To this extent, we aim to design a plan to provide hydration, cheering, or entertainment when a stressed senior feels terrified, sad, or bored. The plan is then executed, considering the dynamic nature of the AmI environment.

Figure C.1 shows the AL environment's concept design, whereas the planning task's description is explained in the next section.

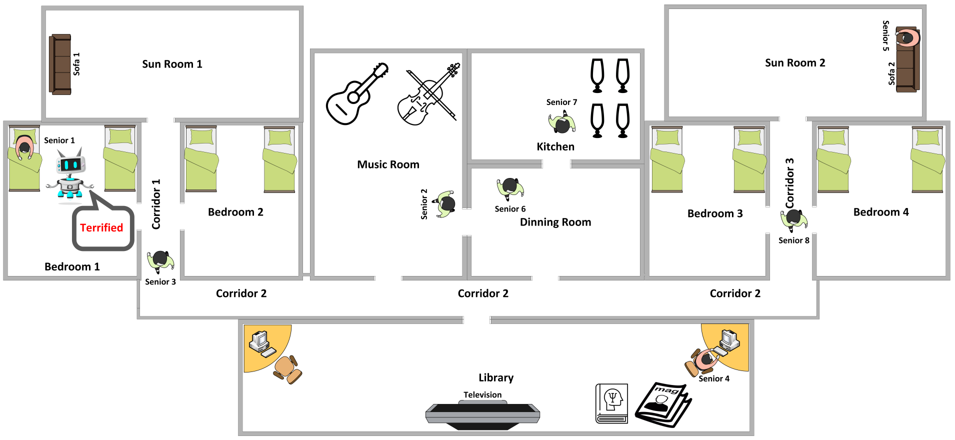


Figure C.1: Concept design of an AL home using the system

The environment consists of four bedrooms, two sunrooms, a music room, a dining room, a kitchen, a library, three corridors linking the above spaces, a robot, and several senior citizens wearing IoT wristbands. The environment features a set of IoT objects of different categories (types) distributed in the AL home. Drinking glasses fall under the category of drinking vessels in the kitchen. A violin and a guitar fall under the category of musical instruments in the music room. Entertainment objects in the library are to be given, such as magazines and books, or to be suggested for use, such as PCs and TVs. There are two types of actions by the robot in the environment: doing an activity (moving between rooms, carrying an object, giving an object, filling a drinking vessel) and suggesting an activity. Due to the environment's dynamic nature, existing IoT objects could become unavailable (being unexpectedly used by other seniors or due to a malfunction). Additionally, new non-IoT objects that have no connection to the grid and were not previously introduced in the environment may also be encountered (added by the caretakers) during the plan's execution and could be used to repair failures.

C.4 Planning Task of the AL Concept Design

We represent the AL environment's concept design as a temporal planning task using version 2.2 of the popular planning domain definition language (PDDL). Temporal planning involves selecting and organising actions to satisfy the goals and assigning a start time and duration to each action. The planning domain and the planning problem describe a temporal planning task. The planning problem varies depending on the particular problem to be solved. In contrast, the planning domain is a fixed description of what the planner can do to achieve a set of goal conditions starting from an initial state. The planning domain characterises the planning task behaviour, comprises the model (shown in Figure C.2) of the AL environment, and consists of the following:

1. Set of object types, such as `robot`, `senior`, `location`, `drinking_vessel`, `making_music` and `entertainment`.
2. Set of variables such as `(path ?loc1 ?loc2 - location)` to indicate that two locations are linked and `(moving_time ?loc1 ?loc2 - location)`, representing the moving time between two locations.

```
(:types
robot senior location drinking_vessel making_music - object
entertainment - object
solving reading communication_device - entertainment
drinking_glass glass_bottle - drinking_vessel
violin guitar - making_music
puzzle yoyo - solving
magazine book  - reading
tv desktop - communication_device
dinning_room kitchen music_room library bedroom - location
corridor sunroom - location)
```

```
(:predicates
;; locatable is at a location
(be ?locatable (either robot senior drinking_vessel
    making_music entertainment) ?loc - location)
;; a link exists between two location
(path ?loc1 - location ?loc2 - location)
;; time-window for soothing terrified
(hydration_time ?senior - senior)
;; the senior has fear (terrified)
(terrified ?senior - senior)
;; the senior is sad
(sad ?senior - senior)
;; the senior is bored
(bored ?senior - senior)
;; fear is soothed
(soothed ?senior - senior)
;; sadness is soothed and the person is cheerful
(cheerful ?senior - senior)
;; boredom is soothed and the person is entertained
(entertained ?senior - senior)
;; availability of an object
(available ?object - (either drinking_vessel
    making_music entertainment))
;; status of a drinking vessel
(filled ?item - drinking_vessel)
;; the robot is carrying an object
(holding ?robot - robot ?item - (either drinking_vessel
    making_music solving reading))
;; an object has already been given to a senior
(given ?item - (either drinking_vessel making_music
     solving reading) ?senior - senior)
;; a suggestion has already been given to a senior
(suggested ?item - (either tv desktop) ?senior - senior)
```

```
(:durative-actions heads
;; movement of robot between two connected locations
(move (?robot - robot ?loc1 - location ?loc2 - location))
;; the robot carries an item from a location
(carry (?robot - robot ?item - (either drinking_vessel
     making_music solving reading) ?location - location))
;; the robot fills up a drinking vessel in the kitchen
(fill (?robot - robot ?item - drinking_vessel
     ?location - kitchen))
;; the robot gives a senior an item
(give (?robot - robot ?senior - senior ?item - (either
     drinking_vessel making_music solving reading)
     ?location - location))
;; senior received soothing for fear state
(soothed_received (?senior - senior
     ?item - drinking_vessel))
;; the senior received the cheering up to soothe sadness
(cheer_received (?senior - senior ?item - making_music))
;; the senior received the entertainment to soothe boredom
(entertainment1_received (?senior - senior ?item - (either
     solving reading)))
;; the robot suggest a different course of entertainment
(suggest_entertainment2 (?robot - robot ?senior - senior
     ?item - (either tv desktop) ?location - location))
;; the senior has received the entertainment suggestion
(entertainment2_suggested senior (?senior - senior
     ?item - (either tv desktop)))
```

Figure C.2: T, V and OP heads of the AL environment domain Dom

3. Action schemas representing operations that can be performed, e.g., `(move ?robot - robot ?loc1 ?loc2 - location)...`, which allows the robot to move between locations

The planning domain is designed to penalise the actions of suggestion (action costs) in the metric to favour actions that include providing an object to seniors rather than merely suggesting an activity. Suggesting an activity is left as a last resort when there are no available objects to comfort the senior.

On the other hand, the planning problem represents a particular problem instance to be solved, specified by the initial state and the goal. The information of the basic planning problem includes three corridors; `corridor1...3`, four bedrooms; `bedroom1...4`, one kitchen `kitchen1`, one music room `music_room1`, one library `library1`, one dining room `dining_room1`, and two sunrooms; `sunroom1` and `sunroom2`. The paths between locations (the layout), e.g., `(path sunroom1 corridor1)`. Eight seniors; `senior1...8`. One robot `robot1`. Eight glasses; `glass_id01...08`. One violin `violin_id01` and one guitar `guitar_id01`. Two desktops, `pc_id01` and `pc_id02`, and one television, `tv_id01`. One book `book_id01` and one magazine `magazine_id01`. The information of the planning problem specifies the initial state; that is, the locations of the environment objects, the seniors and the robot, and the availability of objects will be sensed and filled from IoT objects. The goal condition will be dynamically generated, as explained in the following section.

C.5 Proposed methodology

To achieve our aim, we designed the deliberation function schema (sketched in Figure C.3). The function consists of five layers. The offline layer (layer 0 learning and knowledge augmentation) consists of four modules: learning stress, learning emotions, types augmentation, and learning types. On the other hand, the online layers are the perception layer (layer 1), the goal generation layer (layer 2), the planning layer (layer 3), and the acting and monitoring layer (layer 4).

Figure C.3 in Roman numerals shows the input to our system. The input consists of the following: (I) a multi-modal database for stress classification; (II) a dataset with 4,900 images of human facial expressions; (III) the basic planning domain model follows our AL environment concept design and a

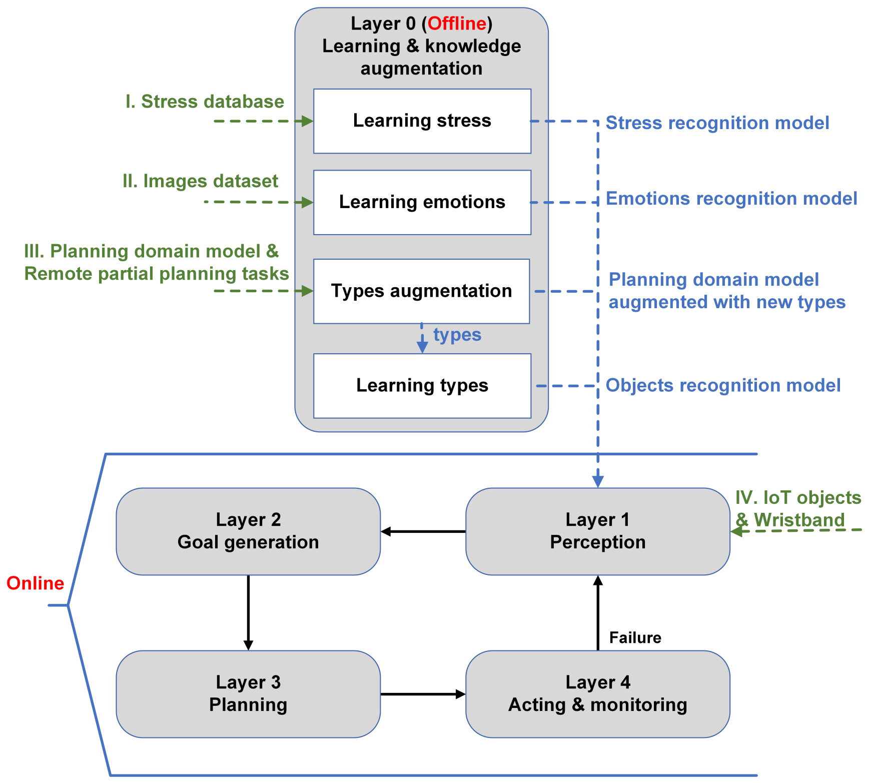


Figure C.3: Schematic view of the deliberation function system architecture.

repository of several semi-cooperative agents' remote partial planning tasks; (IV) the data related to the IoT objects (e.g., glasses, violins, guitars, magazines, books, televisions, and computers) and IoT wristbands. Details about the input are found in the system details section.

For context-aware human-robot interaction with seniors, the system can determine online whether seniors wearing an IoT wristband suffer from stress. Then the system recognises the stressed senior negative emotion and dynamically generates a goal that corresponds to the negative emotion. This is achieved by the perception layer and the goal generation layer shown in Figure C.3. The system utilises the learning stress and emotions modules to train the system offline and prepare to perform them online.

The system utilises perception, planning, and acting and monitoring layers, shown in Figure C.3, to achieve perceptiveness, planning capabilities, and reactivity when handling failures and context awareness toward the environment online. The system uses types augmentation and learning types modules to train it offline to be prepared to perform these tasks online.

The deliberation layers depicted in Figure C.3 are as follow:

1. Layer 0: learning and knowledge augmentation, executed offline to produce three classification models for recognising stress, emotions, and objects. Layer 0 consists of four modules:

   (a) Learning stress: processes the multi-modal database for stress and trains the system to prepare it to detect stress by providing the stress classifier model.

   (b) Learning emotions: processes the dataset of images of facial expressions and trains the system to prepare it to recognise emotions by providing the emotions classifier model.

   (c) Types augmentation: is a cognitive semantic module that dynamically augments the predefined list of object types of the planning task with relevant new object types; to be context-aware toward the environment and the task being performed and reason with incomplete knowledge, thereby boosting the system's autonomy and context awareness.

   (d) Learning types: trains the system on the augmented set of object types to prepare it to recognise objects of the relevant types by providing the objects classifier model.

2. Layer 1: perception provides our system with the critical capability of observing by:

   (a) Reading the seniors' bio-signals using the IoT wristbands and sending these signals to the service in charge of the analysis using the stress classifier for stress detection; then recognising the negative emotion of the stressed senior using the emotions classifier model.

   (b) Sensing the IoT objects' values characterising the environment's actual state, e.g., the seniors' locations, the robot's location, and the objects' availability.

   (c) If a failure occurs, perception provides reactivity. During the plan's execution, the robot may encounter new non-IoT objects of the relevant types not predefined in the model. The robot uses the objects classifier to recognise such objects dynamically, integrates them into the planning problem, and uses them to repair failures.

3. Layer 2: goal generation is responsible for dynamically generating the goal associated with resolving the recognised negative emotion.

4. Layer 3: planning is responsible for:

   (a) Formulating the planning problem.

   (b) Reformulating the planning problem (in case of failure) starting from the state where the failure occurred, considering the new observed state and the newly integrated objects.

   (c) Calling an external planner to synthesise a plan of actions to be executed by the robot to achieve the goal and comfort the senior suffering from negative emotions.

5. Layer 4: acting and monitoring are responsible for executing the plan, monitoring action, and detecting failures that may occur when an IoT object used in the plan becomes unavailable.

The logic behind recognising the emotion in addition to sensing stress for goal generation is that the IoT wristband is prone to noise, thus, affecting the stress classification process, as shown in the validation section. The technical details of the deliberation function are shown in the next section.

C.6 System details

This section details the proposed deliberation function. It describes the wristband hardware used in the perception layer, stress learning and detection, emotions learning and recognition, semantic knowledge augmentation, object types learning, planning, acting, and monitoring.

C.6.1 Stress Detection and Emotions Recognition

The aim is to train the system to prepare it to detect stress and recognise negative emotions to generate the goal corresponding to comforting the negative emotion dynamically.

The wristband we used has a PPG sensor, and it was designed using an ESP32 Oled Lora TTGO LoRa32 development board (Figure C.4). The TTGO is responsible for acquiring the PPG sensor signal. The signal is processed and sent through the Wi-Fi module to the web service that was programmed using Flask[1] and is in charge of signals analysis and detection of stress levels.

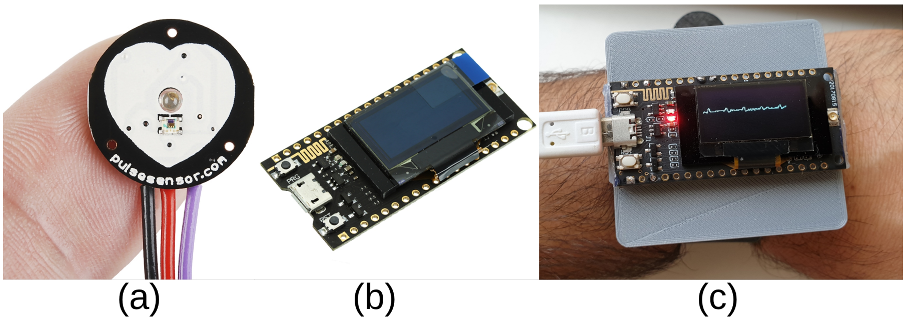


Figure C.4: The wristband hardware. (a) Cardiac pulse sensor, (b) ESP-32 developer board and (c) ESP-32 developer board.

Since our system only incorporates a single PPG sensor, and it is possible to detect stress levels using the heart rate (HR) and the interval between the PPG signal peaks (less intrusive than ECG), a new database was created from the multi-modal WESAD dataset [2] (Schmidt et al. 2018). Figure C.5 shows two signals (ECG, PPG) captured by our system to show the relationship between the cardiac and the PPG signals. This new database has only two inputs and one output (stressed or unstressed). The new database was divided into two parts to carry out the training process: 80% for training and 20% for testing. A k-fold cross-validation statistical method (Ling et al. 2019) was used to independently determine the partitions between training and test data and estimate the learning model's ability. The best results were obtained with a *k=4*. The obtained results obtained are shown in Section C.7.2.

Our system incorporates a convolutional neural network (CNN) of three convolutional layers [3, 32, 32] to classify human emotions. To perform the training

[1] Flask https://flask.palletsprojects.com/en/1.1.x/
[2] Stress classifier with AutoML https://github.com/chriotte/wearable_stress_classification

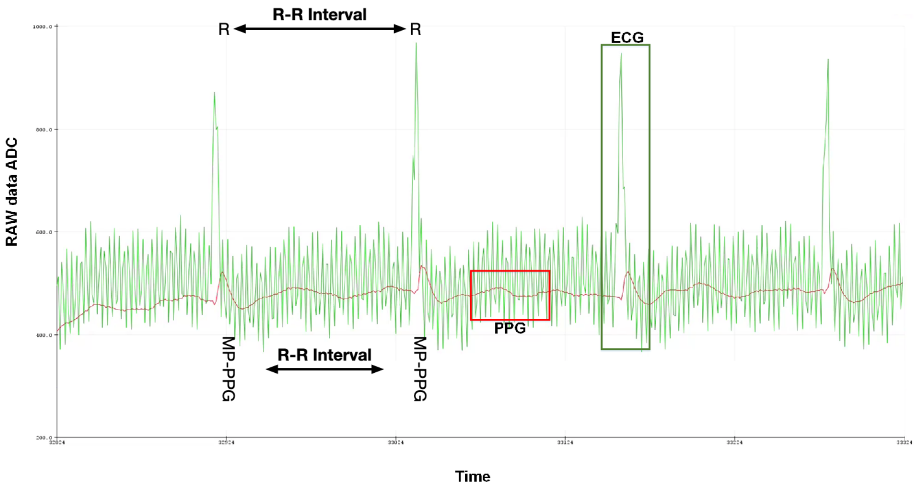


Figure C.5: The relationship between ECG and PPG signals.

and the test, we used the Karolinska Directed Emotional Faces (KDEF) database (Calvo et al. 2018). KDEF has 4,900 images of facial expressions of 70 actors (35 men and 35 women) showing seven different emotions: terror, anger, disgust, happiness, neutral, sadness, and surprise. Since the KDEF database does not include the boredom emotion, a new class has been created using the UMB-DB database (Colombo et al. 2011). The new dataset was divided into two parts: 80% of the images were used for training, and 20% were used for testing. The images were re-sized to 48×48 pixels. The CNN hyper-parameters are as follows: 0.01 for the l2 regularisation or l2-penalty, [32, 64, 128, 64, 32] as hidden layers, 0.2 for the dropout rate, val_loss for the monitor, and 10 for the min delta. The obtained results are shown in Section C.7.3.

Two examples of the emotions recognised by the system are shown in Figure C.6 and C.7 for detecting the emotions of terrified and bored, respectively. Once the emotion of a stressed senior is recognised, the system dynamically generates the goal corresponding to the emotion (`soothed senior`) if terrified, `(cheerful senior)` if sad, and `(entertained senior)` if bored. The variable `(soothed senior)` represents the goal that needs to be accomplished by the plan computed by the planning solver to set the variable `(soothed senior)` to true.

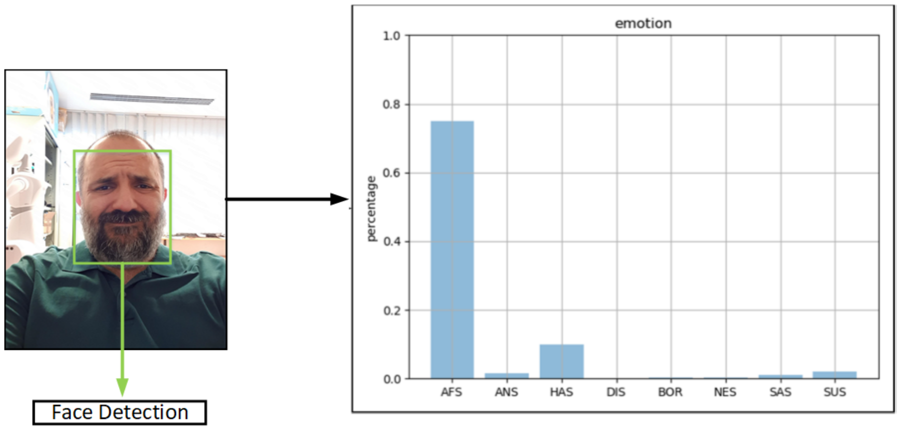


Figure C.6: An example of the emotions recognised by the system: terrified

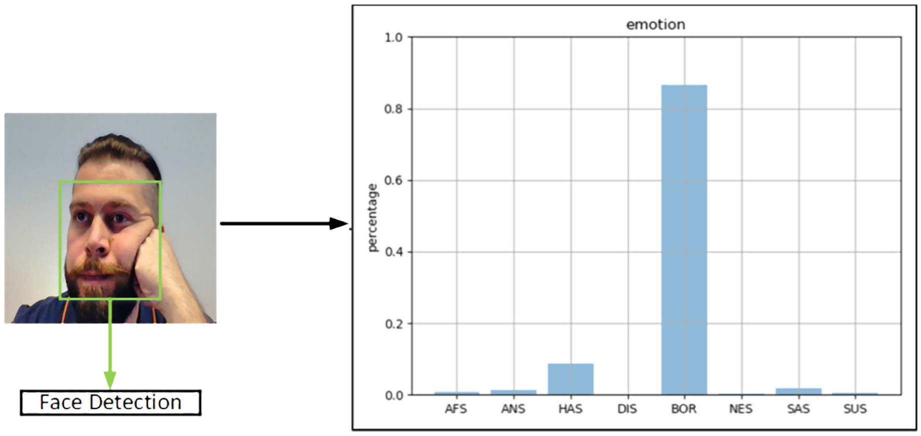


Figure C.7: An example of the emotions recognised by the system: bored

C.6.2 Semantic Knowledge Augmentation and Types Learning

For the deliberation function to recognise new relevant objects and be context-aware of the execution environment and the task it performs, it needs to extend its knowledge dynamically, then it needs to be trained to recognise objects of relevant types, so it can observe relevant objects when a failure occurs that could be used to repair the failure. This layer aims to provide the agent context awareness of the execution environment and boost the agent's autonomy for repairing failures. This is achieved by extending the agent's knowledge and correspondingly its planning domain, with types similar to T and also operable by the agent capabilities, i.e. using the agent's existing operators OP.

It might be tempting to think that a solution to suggest alternatives to types of objects in a domain model is a mere traversal of hypernyms (more general terms) and hyponyms (more specific terms) hierarchies from online linguistic resources or lexical databases. However, this solution has two limitations. The first limitation is that the newly imported types would require an evaluation from a human domain modeller or an expert for each application domain, degrading the agent's autonomy. The second limitation, there is no guarantee that the newly suggested types can be handled by the agent capabilities (compatible with the agent's operators). For that purpose, we adapted the context-aware knowledge acquisition method using the ontologies approach presented in (Babli and Onaindia 2019) and explained in Chapter 5. This chapter has abstracted out the details of the ontological representation, the preliminary similarity measure (VSM), and the tailored similarity measure for planning dynamics (TSM). We refer the reader to Chapter 5 for their detailed description.

In the work of (Babli, Rincon, et al. 2021), the author has adapted the context-aware knowledge acquisition approach presented in (Babli and Onaindia 2019).

The adaptation was required as the original system receives as input a new object and requests its type either from the agent that delivered it or by finding the class to which the individual (representing the object) belongs from a remote ontology, which is not available in the case of the AL environment domain. Instead, in the AL environment domain, we have a robot that moves around, and we want to teach it to recognise the object's relevant types. Therefore, the agent:

1. Creates a preliminary OWL ontological representation called η_ϕ of only the types T of ϕ.
2. Retrieves a set of partial remote planning tasks Δ, specifically the types of several semi-cooperative agents from online repositories, and creates their OWL ontological representation R^Δ.
3. Augments the types classes of η_ϕ and R^Δ using ConceptNet.
4. Applies a quick vector space distance (VSM) similarity measure to R^Δ and obtain the set R', that contains ontologies most similar to η_ϕ according to types.

As explained in Section 5.4 of Chapter 5, when we do not know the type of the new object upfront, we prepare the agent to be capable of recognising the objects' types that are relevant and manageable. Thereby comes the main difference: for each $t_i \in T$ (each type within the agent's predefined list of types), the agent will attempt to find a specified number of relevant types. Therefore, for each $t_i \in T$ the agent:

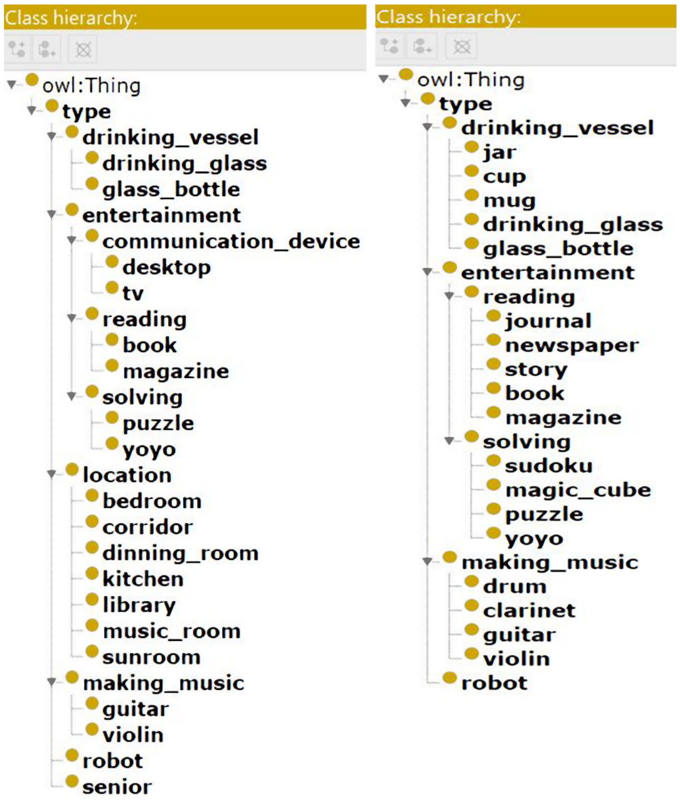
1. Filters out the remote ontologies that do not contain t_i to obtain R'_{t_i}.
2. Extends η_ϕ and R'_{t_i} to represent not only types T but also variables V and the heads part of actions schemas OP.

3. Augments the variables and operators classes of η_ϕ and R'_{t_i} using ConceptNet.

4. Applies a tailored semantic similarity (TSM) measure between η_ϕ and ontologies in R'_t (to take into account the planning dynamics of V and OP and obtain N_ϕ. An ontology in N_ϕ with a high similarity value means that the remote agent is equipped with similar capabilities.

Since these steps above are going to be applied for each type $t_i \in T$, the agent is going to end up with multiple sets of remote ontologies $\{N^\phi_{t_1} \dots N^\phi_{t_i} \dots N^\phi_{t_n}\}$, as explained in the offline usage of Phase 2: Thorough identification of similar ontologies in Section 5.4 of Chapter 5. These ontologies represent the ontologies of the remote agents equipped with similar capabilities to the agent with the planning task ϕ and contain t_i and other classes similar to t_i (t_i siblings). We are interested in these sibling classes because they represent prospect types. The agent chooses the most similar ontology from each set in $\{N^\phi_{t_1} \dots N^\phi_{t_i} \dots N^\phi_{t_n}\}$ and attempts an alignment of a specified number of classes from these ontologies with η_ϕ. The agent aggregates a specified number of types classes (which has a successful alignment) and generates the list of augmented types, which the agent will be trained to recognise (offline and only once). Figure C.8 shows the output for our healthcare application domain.

The left side of Figure C.8 shows the original hierarchy of types of η_ϕ, whereas the right side of Figure C.8 shows the extended hierarchy of types T' useful for repairing failures. Note the new types; `jar`, `cup`, and `mug` under the superclass `drinking_vessel`; `journal`, `newspaper`, and `story` under the superclass `reading`; `sudoku` and `magic_cube` under the superclass `solving`; and `drum` and `clarinet` under the superclass `making_music`. The system will train the agent on the extended set of types T' (as shown in the following section) to achieve smart environment monitoring during plan execution.



Class hierarchy:

```
owl:Thing
  type
    drinking_vessel
      drinking_glass
      glass_bottle
    entertainment
      communication_device
        desktop
        tv
      reading
        book
        magazine
      solving
        puzzle
        yoyo
    location
      bedroom
      corridor
      dinning_room
      kitchen
      library
      music_room
      sunroom
    making_music
      guitar
      violin
    robot
    senior
```

Class hierarchy:

```
owl:Thing
  type
    drinking_vessel
      jar
      cup
      mug
      drinking_glass
      glass_bottle
    entertainment
      reading
        journal
        newspaper
        story
        book
        magazine
      solving
        sudoku
        magic_cube
        puzzle
        yoyo
    making_music
      drum
      clarinet
      guitar
      violin
    robot
```

Figure C.8: OWL Representation of the hierarchy of the predefined types and the augmented types

C.6.3 Learning objects

The training for the recognition of objects is performed using the sub-classes located within the ontology that represents the extended knowledge T' that the system possesses. Learning is done dynamically and only once per planning task.

The system extracts the subclasses' names of the augmented set of types. It uses Google Images Download[3], a command-line Python program to download 200 images from Google of each subclass. We have developed a CNN composed of three convolutional layers for object recognition [3, 32, 32]. The best results were obtained using three convolutional filters for each layer. The filter for each layer has the architecture [32, 32, 32, 32]. Finally, the fall rate was 0.2. Once the training is complete, the perception layer will be ready to observe new objects of the relevant types when the plan is executed. For example, later, during execution, the system can autonomously recognise the new object `mug_id01` of the type `mug`, incorporate its related information, and integrate it within the planning task. The results obtained are shown in Section C.7.4.

C.6.4 Planning, Acting, Failure Detection, and Monitoring

The planner provides a personalised plan that must reflect seniors' satisfaction with the AL home. To build the plan, the planner needs to consider the activities' durations and the IoT objects' availability. Solving this problem requires using a planning system capable of dealing with durative actions to represent the duration of actions and hard goals according to seniors' emotions. For the implementation, among the few automated planners capable of handling temporal planning problems with time windows, we opted for the planner OPTIC because it handles PDDL 2.2.

The values of the planner's variables are read using the perception layer and compiled into a planning problem. The planner is called to generate plan π_1 (shown in Figure C.9). π_1 consists of twelve actions for the robot `robot1` to move from `bedroom1` to `kitchen1`, carry and fill `glass_id05`, and then return to comfort terrified senior `senior1`.

[3]Google Images Download® https://google-images-download.readthedocs.io/en/latest/index.html

a_1 : 0.000 : $(move\ robot1\ bedroom1\ corridor1)$[30.000]
a_2 : 30.002 : $(move\ robot1\ corridor1\ corridor2)$[50.000]
a_3 : 80.004 : $(move\ robot1\ corridor2\ dinning\_room1)$[40.000]
a_4 : 120.006 : $(move\ robot1\ dinning\_room1\ kitchen1)$[20.000]
a_5 : 140.008 : $(carry\ robot1\ glass\_id05\ kitchen1)$[10.000]
a_6 : 150.010 : $(fill\ robot1\ glass\_id05\ kitchen1)$[10.000]
a_7 : 160.012 : $(move\ robot1\ kitchen1\ dinning\_room1)$[20.000]
a_8 : 180.014 : $(move\ robot1\ dinning\_room1\ corridor2)$[40.000]
a_9 : 220.016 : $(move\ robot1\ corridor2\ corridor1)$[50.000]
a_{10} : 270.018 : $(move\ robot1\ corridor1\ bedroom1)$[30.000]
a_{11} : 300.020 : $(give\ robot1\ senior1\ glass\_id05\ bedroom1)$[10.000]
a_{12} : 310.022 : $(soothed\_received\ senior1\ glass\_id05)$[30.000]

Figure C.9: The temporal plan π_1. For the robot to go to `kitchen1`, carry and fill `glass_-id05`, go back to `senior1`'s location, `bedroom1`, and provide the required comforting

To simulate actions execution monitoring while the robot is executing plan π_1, our simulator transforms π_1 and the planning task into an infrastructure called the timeline, similar to the work of (Babli, Ibáñez-Ruiz, et al. 2016) explained in Chapter 4. However, the timeline is only a collection of chronologically ordered conditions that must be satisfied in the observed world-state. The simulation starts at $time = 0$, with the simulator's internal state equal to the initial state. Next, the idea is that the simulator advances through time in every execution step. Every execution step at instance $time_i$ involves:

1. Moving through time $time_i = time_i + stepsize$.
2. Using the perception layer to read the observed state of the world.
3. Checking if a failure is detected: a condition in the timeline is not satisfied in the observed state.

- The robot checks if it can recognise new objects of the augmented set of types and integrates them into the planning task along with their corresponding information.
- Goal generation is re-invoked to ensure the goal is still active.
- The planning layer is re-invoked; the planning problem is reformulated with the new initial state set to the observed state, the newly integrated objects' information, and the goal. The planner is called to generate a new plan to achieve the goal. The acting layer is re-invoked.

4. If no failure is detected, the simulation continues from step 1 until all the conditions are monitored.

C.7 Validation

This section aims to demonstrate the validity of the following: (1) the behaviour of our system using three simulated scenarios for fixing failures; (2) the proposed method for stress detection; (3) the proposed method for emotions recognition; and (4) the proposed method for classifying new objects.

C.7.1 Validity of Behaviour of the System

For simplicity, we show the behaviour of the system for one senior. The failure is repaired using the observed new objects in the first and second scenarios. In contrast, it is fixed by suggesting activity in the third scenario.

Scenario 1. Considering the healthcare scenario described in the concept design and the planning task sections. When perception detects a stressed senior, the robot recognises the senior's emotion, as explained in the stress detection and emotions recognition section, which is terrified in this case. The system dynamically generates a new goal to comfort the terrified senior (`soothed senior1`) and adds the new goal to the planning problem.

Perception reads the origin places from the IoT environment objects (shown in Figure C.1) and adds them to the planning problem: the robot is at bedroom1 (`be robot1 bedroom1`). The origin locations of each senior, e.g., (`be senior1 bedroom1`). The origin locations for glasses in the kitchen, e.g., (`be glass_id05 kitchen1`). The origin locations of instruments in the music room (`be violin_id01 music_room1`) and (`be guitar_id01 music_room1`). The origin locations of the entertainment objects in the library (`be book_id01 library1`) and (`be magazine_id01 library1`). The origin locations of communication devices (PCs and TV) in the library, (`be pc_id01 library1`), (`be pc_id02 library1`), and (`be tv_id01 library1`). The availability of objects, (`available glass_id05`), (`available glass_id06`), (`available glass_id07`), and (`available glass_id08`). In addition to actions' durations, such as the duration of movements between the locations, there are carrying and giving an item, filling drinking vessels, and suggesting an activity.

The planner is called to generate plan π_1 (Figure C.9) consisting of 12 actions to comfort the terrified senior by providing the glass `glass_id05` filled with water. The system transforms π_1 to a timeline of events (chronologically ordered conditions). π_1 execution starts, the robot executes the first four actions correctly (a_1, a_2, a_3 and a_4), an exogenous event occurs, and all drinking vessels become unavailable. When the observed state is read at time 140.008, the system detects a violated condition (`available glass_id05`) required by a_5. Subsequently, a failure is detected.

The robot executes perception to read the IoT objects new values and recognise new objects. A new object of type mug is recognised. The object is named `mug_id01` and integrated into the planning problem along with its location (`be mug_id01 kitchen1`) and availability (`available mug_id01`). The goal generation layer indicates that the senior is still terrified (the goal is still active). The planning problem is reformulated with the new observed state as the new initial state and the information of the new object `mug_id01` and the goal (`soothed senior1`). The planner is called to fix the failure, and a new plan π'_1 is generated (Figure C.10). The new object `mug_id01` is carried and filled, and then the robot moves to provide the terrified senior with the filled mug.

$a_1 : 140.008 : (\mathit{carry\ robot1\ mug\_id01\ kitchen1})[10.000]$

$a_2 : 150.010 : (\mathit{fill\ robot1\ mug\_id01\ kitchen1})[10.000]$

$a_3 : 160.012 : (\mathit{move\ robot1\ kitchen1\ dinning\_room1})[20.000]$

$a_4 : 180.014 : (\mathit{move\ robot1\ dinning\_room1\ corridor2})[40.000]$

$a_5 : 220.016 : (\mathit{move\ robot1\ corridor2\ corridor1})[50.000]$

$a_6 : 270.018 : (\mathit{move\ robot1\ corridor1\ bedroom1})[30.000]$

$a_7 : 300.020 : (\mathit{give\ robot1\ senior1\ mug\_id01\ bedroom1})[10.000]$

$a_8 : 310.022 : (\mathit{soothed\_received\ senior1\ mug\_id01})[30.000]$

Figure C.10: The temporal plan π'_1. For the robot to carry and fill `mug_id01`, go back to `senior1`'s location and provide the required comforting

Scenario 2. In this scenario, plan π_2 (shown in Figure C.11) was generated to cheer up a sad senior, `senior1`, located in `bedroom1`. The robot moves from the senior's location to `music_room`, carries `guitar_id01`, and then moves back to cheer up the sad senior by giving `guitar_id01`.

The robot executes the first three actions correctly (a_1, a_2, and a_3), an exogenous event occurs, and all musical instruments become unavailable. When the observed state is read at 120.006, the system detects a violated condition (`available guitar_id01`) required by action a_4 in π_2. Subsequently, a failure is detected due to the discrepancy. Perception recognises a new object of

a_1 : 0.000 : $(move\ robot1\ bedroom1\ corridor1)[30.000]$

a_2 : 30.002 : $(move\ robot1\ corridor1\ corridor2)[50.000]$

a_3 : 80.004 : $(move\ robot1\ corridor2\ music\_room1)[40.000]$

a_4 : 120.006 : $(carry\ robot1\ guitar\_id01\ music\_room1)[10.000]$

a_5 : 130.008 : $(move\ robot1\ music\_room1\ corridor2)[40.000]$

a_6 : 170.010 : $(move\ robot1\ corridor2\ corridor1)[50.000]$

a_7 : 220.012 : $(move\ robot1\ corridor1\ bedroom1)[30.000]$

a_8 : 250.014 : $(give\ robot1\ senior1\ guitar\_id01\ bedroom1)[10.000]$

a_9 : 260.016 : $(cheer\_received\ senior1\ guitar\_id01)[30.000]$

Figure C.11: The temporal plan π_2. For the robot to go to `music_room1`, carry `guitar_-id01`, go back to `senior1`'s location `bedroom1` and provide the `guitar_id01` to cheer up the sad senior

type clarinet. The object is named `clarinet_id01` and is integrated into the planning task with its location `(be clarinet_id01 music_room1)` and availability `(available clarinet_id01)`. The goal generation layer indicates that the senior is still sad. The planning problem is reformulated, and the planner is called to generate a new plan π_2' (shown in Figure C.12) that cheers up the sad senior by providing the `clarinet_id01`.

a_4 : 120.006 : $(carry\ robot1\ clarinet\_id01\ music\_room1)[10.000]$

a_5 : 130.008 : $(move\ robot1\ music\_room1\ corridor2)[40.000]$

a_6 : 170.010 : $(move\ robot1\ corridor2\ corridor1)[50.000]$

a_7 : 220.012 : $(move\ robot1\ corridor1\ bedroom1)[30.000]$

a_8 : 250.014 : $(give\ robot1\ senior1\ clarinet\_id01\ bedroom1)[10.000]$

a_9 : 260.016 : $(cheer\_received\ senior1\ clarinet\_id01)[30.000]$

Figure C.12: The temporal plan π_2'. For the robot to carry `clarinet_id01`, go back to `senior1`'s location, `bedroom1`, and cheer up the sad senior

Scenario 3. In this scenario, plan π_3 (shown in Figure C.13) was generated to comfort a bored senior by providing the object `book_id01` of the type book.

a_1 : 0.000 : $(move\ robot1\ bedroom1\ corridor1)$[30.000]
a_2 : 30.002 : $(move\ robot1\ corridor1\ corridor2)$[50.000]
a_3 : 80.004 : $(move\ robot1\ corridor2\ library1)$[40.000]
a_4 : 120.006 : $(carry\ robot1\ book\_id01\ library1)$[10.000]
a_5 : 130.008 : $(move\ robot1\ library1\ corridor2)$[40.000]
a_6 : 170.010 : $(move\ robot1\ corridor2\ corridor1)$[50.000]
a_7 : 220.012 : $(move\ robot1\ corridor1\ bedroom1)$[30.000]
a_8 : 250.014 : $(give\ robot1\ senior1\ book\_id01\ bedroom1)$[10.000]
a_9 : 260.016 : $(entertainment\_received\ senior1\ book\_id01)$[30.000]

Figure C.13: The temporal plan π_3. For the robot to go to `library1` carry `book_id01`, go back to `senior1`'s location, `bedroom1`, and provide the required entertainment.

During the execution, entertainment objects become unavailable. A discrepancy is detected due to `book_id01`. The system detects a failure. This time, the difference is that perception does not find any alternative objects in the execution environment; alternative objects cannot repair the failure. The planning task is reformulated. The planner is called from the state wherein the failure occurred to generate a new plan. The newly generated plan π'_3 (shown in Figure C.14) includes a suggested action for the bored senior to watch TV (a different course of action).

a_1 : 0.000 : $(suggest\_entertainment2\ robot1\ senior1\ tv\_id01\ bedroom1)$[5.000]
a_2 : 5.002 : $(entertainment2\_suggested\ senior1\ tv\_id01)$[30.000]

Figure C.14: The temporal plan π'_3. For the robot to suggest using `tv_id01` to `senior1` and get the required entertainment.

C.7.2 Validation of Stress Detection

Detecting stress is complicated mainly due to two factors; the determination as to which external elements are necessary to experience stress and the noise introduced when the seniors move their hands.

The dataset we used was divided into two parts: 80% for training and 20% for testing. The confusion matrix obtained using the database as validation is presented in Table C.1.

Table C.1: Confusion matrix for the validation of stress detection using the dataset

		Predicted	
		Stressed	**Un-stressed**
Real	Stressed	95.0	5.0
	Un-stressed	11.0	89.0

When validating using the data acquired by the wristband (shown in Table C.1), a reduction in classification was observed, reducing it to 70%, due mainly to the two factors explained earlier. However, this is not an issue for our system as we do not depend solely on stress but instead continue to recognise the emotion and, only then, generate the goal.

Table C.2: Confusion matrix for the validation of stress detection using the wristband

		Predicted	
		Stressed	**Un-stressed**
Real	Stressed	75.0	25.0
	Un-stressed	20.0	80.0

The statistical data obtained from the stress classification process are analysed to determine if our system performs the classification correctly. The dataset's classification is 85%, with an average precision-recall of 0.8. However, it is 70% without the dataset with an average precision-recall of 0.65.

C.7.3 Validation of Emotions Recognition

A series of experiments were performed to validate the classification of emotions to determine classification accuracy. The robot captured images wherein a person could be observed, trying to imitate a particular emotional state. These images were sent to the web service for analysis.

The result of these experiments is shown in Table C.3, wherein the confusion matrix obtained from our validation process can be seen. The accuracy obtained by the validation with our model is 93.6%. It is important to note that, in some emotions, some values are lower than other emotions due to their facial similarities, such as sadness and anger.

Table C.3: Confusion matrix for the validation of emotion classification

		Predicted						
		Terrified	**Angry**	**Disgusted**	**Happy**	**Neutral**	**Sad**	**Bored**
Real	Terrified	**98.0**	3.1	0.0	2.0	1.3	0.0	0.87
	Angry	0.68	**89.0**	1.1	1.0	1.3	0.0	0.0
	Disgusted	0.0	1.1	**94.0**	0.0	1.3	5.1	0.87
	Happy	0.34	2.1	3.3	**94.0**	2.6	5.1	1.7
	Neutral	0.68	1.1	1.1	3.0	**91.0**	1.7	0.0
	Sad	0.34	1.1	0.0	0.0	0.65	**81.0**	0.87
	Bored	0.34	2.1	0.0	0.0	2.0	6.8	**96.0**

C.7.4 Validation of Objects Classification

Not all the objects used in our system are found in typical datasets. Thus, we decided to build our 17-class dataset. For each class, a total of 200 images were downloaded. A MobileNet (Howard et al. 2017) network was used, and the classification results can be seen in the confusion matrix presented in Table C.4. Some values of the objects in the confusion matrix were lower than others. The objects associated with these values, such as glass bottles, jars, and mugs, have a certain resemblance. This is not an issue for our system as these objects belong to the same category –a drinking vessel– and are intended to be used interchangeably. The lowest rating percentage (79.2%) is associated with the yo-yo, which is difficult to rate due to its shape.

Table C.4: Confusion matrix for the validation of objects classification

		Predicted																	
		book	clarinet	cup	drinking_glass	drum	glass_bottle	guitar	jar	journal	magazine	magic_cube	mug	newspaper	puzzle	story	sudoku	violin	yo-yo
Real	book	**90**	2.0	1.0	1.0	1.5	0.5	0.36	0.36	0.65	0.99	0.36	0.56	0.36	0.36	0.36	0.36	0.96	3.5
	clarinet	1.0	**87**	2.0	0.95	0.65	0.36	0.45	0.23	0.45	0.56	0.45	0	0.25	0.3	0	0.25	0.36	3.0
	cup	0.25	1.0	**90**	0.65	0.45	1.5	0.25	0.58	0.32	0.48	0.98	0.15	0.36	0.54	0.36	0.12	0.48	1.5
	drinking_glass	0.5	0.25	0.56	**88**	0.36	0.35	0.45	0.36	0.12	0.57	0.14	0.19	0.87	0.36	0.36	0.36	0.75	0.35
	drum	0.35	0.89	0.68	3.0	**92**	2.3	0.6	0.45	0.14	0.89	0.56	0.25	0.45	0.9	0.45	0.95	0.35	0.99
	glass_bottle	0.27	0.25	0.47	0.24	0.45	**88**	0.87	0.98	0.36	0.78	0.78	0.36	0.98	2.0	0.71	0.4	0.48	0.78
	guitar	0.5	0.36	0	0.45	0.25	0.69	**90**	0.78	0.98	0.65	0.9	0.98	0.78	0.36	0.56	0.84	3.0	2.5
	jar	0.5	0.48	0	0	1.5	0.95	2.3	**88**	1.5	0.56	0.65	0.31	0.36	0.97	0	3.0	0.34	3.6
	journal	1.5	0.9	0	0.92	0.9	0.35	0.23	2.3	**90**	0.36	0.78	2.5	0.29	0.45	0	0.45	0.14	0.58
	magazine	1.5	1.0	0	0.9	0.45	0.45	0.6	0.12	0.31	**90**	0.7	0.36	0	0.65	0	0.68	0.25	0.56
	magic_cube	0.68	0.96	0.56	0	0	0.67	0.78	0.36	0.98	0.45	**88**	0.45	0	0.31	0.3	0.99	0.89	0.14
	mug	0.25	0.78	0.9	0	0	0.68	0.89	0.69	0.36	0.87	0.98	**88**	0	0.45	0	0	0.35	0.23
	newspaper	0.55	0.95	0.65	0.24	0	0.69	0.45	0.97	0.93	0.98	1.5	2.5	**95**	1.8	0	0	0.35	0.69
	puzzle	0.1	1.0	0.35	0.35	0	0.99	0.63	0.45	0.45	1.0	0.36	0.36	0	**88**	0	0	0.99	0.99
	story	0.1	0.44	0.9	0.32	0	0.78	0.45	0.63	0.99	0	0.97	0.78	0	0.23	**96**	0	0.87	0.58
	sudoku	0.36	0.58	0.8	0.9	0	0.45	0.23	0.48	0.45	0	0.35	0.98	0	0.45	0	**92**	0.45	0.36
	violin	0.36	1.0	0.93	0.98	0	0.36	0.36	1.5	0.36	0	0.9	0.45	0	2.3	0	0	**88**	0.45
	yo-yo	0.98	0.56	0	1.5	0.99	0.43	0.1	0.69	0.25	0.36	0.36	0.32	0	0.07	1.3	0	1.5	**79**

C.8 Discussion

Abstractly, the system can be looked at as providing members of a target group with objects depending on their dynamically recognised emotions and following up on their status evolution in a dynamic execution environment. Since our deliberative function is domain-independent, the system can be further specialised depending on the target group.

The system was not validated with elders in a real AL home, mainly due to the robotic platform we used and the safety regulations. We used the RobElf robotic platform from Robotelf Technologies [4], shown in Figure C.15. RobElf does not have a clamp to pick up objects; it executes voice interaction to simulate actions that require a gripper, such as carrying a glass.

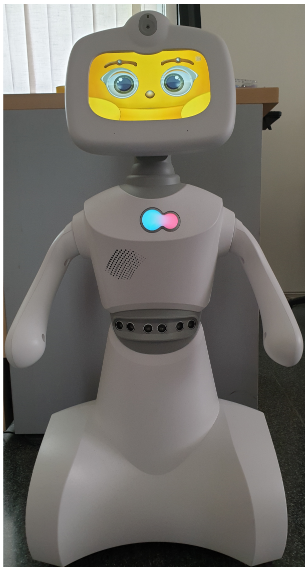


Figure C.15: RobElf robotic platform

[4]Robotelf Technologies https://www.robelf.com

The system was presented to a group of users. They were all researchers in the Department of Computer Systems and Computation. For future research, the system can be adapted following a user-centric methodology to survey impressions from real caregivers in a seniors' living home, acquire approval to conduct live laboratory demonstrations with senior citizens attended by health professionals, and further analyse the demonstrations data to adjust the performance and avoid unforeseen inconvenient situations.

We opted for reactivity to reduce the information load in the current implementation. The system looks for alternative objects only when a failure occurs. On the other hand, the system can proactively look for alternative objects every time the observed state is read. The newly integrated objects can be considered information overload when there are no failures, and they end up being unused. The reactive-proactive design detail is left to the system administrators to meet their execution environment needs.

In the validation section, the system's behaviour was demonstrated with one senior for simplicity; however, the implementation has multi-user capabilities. Temporal planning limited time windows (TILs) supported by PDDL 2.2 were utilised to deal with multiple seniors suffering from negative emotions. When using such literals in actions' conditions, time windows allow priorities. Comforting the terrified emotion is assigned a shorter time window than the ones assigned for comforting the sadness or boredom emotions. The planner handles these constraints when generating the plan. In the current implementation, durative actions are generated to provide a physical object to serve the senior suffering from negative emotions; the work can be extended by adding instantaneous responses, such as turning on the TV and modifying lights intensity, as in (Quinde et al. 2020), and further increasing multi-user capability as the robot is no longer a bottleneck resource.

Our system provides a contextualised explanation when it deviates from the expected behaviour. The explanation is threefold: (1) based on the dynamically generated goal (reason for the original plan); (2) based on what went wrong (violated condition leading to the failure); and (3) based on the plan's fix (reason for fixing using an alternative object or via a recommendation). In the current implementation, we assume that a senior is comforted after a duration of time from receiving the object to soothe the negative emotion. If not comforted, the senior will be detected as stressed, and the process will start again. For future work, this is an area for improvement using explainable AI as people react differently. The system must learn why the suggested solution did not comfort the senior and adapt for future runs.

The solution to certain classes' similarity in classifying emotions and objects is to add more images wherein a marked differentiation between the classes can be observed. Adding more images in which the actors emphasise anger and sad emotions would allow the system to identify these emotions correctly. Similarly, introducing images from different perspectives or rotating the images can improve the classification of the glass bottle, the jar, and the cup. The noise to the wristband when the senior moves the hand can lead to false positives. A possible solution is to incorporate a breathing frequency measurement in the wristband. Thus, the system would have one more variable to infer if the user is exhibiting a state of stress. Another possible solution is to calculate the HR measurements' average over time and send the result to the system for classification.

Concerning new unknown objects' recognition, it is essential to note that we have presented a basic object recognition mechanism with some limitations and assumptions. For example, suppose several new non-IoT objects of the same type are introduced. In that case, the system cannot differentiate them or their states. Although it is not important which object the planner will use since they are of the same type; however, our basic recognition method does not distinguish whether the mugs are full or empty, and we assume that the caretakers handle this. In future work, this must be investigated, and other advanced techniques (Pang et al. 2021) can be used for safe human-to-robot handovers with estimations of the physical properties (such as empty or full) of cups and objects with estimations of the human hands manipulating the container.

C.9 Conclusion

The use of AmI and AL systems is increasing in line with the growing ageing population. Stress is a significant cause of human illnesses directly affecting the emotional state. Literature was reviewed to identify the convergence between deliberation and AmI' latest healthcare trends and the lack of a balanced deliberative function. As the contribution of this study, it describes an integral deliberation function architecture that provides comfort to stressed seniors suffering from negative emotions in an AL environment. The integral deliberation function makes a balanced synthesis of data and knowledge, learning, perception, dynamic goal generation, planning, acting, and monitoring functionalities. Such functionalities are required to detect, address, and follow up on the senior's status evolution in a dynamic execution environment. The five layers constituting our intelligent deliberation pillars are learning and knowledge augmentation, perception, goal generation, planning, acting, and monitoring. Our contribution's differential value lies in the integral view of the architecture: (1) integrates learning, perception, and dynamic goal generation to achieve context-aware human-robot interaction; (2) integrates cognitive knowledge augmentation, learning, and perception to achieve context awareness toward the execution environment; and (3) integrates perception, planning, acting, and monitoring to achieve planning capabilities and reactivity when handling failures.

The behaviour and the approach's validity were demonstrated in three experimental case studies in a simulated AL home scenario. Moreover, the proposed methods for stress detection, emotions, and object recognition were validated to show classification accuracy. The validation demonstrated that the proposed deliberation function has effectively achieved its deliberative objectives in the AL home: perceptiveness and context-aware dynamic human-robot interaction; planning, acting, and monitoring capabilities to achieve reactiveness; and environment context awareness to adapt to the change in the environment.

This chapter of the PhD thesis demonstrates the usefulness and the validity of using the execution simulation system, presented in Chapter 4, the context-aware knowledge acquisition approach, presented in Chapter 5, and other AI techniques in an AmI healthcare application. More importantly, it demonstrates how the author allows an autonomous agent to deal with the dynamic environment change, not depending solely on the prior knowledge of designers but instead relying on its own percepts to provide context-aware deliberative responses for repairing a failure.

References

Abbas, O. M., A. A. Lotfy, A. N. Abdul-Basset, and M. Ei-Abd (2017). “Ambient Intelligence in Automated Houses”. In: *2017 9th IEEE-GCC Conference and Exhibition (GCCCE)*. IEEE, pp. 1–9 (cit. on p. 236).

Aha, D. W. (2018). “Goal Reasoning: Foundations, Emerging Applications, and Prospects”. In: *AI Mag.* 39.2, pp. 3–24 (cit. on p. 115).

Alarcos, A. O., D. Beßler, A. M. Khamis, P. J. S. Gonçalves, M. K. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. I. Olszewska, H. Nakawala, E. P. de Freitas, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, and H. Li (2019). “A review and comparison of ontology-based approaches to robot autonomy”. In: *The Knowledge Engineering Review* 34, e29 (cit. on p. 35).

Amigoni, F., N. Gatti, C. Pinciroli, and M. Roveri (2005). “What planner for ambient intelligence applications?” In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 35.1, pp. 7–21 (cit. on p. 234).

Augusto, J. C., V. Callaghan, D. J. Cook, A. Kameas, and I. Satoh (2013). “"Intelligent Environments: a manifesto"”. In: *Human-Centric Computing And Information Sciences* 3, p. 12 (cit. on pp. 231, 232).

Augusto, J. C. and C. D. Nugent (2004). “The Use of Temporal Reasoning and Management of Complex Events in Smart Homes”. In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI’2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*. Ed. by R. L. de Mántaras and L. Saitta. IOS Press, pp. 778–782 (cit. on p. 235).

Ayan, N. F., U. Kuter, F. Yaman, and R. P. Goldman (2007). “Hotride: Hierarchical ordered task replanning in dynamic environments”. In: *Planning and Plan Execution for Real-World Systems–Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop. Providence, RI*. Vol. 38 (cit. on pp. 26, 27).

Baader, F., I. Horrocks, and U. Sattler (2008). “Description Logics”. In: *Handbook of Knowledge Representation*. Ed. by F. van Harmelen, V. Lifschitz, and B. W. Porter. Vol. 3. Foundations of Artificial Intelligence. Elsevier, pp. 135–179 (cit. on p. 33).

Babli, M., J. Ibáñez-Ruiz, L. Sebastia, A. G. Tejero, and E. Onaindia (2016). “An Intelligent System for Smart Tourism Simulation in a Dynamic Environment”. In: *Proceedings of the 2nd Workshop on Artificial Intelligence and Internet of Things co-located with the 22nd European Conference on Artificial Intelligence (ECAI 2016), The Hague, THE NETHERLANDS, August 30, 2016*. Ed. by C. D. Spyropoulos, G. Pierris, and G. Tzortzis. Vol. 1724.

CEUR Workshop Proceedings. CEUR-WS.org, pp. 15–22 (cit. on pp. 4, 60, 88, 93, 153, 252).

Babli, M., E. Marzal, and E. Onaindia (2018). “On the use of ontologies to extend knowledge in online planning”. In: *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) 2018.* Delft, THE NETHERLANDS., pp. 54–61 (cit. on pp. 4, 91, 98, 211).

Babli, M. and E. Onaindia (2019). “A Context-Aware Knowledge Acquisition for Planning Applications Using Ontologies”. In: *EDUCATION EXCELLENCE AND INNOVATION MANAGEMENT THROUGH VISION 2020*. Ed. by Soliman, KS. 33rd International-Business-Information-Management-Association (IBIMA) Conference, Granada, SPAIN, APR 10-11, 2019. Int Business Informat Management Assoc, 3602–3614 (cit. on pp. 5, 91, 95, 97, 98, 170, 235, 247).

Babli, M., E. Onaindia, and E. Marzal (2018). “Extending Planning Knowledge Using Ontologies for Goal Opportunities”. In: *EDUCATION EXCELLENCE AND INNOVATION MANAGEMENT THROUGH VISION 2020, VOLS I -XI*. Ed. by Soliman, KS. 31st International-Business-Information-Management-Association (IBIMA) Conference, Milan, ITALY, APR 25-26, 2018. Int Business Informat Management Assoc, 3199–3208 (cit. on pp. 4, 91, 94, 97, 98, 103).

Babli, M., J. A. Rincon, E. Onaindia, C. Carrascosa, and V. Julian (Apr. 2021). “Deliberative Context-Aware Ambient Intelligence System for Assisted Living Homes”. In: *HUMAN-CENTRIC COMPUTING AND INFORMATION SCIENCES* 11 (cit. on pp. 5, 88, 95, 97, 99, 120, 230, 247).

Babli, M., Ó. Sapena, and E. Onaindia (2023). “Plan commitment: Replanning versus plan repair”. In: *Engineering Applications of Artificial Intelligence* 123, p. 106275 (cit. on pp. 5, 117).

Bäckström, C. and B. Nebel (1995). “Complexity Results for SAS+ Planning”. In: *Computational Intelligence* 11, pp. 625–656 (cit. on p. 146).

Bai, H., D. Hsu, and W. S. Lee (2014). “Integrated perception and planning in the continuous space: A POMDP approach”. In: *The International Journal of Robotics Research* 33.9, pp. 1288–1302 (cit. on p. 124).

Bajo, J., V. Julián, J. M. Corchado, C. Carrascosa, Y. de Paz, V. J. Botti, and J. F. de Paz (2008). “An execution time planner for the ARTIS agent architecture”. In: *Engineering Applications of Artificial Intelligence* 21.5, pp. 769–784 (cit. on p. 120).

Bechon, P., C. Lesire, and M. Barbier (2020). “Hybrid planning and distributed iterative repair for multi-robot missions with communication losses”. In: *Autonomous Robots* 44.3-4, pp. 505–531 (cit. on pp. 121, 127).

Beetz, M., D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoglu, and G. Bartels (2018). “Know Rob 2.0 - A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents”. In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, pp. 512–519 (cit. on p. 36).

Benelam, B. and L. Wyness (2010). “Hydration and health: a review”. In: *Nutrition Bulletin* 35.1, pp. 3–25 (cit. on p. 236).

Benton, J., A. J. Coles, and A. Coles (2012). “Temporal Planning with Preferences and Time-Dependent Continuous Costs”. In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. Ed. by L. McCluskey, B. C. Williams, J. R. Silva, and B. Bonet. AAAI (cit. on pp. 21, 22, 70).

Bermejo-Alonso, J. (2018). “Reviewing Task and Planning Ontologies: An Ontology Engineering Process”. In: *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2018, Volume 2: KEOD, Seville, Spain, September 18-20, 2018*. Ed. by D. Aveiro, J. L. G. Dietz, and J. Filipe. SciTePress, pp. 181–188 (cit. on pp. 38, 41).

Berners-Lee, T., J. Hendler, and O. Lassila (2001). “The Semantic Web”. In: *Scientific American* 284.5, pp. 34–43 (cit. on pp. 31, 33).

Beßler, D., M. Pomarlan, A. Akbari, Muhayyuddin, M. Diab, J. Rosell, J. A. Bateman, and M. Beetz (2018). “Assembly Planning in Cluttered Environments Through Heterogeneous Reasoning”. In: *KI 2018: Advances in Artificial Intelligence - 41st German Conference on AI, Berlin, Germany, September 24-28, 2018, Proceedings*. Ed. by F. Trollmann and A. Turhan. Vol. 11117. Lecture Notes in Computer Science. Springer, pp. 201–214 (cit. on pp. 38, 39, 42).

Bidot, J., B. Schattenberg, and S. Biundo (2008). “Plan Repair in Hybrid Planning”. In: *KI 2008: Advances in Artificial Intelligence, 31st Annual German Conference on AI*. Ed. by A. Dengel, K. Berns, T. M. Breuel, F. Bomarius, and

T. Roth-Berghofer. Vol. 5243. Lecture Notes in Computer Science. Springer, pp. 169–176 (cit. on p. 125).

Blois, M., M. Escobar, and R. Choren (2007). “Using Agents and Ontologies for Application Development on the Semantic Web”. In: *J. Braz. Comput. Soc.* 13.2, pp. 35–44 (cit. on p. 36).

Bonet, B. and H. Geffner (2001). “Planning as heuristic search”. In: *Artificial Intelligence* 129.1-2, pp. 5–33 (cit. on p. 20).

Borrajo, D. and M. Veloso (2021). “Computing Opportunities to Augment Plans for Novel Replanning during Execution”. In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICASP 2021, Guangzhou, China (virtual), August 2-13, 2021*. Ed. by S. Biundo, M. Do, R. Goldman, M. Katz, Q. Yang, and H. H. Zhuo. AAAI Press, pp. 51–55 (cit. on pp. 26, 30, 115).

Bouguerra, A., L. Karlsson, and A. Saffiotti (2007). “Active execution monitoring using planning and semantic knowledge”. In: *ICAPS Workshop on Planning and Plan Execution for Real-World Systems* (cit. on pp. 35, 41).

Bozan, K. and A. Berger (2019). “Revisiting the Technology Challenges and Proposing Enhancements in Ambient Assisted Living for the Elderly”. In: *52nd Hawaii International Conference on System Sciences, HICSS 2019, Grand Wailea, Maui, Hawaii, USA, January 8-11, 2019.* Ed. by T. Bui. ScholarSpace, pp. 1–10 (cit. on pp. 231, 235).

Brooks, F. P. (1987). “No Silver Bullet - Essence and Accidents of Software Engineering”. In: *Computer* 20.4, pp. 10–19 (cit. on p. 63).

Bruno, B., C. T. Recchiuto, I. Papadopoulos, A. Saffiotti, C. Koulouglioti, R. Menicatti, F. Mastrogiovanni, R. Zaccaria, and A. Sgorbissa (2019). “Knowledge Representation for Culturally Competent Personal Robots: Requirements, Design Principles, Implementation, and Assessment”. In: *Int. J. Soc. Robotics* 11.3, pp. 515–538 (cit. on pp. 37, 42).

Buhalis, D. and A. Amaranggana (2015). “Smart Tourism Destinations Enhancing Tourism Experience Through Personalisation of Services”. In: *Information and Communication Technologies in Tourism 2015, ENTER 2015, Proceedings of the International Conference in Lugano, Switzerland, February 3 - 6, 2015*. Ed. by I. Tussyadiah and A. Inversini. Springer, pp. 377–389 (cit. on p. 61).

Burns, M. N., M. Begale, J. Duffecy, D. Gergle, C. J. Karr, E. Giangrande, and D. C. Mohr (2011). “Harnessing Context Sensing to Develop a Mobile Intervention for Depression”. In: *Journal of Medical Internet Research* 13.3 (cit. on p. 234).

Calvo, M. G., A. Fernández-Martín, G. Recio, and D. Lundqvist (2018). “Human observers and automated assessment of dynamic emotional facial expressions: KDEF-dyn database validation”. In: *Frontiers in Psychology* 9, p. 2052 (cit. on p. 245).

Carreno, Y., J. S. Willners, Y. R. Petillot, and R. P. Petrick (2021). “Situation-Aware Task Planning for Robust AUV Exploration in Extreme Environments”. In: *Proceedings of the IJCAI Workshop on Robust and Reliable Autonomy in the Wild* (cit. on p. 126).

Cashmore, M., M. Fox, D. Long, D. Magazzeni, and B. Ridder (2016). “Opportunistic planning for increased plan utility”. In: *Proceedings of the 4th ICAPS Workshop on Planning and Robotics (PlanRob 2016)* (cit. on p. 89).

Cashmore, M., M. Fox, D. Long, D. Magazzeni, and B. Ridder (2018). “Opportunistic Planning in Autonomous Underwater Missions”. In: *IEEE Trans Autom. Sci. Eng.* 15.2, pp. 519–530 (cit. on pp. 26, 29, 54, 88, 89).

Castelfranchi, C. and R. Falcone (2003). “From Automaticity to Autonomy: The Frontier of Artificial Agents”. In: *Agent Autonomy*. Ed. by H. Hexmoor, C. Castelfranchi, and R. Falcone. Boston, MA: Springer US, pp. 103–136 (cit. on p. 178).

Castellini, A., E. Marchesini, and A. Farinelli (2021). “Partially Observable Monte Carlo Planning with state variable constraints for mobile robot navigation”. In: *Engineering Applications of Artificial Intelligence* 104, p. 104382 (cit. on p. 120).

Cenamor, I., T. de la Rosa, M. Vallati, F. Fernández, and L. Chrpa (2018). “TemPoRal: Temporal Portfolio Algorithm”. In: *2018 International Planning Competition, Temporal Tack* (cit. on p. 145).

Chen, C., R. Xu, S. Zhu, Z. Li, and H. Jiang (2020). “RPRS: a reactive plan repair strategy for rapid response to plan failures of deep space missions”. In: *Acta Astronautica* 175, pp. 155–162 (cit. on pp. 121, 125).

Chen, L. and C. D. Nugent (2019). “Semantic Smart Homes: Situation-Aware Assisted Living”. In: *Human Activity Recognition and Behaviour Analysis: For Cyber-Physical Systems in Smart Environments*. Cham: Springer International Publishing, pp. 201–215 (cit. on p. 232).

Chen, Y., B. W. Wah, and C. Hsu (2006). “Temporal Planning using Subgoal Partitioning and Resolution in SGPlan”. In: *J. Artif. Intell. Res.* 26, pp. 323–369 (cit. on p. 21).

Chien, S. A. and R. Morris (2014). “Space Applications of Artificial Intelligence”. In: *AI Magazine* 35.4, pp. 3–6 (cit. on p. 120).

Chopra, A. K., A. Artikis, J. Bentahar, M. Colombetti, F. Dignum, N. Fornara, A. J. I. Jones, M. P. Singh, and P. Yolum (2013). “Research directions in agent communication”. In: *ACM Trans. Intell. Syst. Technol.* 4.2, 20:1–20:23 (cit. on p. 55).

Cocchiarella, N. B. (2007). “Formal ontology and conceptual realism”. In: *Formal Ontology and Conceptual Realism*. Springer, pp. 3–24 (cit. on p. 35).

Coddington, A. M., M. Fox, J. Gough, D. Long, and I. Serina (2005). “MADbot: A Motivated and Goal Directed Robot”. In: *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. Ed. by M. M. Veloso and S. Kambhampati. AAAI Press / The MIT Press, pp. 1680–1681 (cit. on pp. 26, 28).

Cohen, W. W., P. Ravikumar, and S. E. Fienberg (2003). “A Comparison of String Distance Metrics for Name-Matching Tasks”. In: *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico*. Ed. by S. Kambhampati and C. A. Knoblock, pp. 73–78 (cit. on p. 107).

Colombo, A., C. Cusano, and R. Schettini (2011). “UMB-DB: A database of partially occluded 3D faces”. In: *IEEE International Conference on Computer*

Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011. IEEE Computer Society, pp. 2113–2119 (cit. on p. 245).

Cooksey, P. and M. M. Veloso (2017). “Intra-robot replanning to enable team plan conditions”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. IEEE, pp. 1113–1118 (cit. on p. 66).

Coppin, B. (2004). *Artificial intelligence illuminated*. Jones & Bartlett Learning (cit. on pp. 9, 52).

Cox, M. T. (2007). “Perpetual Self-Aware Cognitive Agents”. In: *AI Mag.* 28.1, pp. 32–46 (cit. on pp. 26, 54, 89).

Csikszentmihalyi, M. (1988). “The flow experience and its significance for human psychology”. In: *Optimal Experience: Psychological Studies of Flow in Consciousness*. Ed. by M. Csikszentmihalyi and I. S. Csikszentmihalyi. Cambridge University Press, pp. 15–35 (cit. on p. 51).

Cushing, W. and S. Kambhampati (2005). “Replanning: A new perspective”. In: *Proceedings of the International Conference on Automated Planning and Scheduling Monterey, USA*, pp. 13–16 (cit. on p. 128).

Dannenhauer, D. and H. Muñoz-Avila (2013). “LUIGi: a goal-driven autonomy agent reasoning with ontologies”. In: *Advances in Cognitive Systems (ACS-13)* (cit. on pp. 26, 27, 36, 38, 41, 54, 89).

Dannenhauer, D. (2016). “Self Monitoring, Goal Driven Autonomy Agents”. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial*

Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. Ed. by S. Kambhampati. IJCAI/AAAI Press, pp. 3986–3987 (cit. on p. 185).

Dannenhauer, D. and H. Muñoz-Avila (2015). “Goal-Driven Autonomy with Semantically-Annotated Hierarchical Cases”. In: *Case-Based Reasoning Research and Development - 23rd International Conference, ICCBR 2015, Frankfurt am Main, Germany, September 28-30, 2015, Proceedings*. Ed. by E. Hüllermeier and M. Minor. Vol. 9343. Lecture Notes in Computer Science. Springer, pp. 88–103 (cit. on pp. 26, 27, 40, 89).

Daoutis, M., S. Coradeschi, and A. Loutfi (2009). “Grounding commonsense knowledge in intelligent systems”. In: *Journal of Ambient Intelligence and Smart Environments* 1.4, pp. 311–321 (cit. on p. 235).

David, J. and J. Euzenat (2008). “Comparison between Ontology Distances (Preliminary Results)”. In: *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*. Ed. by A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan. Vol. 5318. Lecture Notes in Computer Science. Springer, pp. 245–260 (cit. on p. 106).

Davis, R., H. E. Shrobe, and P. Szolovits (1993). “What Is a Knowledge Representation?” In: *AI Mag.* 14.1, pp. 17–33 (cit. on p. 31).

Dawkins, R. and N. Davis (2017). *The selfish gene*. Macat Library (cit. on p. 85).

De Waal, F. (2017). *The surprising science of alpha males*. TED (cit. on p. 119).

De Waal, F. and F. B. Waal (2007). *Chimpanzee politics: Power and sex among apes*. Johns Hopkins University Press (cit. on p. 119).

Deliyanni, A. and R. A. Kowalski (1979). “Logic and Semantic Networks”. In: *Commun. ACM* 22.3, pp. 184–192 (cit. on p. 32).

Diab, M., A. Akbari, M. U. Din, and J. Rosell (2019). “PMK - A Knowledge Processing Framework for Autonomous Robotics Perception and Manipulation”. In: *Sensors* 19.5, p. 1166 (cit. on p. 37).

Diab, M., M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. A. Bateman, and M. Beetz (2019). “An Ontology for Failure Interpretation in Automated Planning and Execution”. In: *Robot 2019: Fourth Iberian Robotics Conference - Advances in Robotics, Volume 1, Porto, Portugal, 20-22 November, 2019*. Ed. by M. F. Silva, J. L. Lima, L. P. Reis, A. Sanfeliu, and D. Tardioli. Vol. 1092. Advances in Intelligent Systems and Computing. Springer, pp. 381–390 (cit. on pp. 38, 42).

Doan, A., J. Madhavan, R. Dhamankar, P. M. Domingos, and A. Y. Halevy (2003). “Learning to match ontologies on the Semantic Web”. In: *VLDB J.* 12.4, pp. 303–319 (cit. on p. 113).

Edelkamp, S. and J. Hoffmann (2004). *PDDL2. 2: The language for the classical part of the 4th international planning competition*. Tech. rep. 195, University of Freiburg (cit. on pp. 15, 48).

Fehr, E. and U. Fischbacher (2003). “The nature of human altruism”. In: *Nature* 425.6960, pp. 785–791 (cit. on p. 119).

Fikes, R., P. E. Hart, and N. J. Nilsson (1972). “Learning and Executing Generalized Robot Plans”. In: *Artificial Intelligence* 3.1-3, pp. 251–288 (cit. on p. 125).

Fikes, R. and N. J. Nilsson (1971). “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”. In: *Artificial Intelligence* 2.3/4, pp. 189–208 (cit. on p. 13).

Fox, M., A. Gerevini, D. Long, and I. Serina (2006). “Plan Stability: Replanning versus Plan Repair”. In: *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*. Ed. by D. Long, S. F. Smith, D. Borrajo, and L. McCluskey. AAAI, pp. 212–221 (cit. on pp. 26, 28, 55, 79, 122, 125, 128, 129, 135).

Fox, M. and D. Long (2002). “PDDL+: Modeling continuous time dependent effects”. In: *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*. Vol. 4, p. 34 (cit. on p. 16).

Fox, M. and D. Long (2003). “PDDL2. 1: An extension to PDDL for expressing temporal planning domains”. In: *Journal of artificial intelligence research* 20, pp. 61–124 (cit. on p. 15).

Furelos-Blanco, D. and A. Jonsson (2018). “CP4TP: A Classical Planning for Temporal Planning Portfolio”. In: *2018 International Planning Competition, Temporal Tack* (cit. on p. 145).

Garrido, A., E. Onaindia, and O. Sapena (2008). “Planning and scheduling in an e-learning environment. A constraint-programming-based approach”. In: *Engineering Applications of Artificial Intelligence* 21.5. Constraint Satisfaction Techniques for Planning and Scheduling Problems, pp. 733–743 (cit. on p. 120).

Gerevini, A. and D. Long (2005). “Plan Constraints and Preferences in PDDL 3 The Language of the Fifth International Planning Competition”. In: *5th International Planning Competition (IPC), at ICAPS* (cit. on p. 15).

Gerevini, A., A. Saetti, and I. Serina (2006). “An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events”. In: *JAIR* 25, pp. 187–231 (cit. on p. 20).

Gerevini, A., P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos (2009). “Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners”. In: *Artificial Intelligence* 173.5-6, pp. 619–668 (cit. on p. 70).

Gerevini, A., A. Saetti, and I. Serina (2003). “Planning Through Stochastic Local Search and Temporal Action Graphs in LPG”. In: *JAIR* 20, pp. 239–290 (cit. on p. 20).

Gerevini, A. and I. Serina (2000). “Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques”. In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*. Ed. by S. A. Chien, S. Kambhampati, and C. A. Knoblock. Breckenridge, CO, USA: AAAI, pp. 112–121 (cit. on pp. 121, 125).

Ghallab, M., D. Nau, and P. Traverso (2014). “The actor’s view of automated planning and acting: A position paper”. In: *Artif. Intell.* 208, pp. 1–17 (cit. on p. 120).

Giménez-Manuel, J. G., J. C. Augusto, and J. Stewart (2022). “AnAbEL: towards empowering people living with dementia in ambient assisted living”. In:

Universal Access in the Information Society 21.2, pp. 457–476 (cit. on p. 231).

Goldman, R., U. Kuter, and R. Freedman (2020). “Stable Plan Repair for State-Space HTN Planning”. In: *The 3rd ICAPS Workshop on Hierarchical Planning (HPlan 2020)*, pp. 27–35 (cit. on pp. 120, 122, 125).

Goldman, R. P. and U. Kuter (2019). “Hierarchical Task Network Planning in Common Lisp: the case of SHOP3”. In: *Proceedings of the 12th European Lisp Symposium (ELS 2019), Genova, Italy, April 1-2, 2019*. Ed. by N. Neuss. ELSAA, pp. 73–80 (cit. on p. 22).

Gómez-Pérez, A., M. Fernández-López, and Ó. Corcho (2004). *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer (cit. on p. 34).

Gracia, J. and K. Asooja (2013). “Monolingual and cross-lingual ontology matching with CIDER-CL: evaluation report for OAEI 2013”. In: *Proceedings of the 8th International Workshop on Ontology Matching co-located with the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, October 21, 2013*. Ed. by P. Shvaiko, J. Euzenat, K. Srinivas, M. Mao, and E. Jiménez-Ruiz. Vol. 1111. CEUR Workshop Proceedings. CEUR-WS.org, pp. 109–116 (cit. on p. 109).

Gradl, S., M. Wirth, R. Richer, N. Rohleder, and B. M. Eskofier (2019). “An Overview of the Feasibility of Permanent, Real-Time, Unobtrusive Stress Measurement with Current Wearables”. In: *Proceedings of the 13th EAI International Conference on Pervasive Computing Technologies for Health-*

care, PervasiveHealth 2019, Trento, Italy, 20-23 May 2019. Ed. by O. Mayora, S. Forti, J. Meyer, and L. Mamykina. ACM, pp. 360–365 (cit. on p. 235).

Gréa, A. (2020). “Méta-langage endomorphe et planification abstraite pour la reconnaissance des intentions en temps réel. (Endomorphic metalanguage and abstract planning for real-time intent recognition)”. PhD thesis. University of Lyon, France (cit. on pp. 87, 115).

Gretzel, U., M. Sigala, Z. Xiang, and C. Koo (2015). “Smart tourism: foundations and developments”. In: *Electronic Markets* 25.3, pp. 179–188 (cit. on p. 61).

Group, O. W. et al. (2009). “OWL 2 Web Ontology Language Document Overview: W3C Recommendation 27 October 2009”. In: (cit. on p. 37).

Gruber, T. R. (1995). “Toward principles for the design of ontologies used for knowledge sharing?” In: *Int. J. Hum. Comput. Stud.* 43.5-6, pp. 907–928 (cit. on p. 34).

Guarino, N., D. Oberle, and S. Staab (2009). “What is an ontology?” In: *Handbook on ontologies*. Springer, pp. 1–17 (cit. on p. 35).

Guzman, C., P. Castejón, E. Onaindia, and J. Frank (2015). “Reactive execution for solving plan failures in planning control applications”. In: *Integrated Computer-Aided Engineering* 22.4, pp. 343–360 (cit. on p. 88).

Haage, M., J. Malec, A. Nilsson, K. Nilsson, and S. Nowaczyk (2011). “Declarative-knowledge-based reconfiguration of automation systems using a blackboard architecture”. In: *Eleventh Scandinavian Conference on Artificial Intelligence, SCAI 2011, Trondheim, Norway, May 24th - 26th, 2011*. Ed. by A.

Kofod-Petersen, F. Heintz, and H. Langseth. Vol. 227. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 163–172 (cit. on pp. 36, 41).

Hammond, K. J. (1990). “Explaining and Repairing Plans That Fail”. In: *Artificial intelligence* 45.1-2, pp. 173–228 (cit. on p. 125).

Hanheide, M., N. Hawes, J. Wyatt, M. Göbelbecker, M. Brenner, K. Sjöö, A. Aydemir, P. Jensfelt, H. Zender, and G.-J. Kruijff (2010). “A framework for goal generation and management”. In: *Proceedings of the AAAI workshop on goal-directed autonomy* (cit. on pp. 26, 28, 38, 89).

Hawes, N., M. Hanheide, J. Hargreaves, B. Page, H. Zender, and P. Jensfelt (2011). “Home alone: Autonomous extension and correction of spatial representations”. In: *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*. IEEE, pp. 3907–3914 (cit. on pp. 26, 28, 38, 89).

Helmert, M. (2006). “The Fast Downward Planning System”. In: *JAIR* 26, pp. 191–246 (cit. on pp. 21, 152).

Hitzler, P. (2021). “A Review of the Semantic Web Field”. In: *Commun. ACM* 64.2, pp. 76–83 (cit. on pp. 33, 34).

Hoffmann, J. (2003). “The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables”. In: *Journal of artificial intelligence research* 20, pp. 291–341 (cit. on pp. 148, 152).

Hoffmann, J. and R. I. Brafman (2005). “Contingent Planning via Heuristic Forward Search with Implicit Belief States”. In: *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS*

2005). Ed. by S. Biundo, K. L. Myers, and K. Rajan. AAAI, pp. 71–80 (cit. on p. 124).

Hoffmann, J. and B. Nebel (2001). “The FF Planning System: Fast Plan Generation Through Heuristic Search”. In: *JAIR* 14, pp. 253–302 (cit. on p. 20).

Horrocks, I., P. F. Patel-Schneider, and F. Van Harmelen (2003). “From SHIQ and RDF to OWL: The making of a web ontology language”. In: *Journal of web semantics* 1.1, pp. 7–26 (cit. on pp. 34, 50).

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2017). “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (cit. on pp. 235, 260).

Howey, R., D. Long, and M. Fox (2004a). “VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL”. In: *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*. IEEE Computer Society, pp. 294–301 (cit. on p. 64).

Howey, R., D. Long, and M. Fox (2004b). “VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL”. In: *16th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, pp. 294–301 (cit. on p. 23).

Ibáñez-Ruiz, J., L. Sebastia, and E. Onaindia (2016). “Planning Tourist Agendas for Different Travel Styles”. In: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The*

Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016). Vol. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 1818–1823 (cit. on pp. 69, 72, 190, 203).

Ingrand, F. and M. Ghallab (2017). “Deliberation for autonomous robots: A survey”. In: *Artificial Intelligence* 247, pp. 10–44 (cit. on pp. 10, 123, 232, 234).

Jiang, Y., H. Yedidsion, S. Zhang, G. Sharon, and P. Stone (2019). “Multi-robot planning with conflicts and synergies”. In: *Autonomous Robots* 43.8, pp. 2011–2032 (cit. on p. 155).

Joslin, D. and M. E. Pollack (1994). “Least-Cost Flaw Repair: A Plan Refinement Strategy for Partial-Order Planning”. In: *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2*. Ed. by B. Hayes-Roth and R. E. Korf. AAAI Press / The MIT Press, pp. 1004–1009 (cit. on p. 135).

Kambhampati, S. (1990). “Mapping and Retrieval During Plan Reuse: A Validation Structure Based Approach”. In: *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes*. Ed. by H. E. Shrobe, T. G. Dietterich, and W. R. Swartout. AAAI Press / The MIT Press, pp. 170–175 (cit. on p. 55).

Karpas, E. and D. Magazzeni (2020). “Automated Planning for Robotics”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1, pp. 417–439 (cit. on pp. 10, 120).

Klenk, M., M. Molineaux, and D. W. Aha (2013). “Goal-Driven Autonomy for Responding to Unexpected Events in Strategy Simulations”. In: *Comput. Intell.* 29.2, pp. 187–206 (cit. on pp. 26, 27, 54, 89).

Klyne, G. (2004). “Resource description framework (RDF): Concepts and abstract syntax”. In: *http://www. w3. org/TR/2004/REC-rdf-concepts-20040210/* (cit. on p. 34).

Knoblock, C. A., S. Minton, J. L. Ambite, M. Muslea, J. Oh, and M. Frank (2001). “Mixed-initiative, multi-source information assistants”. In: *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*. Ed. by V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko. ACM, pp. 697–707 (cit. on p. 88).

Koenig, S. and M. Likhachev (2005). “Fast Replanning for Navigation in Unknown Terrain”. In: *IEEE TRANSACTIONS ON ROBOTICS* 21.3, pp. 354–363 (cit. on p. 120).

Komenda, A., P. Novák, and M. Pěchouček (2014). “Domain-independent multiagent plan repair”. In: *Journal of Network and Computer* 37, pp. 76–88 (cit. on p. 127).

Kovacs, D. L. (2011). “Complete BNF description of PDDL 3.1”. In: *Dept. Meas. Inf. Syst., Budapest Univ. Technol. Econ., Budapest, Hungary* (cit. on p. 15).

Kraus, S. (1997). “Negotiation and Cooperation in Multi-Agent Environments”. In: *Artif. Intell.* 94.1-2, pp. 79–97 (cit. on p. 55).

Krogt, R. van der and M. de Weerdt (2005). “Plan Repair as an Extension of Planning”. In: *Proceedings of the Fifteenth International Conference on*

Automated Planning and Scheduling (ICAPS 2005), Monterey, California, USA. Ed. by S. Biundo, K. L. Myers, and K. Rajan. AAAI, pp. 161–170 (cit. on pp. 121, 125).

Lees, M., R. Ewald, R. Minson, and G. Theodoropoulos (2009). “Simulation Engines for Multi-Agent Systems”. In: *Multi-Agent Systems - Simulation and Applications*. Ed. by A. M. Uhrmacher and D. Weyns. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press / Taylor & Francis, pp. 77–105 (cit. on p. 63).

Levinson, R. (1997). “The planning and execution assistant and trainer (PEAT)”. In: *The Journal of head trauma rehabilitation* 12.2, pp. 85–91 (cit. on p. 235).

Lima, O., M. Cashmore, D. Magazzeni, A. Micheli, and R. Ventura (2020). “Robust Plan Execution with Unexpected Observations”. In: *CoRR* abs/2003.09401 (cit. on p. 126).

Ling, H., C. Qian, W. Kang, C. Liang, and H. Chen (2019). “Combination of Support Vector Machine and K-Fold cross validation to predict compressive strength of concrete in marine environment”. In: *Construction and Building Materials* 206, pp. 355–363 (cit. on p. 244).

López, P. Á., M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. WieBner (2018). “Microscopic Traffic Simulation using SUMO”. In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*. Ed. by W. Zhang, A. M. Bayen, J. J. S. Medina, and M. J. Barth. IEEE, pp. 2575–2582 (cit. on p. 64).

Machado, L. M. O., M. B. Almeida, and R. R. Souza (2020). “What Researchers are Currently Saying about Ontologies: A Review of Recent Web of Science Articles”. In: *KO KNOWLEDGE ORGANIZATION* 47.3, pp. 199–219 (cit. on p. 34).

Maedche, A. and S. Staab (2001). “Ontology Learning for the Semantic Web”. In: *IEEE Intell. Syst.* 16.2, pp. 72–79 (cit. on p. 33).

Maliah, S., G. Shani, and R. Stern (2017). “Collaborative privacy preserving multi-agent planning - Planners and heuristics”. In: *Auton. Agents Multi Agent Syst.* 31.3, pp. 493–530 (cit. on p. 55).

Marzal, E., M. Babli, E. Onaindia, and L. Sebastia (2017). “Handling PDDL3.0 State Trajectory Constraints with Temporal Landmarks”. In: *In:Proceedings of the Workshop on Constraint Satisfaction for Planning and Scheduling (COPLAS). ICAPS17*. Pittsburgh, USA, June 20, 2017 (cit. on pp. 79, 80).

McDermott, D., M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins (1998). “PDDL–the planning domain definition language–version 1.2”. In: *Yale Center for Computational Vision and Control, Tech. Rep. CVC TR-98-003/DCS TR-1165* (cit. on pp. 14, 62).

McNeill, F., A. Bundy, and C. Walton (2005). “Planning from rich ontologies through translation betweeen representations”. In: *Proceedings of the Workshop On The Role of Ontologies in Planning and Scheduling, the 15th International Conference on Automated Planning and Scheduling (ICAPS)* (cit. on pp. 47, 87).

Meditskos, G. and I. Kompatsiaris (2017). “iKnow: Ontology-driven situational awareness for the recognition of activities of daily living”. In: *Pervasive and Mobile Computing* 40, pp. 17–41 (cit. on p. 232).

Megginson, L. C. (1963). “Lessons from Europe for American Business”. In: *The Southwestern Social Science Quarterly* 44.1, pp. 3–13 (cit. on p. 118).

Methnani, L., A. A. Tubella, V. Dignum, and A. Theodorou (2021). “Let Me Take Over: Variable Autonomy for Meaningful Human Control”. In: *Frontiers in Artificial Intelligence* 4, p. 737072 (cit. on p. 186).

Miguez, A., C. Soares, J. M. Torres, P. Sobral, and R. S. Moreira (2019). “Improving Ambient Assisted Living Through Artificial Intelligence”. In: *New Knowledge in Information Systems and Technologies - Volume 2, World Conference on Information Systems and Technologies, WorldCIST 2019, Galicia, Spain, 16-19 April*. Ed. by Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo. Vol. 931. Advances in Intelligent Systems and Computing. Springer, pp. 110–123 (cit. on p. 233).

Mihailidis, A., B. Carmichael, and J. Boger (2004). “The use of computer vision in an intelligent environment to support aging-in-place, safety, and independence in the home”. In: *IEEE Transactions on information technology in biomedicine* 8.3, pp. 238–247 (cit. on p. 234).

Mínguez, I., D. Berrueta, and L. Polo (2010). “CRUZAR: An application of semantic matchmaking to e-tourism”. In: *Cases on semantic interoperability for information systems integration: Practices and applications*. IGI Global, pp. 255–271 (cit. on pp. 53, 62).

Minsky, M. (2019). *A framework for representing knowledge*. de Gruyter (cit. on p. 32).

Mohalik, S. K., M. B. Jayaraman, R. Badrinath, and A. V. Feljan (2018). “HIPR: An Architecture for Iterative Plan Repair in Hierarchical Multi-agent Systems”. In: *Journal of Computers* 13.3, pp. 351–359 (cit. on pp. 66, 127).

Mol, L. D. (2006). “Closing the Circle: An Analysis of Emil Post’s Early Work”. In: *The Bulletin of Symbolic Logic* 12.2, pp. 267–289 (cit. on p. 33).

Motta, E., N. Shadbolt, A. Stutt, and N. Gibbins, eds. (2004). *Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004, Whittlebury Hall, UK, October 5-8, 2004, Proceedings*. Vol. 3257. Lecture Notes in Computer Science. Springer (cit. on p. 88).

Muscettola, N., P. P. Nayak, B. Pell, and B. C. Williams (1998). “Remote Agent: To Boldly Go Where No AI System Has Gone Before”. In: *Artificial Intelligence* 103.1-2, pp. 5–47 (cit. on p. 88).

Musen, M. A. (2015). “The protégé project: a look back and a look forward”. In: *AI Matters* 1.4, pp. 4–12 (cit. on p. 103).

Nau, D. S., T. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman (2003). “SHOP2: An HTN Planning System”. In: *J. Artif. Intell. Res.* 20, pp. 379–404 (cit. on p. 22).

Nebel, B. and J. Koehler (1995). “Plan Reuse Versus Plan Generation: A Theoretical and Empirical Analysis”. In: *Artificial intelligence* 76.1-2, pp. 427–454 (cit. on p. 121).

Nesse, R. M. (2001). *Evolution and the capacity for commitment*. Russell Sage Foundation (cit. on p. 119).

Nesse, R. M. (2007). “Why a lot of people with selfish genes are pretty nice except for their hatred of The Selfish Gene”. In: *Richard Dawkins: how a scientist changed the way we think*. Ed. by A. Grafen and M. Ridley. Oxford University Press, pp. 203–212 (cit. on p. 119).

Neuhofer, B., D. Buhalis, and A. Ladkin (2015). “Smart technologies for personalized experiences: a case study in the hospitality domain”. In: *Electronic Markets* 25.3, pp. 243–254 (cit. on pp. 53, 62).

Nguyen, T., S. W. Loke, T. Torabi, and H. Lu (2011). “PlaceComm: A framework for context-aware applications in place-based virtual communities”. In: *Journal of Ambient Intelligence and Smart Environments* 3.1, pp. 51–64 (cit. on p. 235).

Nguyen, T. A., M. B. Do, A. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati (2012). “Generating diverse plans to handle unknown and partially known user preferences”. In: *Artificial Intelligence* 190, pp. 1–31 (cit. on pp. 129, 135).

Norman, T. J. and D. Long (1995). “Alarms: An Implementation of Motivated Agency”. In: *Intelligent Agents II, Agent Theories, Architectures, and Languages, IJCAI ’95, Workshop (ATAL), Montreal, Canada, August 19-20, 1995, Proceedings*. Ed. by M. J. Wooldridge, J. P. Müller, and M. Tambe. Vol. 1037. Lecture Notes in Computer Science. Springer, pp. 219–234 (cit. on pp. 25, 26, 89).

Nwana, H. S., L. C. Lee, and N. R. Jennings (1997). “Co-ordination in Multi-Agent Systems”. In: *Software Agents and Soft Computing: Towards Enhancing Machine Intelligence, Concepts and Applications*. Ed. by H. S. Nwana and N. Azarmi. Vol. 1198. Lecture Notes in Computer Science. Springer, pp. 42–58 (cit. on p. 55).

Oguego, C. L., J. C. Augusto, A. M. Ortega, and M. Springett (2018). “Using argumentation to manage users’ preferences”. In: *Future Generation Computer Systems* 81, pp. 235–243 (cit. on p. 234).

Ong, S. C. W., S. W. Png, D. Hsu, and W. S. Lee (2010). “Planning under Uncertainty for Robotic Tasks with Mixed Observability”. In: *The International Journal of Robotics Research* 29.8, pp. 1053–1068 (cit. on p. 120).

Palacios, H. and H. Geffner (2009). “Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width”. In: *Journal of Artificial Intelligence Research* 35, pp. 623–675 (cit. on p. 124).

Pang, Y. L., A. Xompero, C. Oh, and A. Cavallaro (2021). “Towards safe human-to-robot handovers of unknown containers”. In: *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. IEEE, pp. 51–58 (cit. on p. 265).

Patel, A. and S. Jain (2018). “Formalisms of Representing Knowledge”. In: *Procedia Computer Science* 125. The 6th International Conference on Smart Computing and Communications, pp. 542–549 (cit. on p. 32).

Paulius, D. and Y. Sun (2018). “A Survey of Knowledge Representation and Retrieval for Learning in Service Robotics”. In: *CoRR* abs/1807.02192. arXiv: 1807.02192 (cit. on p. 35).

Pednault, E. P. D. (1989). “ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus”. In: *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR’89). Toronto, Canada, May 15-18 1989*. Ed. by R. J. Brachman, H. J. Levesque, and R. Reiter. Morgan Kaufmann, pp. 324–332 (cit. on p. 15).

Poli, R. and L. Obrst (2010). “The interplay between ontology as categorial analysis and ontology as technology”. In: *Theory and applications of ontology: Computer applications*. Springer, pp. 1–26 (cit. on p. 34).

Pollack, M. E., L. E. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos (2003). “Autominder: an intelligent cognitive orthotic system for people with memory impairment”. In: *Robotics and Autonomous Systems* 44.3-4, pp. 273–282 (cit. on pp. 232, 235).

Powell, J., M. Molineaux, and D. W. Aha (2011). “Active and interactive learning of goal selection knowledge”. In: *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference* (cit. on pp. 26, 27, 89).

Quinde, M., J. G. Giménez-Manuel, C. L. Oguego, and J. C. Augusto (2020). “Achieving Multi-User Capabilities through an Indoor Positioning System based on BLE Beacons”. In: *16th International Conference on Intelligent Environments, IE 2020, Madrid, Spain, July 20-23, 2020*. IEEE, pp. 13–20 (cit. on p. 263).

Rabin, B. S. (2002). “Understanding How Stress Affects the Physical Body”. In: *The Link Between Religion and Health: Psychoneuroimmunology and the Faith Factor*. Ed. by H. G. Koenig and H. J. Cohen. USA: Oxford University Press, pp. 43–68 (cit. on p. 231).

Raghavan, V. V. and S. K. M. Wong (1986). “A critical analysis of vector space model for information retrieval”. In: *Journal of the American Society for Information Science (JASIS)* 37.5, pp. 279–287 (cit. on p. 109).

Ramoly, N., A. Bouzeghoub, and B. Finance (2018). “A Framework for Service Robots in Smart Home: An Efficient Solution for Domestic Healthcare”. In: *Innovation and Research in BioMedical engineering* 39.6. JETSAN, pp. 413–420 (cit. on pp. 232, 234).

Rao, A. S. and M. P. Georgeff (1995). “BDI Agents: From Theory to Practice”. In: *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*. Ed. by V. R. Lesser and L. Gasser. The MIT Press, pp. 312–319 (cit. on pp. 25, 26, 89).

Rathore, M. M., A. Ahmad, A. Paul, and S. Rho (2016). “Urban planning and building smart cities based on the Internet of Things using Big Data analytics”. In: *Comput. Networks* 101, pp. 63–80 (cit. on p. 61).

Refanidis, I., C. Emmanouilidis, I. Sakellariou, A. Alexiadis, R. Koutsiamanis, K. Agnantis, A. Tasidou, F. Kokkoras, and P. S. Efraimidis (2014). “myVisitPlanner ^{GR}: Personalized Itinerary Planning System for Tourism”. In: *Artificial Intelligence: Methods and Applications - 8th Hellenic Conference on AI, SETN 2014, Ioannina, Greece, May 15-17, 2014. Proceedings*. Ed. by A. Likas, K. Blekas, and D. Kalles. Vol. 8445. Lecture Notes in Computer Science. Springer, pp. 615–629 (cit. on pp. 53, 62).

Reiterer, B. and M. W. Hofbaur (2017). “Opportunistic Planning with Recovery for Robot Safety”. In: *KI 2017: Advances in Artificial Intelligence - 40th Annual German Conference on AI, Dortmund, Germany, September 25-29, 2017, Proceedings*. Ed. by G. Kern-Isberner, J. Fürnkranz, and M. Thimm.

Vol. 10505. Lecture Notes in Computer Science. Springer, pp. 352–358 (cit. on pp. 26, 30).

Richter, S. and M. Westphal (2010). “The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks”. In: *JAIR* 39, pp. 127–177 (cit. on pp. 21, 152).

Rjab, A. B. and S. Mellouli (2019). “Artificial Intelligence in Smart Cities: Systematic Literature Network Analysis”. In: *ICEGOV 2019: 12th International Conference on Theory and Practice of Electronic Governance, Melbourne, VIC, Australia, 3-5 April, 2019*. Ed. by S. I. B. Dhaou, L. D. Carter, and M. A. Gregory. ACM, pp. 259–269 (cit. on p. 62).

Rocco, M. D., F. Pecora, S. Sathyakeerthy, J. Grosinger, A. Saffiotti, M. Bonaccorsi, R. Limosani, A. Manzi, F. Cavallo, P. Dario, and G. Teti (2014). “A Planner for Ambient Assisted Living: From High-Level Reasoning to Low-Level Robot Execution and Back”. In: *2014 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 24-26, 2014*. AAAI Press (cit. on p. 232).

Rodríguez, S., V. Julián, J. Bajo, C. Carrascosa, V. J. Botti, and J. M. Corchado (2011). “Agent-based virtual organization architecture”. In: *Engineering Applications of Artificial Intelligence* 24.5, pp. 895–910 (cit. on p. 120).

Russell, S. J. and P. Norvig (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)* Pearson Education (cit. on p. 120).

Russell, S. J. and P. Norvig (2020). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson (cit. on pp. 9, 24, 52, 54, 88).

Sánchez, D., M. Batet, S. Martínez, and J. Domingo-Ferrer (2015). “Semantic variance: An intuitive measure for ontology accuracy evaluation”. In: *Engineering Applications of Artificial Intelligence* 39, pp. 89–99 (cit. on p. 111).

Santiago, F. J. M., F. J. Ariza-López, A. Montejo-Ráez, and L. A. U. López (2012). “GeOasis: A knowledge-based geo-referenced tourist assistant”. In: *Expert Systems with Applications* 39.14, pp. 11737–11745 (cit. on pp. 53, 62).

Sapena, O., E. Marzal, and E. Onaindia (2018). *TFLAP: a temporal forward partial-order planner*. Tech. rep. 2018 International Planning Competition, Temporal Tack (cit. on pp. 122, 145).

Sapena, O. and E. Onaindia (2002). “Domain-Independent Online Planning for STRIPS Domains”. In: *Advances in Artificial Intelligence - IBERAMIA 2002, 8th Ibero-American Conference on AI, Seville, Spain, November 12-15, 2002, Proceedings*. Ed. by F. J. Garijo, J. C. R. Santos, and M. Toro. Vol. 2527. Lecture Notes in Computer Science. Springer, pp. 825–834 (cit. on p. 54).

Sapena, O. and E. Onaindia (2008). “Planning in highly dynamic environments: an anytime approach for planning under time constraints”. In: *Applied Intelligence* 29.1, pp. 90–109 (cit. on pp. 54, 124).

Sapena, O., E. Onaindia, and A. Torreño (2015). “FLAP: Applying least-commitment in forward-chaining planning”. In: *AI Commun.* 28.1, pp. 5–20 (cit. on p. 21).

Schmidt, P., A. Reiss, R. Dürichen, C. Marberger, and K. V. Laerhoven (2018). “Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection”. In: *Proceedings of the 2018 on International Conference on*

Multimodal Interaction, ICMI 2018, Boulder, CO, USA, October 16-20, 2018. Ed. by S. K. D'Mello, P. G. Georgiou, S. Scherer, E. M. Provost, M. Soleymani, and M. Worsley. ACM, pp. 400–408 (cit. on pp. 231, 244).

Sebastia, L., I. Garcia, E. Onaindia, and C. Guzman (2009). "*E-Tourism*: a Tourist Recommendation and Planning Application". In: *International Journal on Artificial Intelligence Tools* 18.5, pp. 717–738 (cit. on pp. 53, 62).

Shim, J. and R. C. Arkin (2012). "Biologically-Inspired Deceptive Behavior for a Robot". In: *From Animals to Animats 12 - 12th International Conference on Simulation of Adaptive Behavior, SAB 2012, Odense, Denmark, August 27-30, 2012. Proceedings*. Ed. by T. Ziemke, C. Balkenius, and J. Hallam. Vol. 7426. Lecture Notes in Computer Science. Springer, pp. 401–411 (cit. on p. 55).

Simon, H. A. (1964). "On the Concept of Organizational Goal". In: *Administrative Science Quarterly* 9.1, pp. 1–22 (cit. on p. 118).

Spaan, M. T. J., T. S. Veiga, and P. U. Lima (2015). "Decision-theoretic planning under uncertainty with information rewards for active cooperative perception". In: *Autonomous Agents and Multi-Agent Systems* 29.6, pp. 1157–1185 (cit. on p. 120).

Speer, R., J. Chin, and C. Havasi (2017). "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by S. Singh and S. Markovitch. AAAI Press, pp. 4444–4451 (cit. on p. 104).

Srivastava, B., T. A. Nguyen, A. Gerevini, S. Kambhampati, M. B. Do, and I. Serina (2007). “Domain Independent Approaches for Finding Diverse Plans”. In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India*, pp. 2016–2022 (cit. on p. 129).

Talamadupula, K., J. Benton, S. Kambhampati, P. W. Schermerhorn, and M. Scheutz (2010). “Planning for human-robot teaming in open worlds”. In: *ACM Trans. Intell. Syst. Technol.* 1.2, 14:1–14:24 (cit. on pp. 23, 26, 29, 89).

Talamadupula, K., D. E. Smith, W. Cushing, and S. Kambhampati (2013). *A theory of intra-agent replanning*. Tech. rep. Arizona State Univ Tempe Dept of Computer Science and Engineering (cit. on pp. 66, 121, 128, 135, 174).

Tate, A. (1976). *Project planning using a hierarchic non-linear planner*. Department of Artificial Intelligence, Research Report 25, University of Edinburgh (cit. on p. 22).

Teixeira, M. S., V. Maran, and M. Dragoni (2021). “The interplay of a conversational ontology and AI planning for health dialogue management”. In: *SAC ’21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021*. Ed. by C. Hung, J. Hong, A. Bechini, and E. Song. ACM, pp. 611–619 (cit. on pp. 38, 39).

Thosar, M., S. Zug, A. M. Skaria, and A. Jain (2018). “A Review of Knowledge Bases for Service Robots in Household Environments”. In: *Proceedings of the 6th International Workshop on Artificial Intelligence and Cognition, Palermo, Italy, July 2-4, 2018*. Ed. by A. Chella, I. Infantino, and A. Lieto. Vol. 2418. CEUR Workshop Proceedings. CEUR-WS.org, pp. 98–110 (cit. on p. 35).

United Nations Department of Economic and Social Affairs (2020). *World Population Ageing 2019*. United Nations (cit. on p. 231).

Vachtsevanou, D., P. Junker, A. Ciortea, I. Mizutani, and S. Mayer (2020). “Long-Lived Agents on the Web: Continuous Acquisition of Behaviors in Hypermedia Environments”. In: *Companion of The 2020 Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*. Ed. by A. E. F. Seghrouchni, G. Sukthankar, T. Liu, and M. van Steen. ACM / IW3C2, pp. 185–189 (cit. on p. 186).

Vansteenwegen, P., W. Souffriau, G. V. Berghe, and D. V. Oudheusden (2011). “The City Trip Planner: An expert system for tourists”. In: *Expert Systems with Applications* 38.6, pp. 6540–6546 (cit. on pp. 53, 62).

Vaquero, T. S., J. R. Silva, and J. C. Beck (2013). “Post-design analysis for building and refining AI planning systems”. In: *Engineering Applications of Artificial Intelligence* 26.8, pp. 1967–1979 (cit. on p. 88).

Vattam, S., M. Klenk, M. Molineaux, and D. W. Aha (2013). *Breadth of approaches to goal reasoning: A research survey*. Tech. rep. Naval Research Lab Washington DC (cit. on p. 24).

Vendrell, E., M. Mellado, and A. Crespo (2001). “Robot planning and re-planning using decomposition, abstraction, deduction, and prediction”. In: *Engineering Applications of Artificial Intelligence* 14.4, pp. 505–518 (cit. on p. 120).

Vermesan, O. and P. Friess (2013). *Internet of things: converging technologies for smart environments and integrated ecosystems*. River publishers (cit. on p. 61).

Ward, R. R. (1972). *The Living Clocks*. A Mentor book. Collins (cit. on p. 118).

Warfield, I., C. Hogg, S. Lee-Urban, and H. Muñoz-Avila (2007). “Adaptation of Hierarchical Task Network Plans”. In: *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference*. Ed. by D. Wilson and G. Sutcliffe. Key West, Florida, USA: AAAI Press, pp. 429–434 (cit. on p. 125).

Weld, D. S. (1994). “An Introduction to Least Commitment Planning”. In: *AI Mag.* 15.4, pp. 27–61 (cit. on p. 21).

Wilson, M. A., M. Molineaux, and D. W. Aha (2013). “Domain-Independent Heuristics for Goal Formulation”. In: *Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2013, St. Pete Beach, Florida, USA, May 22-24, 2013*. Ed. by C. Boonthum-Denecke and G. M. Youngblood. AAAI Press (cit. on pp. 26, 27, 89).

Winkler, W. E. (1990). “String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage”. In: *Proceedings of the Section on Survey Research Methods*, pp. 354–359 (cit. on p. 109).

Wood, A., G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic (2006). “ALARM-NET: Wireless sensor networks for assisted-living and residential monitoring”. In: *University of Virginia Computer Science Department Technical Report CS-2006-11* 2, p. 17 (cit. on p. 233).

Yehuda, R. (2002). “Post-Traumatic Stress Disorder”. In: *New England Journal of Medicine* 346.2. PMID: 11784878, pp. 108–114 (cit. on p. 236).

Zlotkin, G. and J. S. Rosenschein (1989). “Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains”. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*. Ed. by N. S. Sridharan. Morgan Kaufmann, pp. 912–917 (cit. on p. 55).

Zweben, M., E. Davis, B. Daun, and M. J. Deale (1993). “Scheduling and rescheduling with iterative repair”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 23.6, pp. 1588–1596 (cit. on p. 125).

Acronyms

AI Artificial Intelligence

AP Automated Planning

KR Knowledge Representation

IPC International Planning Competition

HTN Hierarchical Task Network

HTN Hierarchical Task Networks

PDDL Planning Domain Definition Language

STRIPS the input language to the Stanford Research Institute Problem Solver

TILs timed initial literals

TIL timed initial literal

POP Partial-Order Planning

ICAPS International Conference on Automated Planning and Scheduling

DL Description Logic

RDF Resource Description Framework

OWL	Web Ontology Language
W3C	World Wide Web Consortium
IoT	Internet of Things
GPS	The Global Positioning System
GLASS	Goal-management for Long-term Autonomy in Smart citieS
RS	Recommender System
AmI	Ambient Intelligence
AL	Assisted Living