ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Logic Reasoning in BDI Agents: Current Trends and Spatial Integrations

Tesi di laurea in
Sistemi Autonomi

Relatore
Prof. Andrea Omicini

Candidato
Maicol Forti

Correlatore
Prof.ssa Roberta Calegari
Dott. Giovanni Ciatto

Anno Accademico 2019-2020

Keywords

MAS
BDI
Spatial Reasoning
2P-Kt
Geo2p

Abstract

This thesis finds its place in the context of BDI agents and aims to enable a form of situated spatial reasoning. A survey is proposed in which the possible techniques and technologies that can be integrated into the BDI model to provide a form of spatial reasoning are analyzed. This review highlights a technological gap that we have therefore decided to fill, with the goal of providing a way to locate logical information in certain spatial areas and to be able to constrain reasoning on them. In this thesis we propose *Geo2p*, a technological prototype based on *2P-Kt* that allows you to query situated information in tuProlog, enabling a form of spatial reasoning: given a region of space where certain Clauses are valid, a Theory can be defined, constraining the knowledge on what is true in the selected area.

To my family
who gave me the opportunity to choose my own path.

Acknowledgements

I would like to thank Prof. Andrea Omicini for his teachings and for giving me the opportunity to conduct this thesis. I would like to express my gratitude to Prof.ssa Roberta Calegari and Dott. Giovanni Ciatto for having patiently supervised my work and for giving me support. Moreover I would like to show my sincere respect to all the Professors who have passed on their passion to me over the years.

Finally, I would like to express my gratitude to my family, who have always supported me and without whom I would not have achieved these goals.

Contents

List of Figures

Chapter 1

Introduction

Agent-based systems technology has generated lots of excitement in recent years, because of its promise as a new paradigm for conceptualizing, designing and implementing software system, particularly in distributed and open environments such as the internet [85].

A multi-agent-system can be conceived as a group of agents that interact with each other and with the environment, establishing a society. Agents are the entities of the system, they are autonomous, social and immersed in a society, because autonomy does not make sense in isolation, and the environment is the container in which agents are immersed.

Humans tend to interpret the behaviour of an entity in terms of mental properties, as if it were rational. According to this level of abstraction a new successful framework for agent technology is that of Rao and Georgeff [72], the BDI framework. The agents of the BDI framework are called BDI agents, as the notions of Belief, Desire and Intention are central. BDI technologies are a research topic of particular relevance, and the possibility of integrating them with other AI techniques opens up a world of possibilities.

A survey is proposed in which the possible techniques and technologies that can be integrated into the BDI model to provide spatial reasoning are analyzed. This review highlights a technological gap that we have therefore decided to fill. The type of reasoning we want to deliver consists of providing a way to locate logical information in certain spatial areas and to be able to constrain reasoning on them. In this way it is possible to enable a form of situated spatial reasoning. This goal is supported by two technologies: *Tile38*, one of the most famous geospatial database, and *2P-Kt*, a multi-paradigm logic programming framework written in Java.

In this thesis we propose *Geo2p*, a technological prototype based on *2P-Kt* that allows you to query situated information in tuProlog, enabling a form of spatial reasoning: given a region of space where certain Clauses are valid, a Theory can

be defined, constraining the knowledge on what is true in the selected area.

Thesis Structure. This thesis is structured as follows. Chapter 2 provides the background needed to understand the topics of the following chapters, by making available the necessary information and presenting some related works. Chapter 3 presents an SLR conducted on the topic of BDI and AI tech integrations. Chapter 4 presents *Geo2p* and explains the design choices made for integrating a form of spatial reasoning in 2P-Kt. The implementation is presented in Chapter 5. Finally, Chapter 6 concludes this thesis by summarising its main contribution and possible future works.

Chapter 2

State of the Art

2.1 Multi-Agent-Systems

2.1.1 Multi-Agent-Systems

Agent-based systems technology has generated lots of excitement in recent years because of its promise as a new paradigm for conceptualizing, designing and implementing software systems [85]. This promise is particularly attractive for creating software that operates in environments that are distributed and open, such as the internet [85].

Agents, society and the environment, are the basic design abstractions of Multi-Agent-Systems (MAS), which can be conceived as a group of agents that interact with each other and with the environment establishing a society.

Even though somehow blurred throughout the vast literature on multiagent systems, the notion of agent can be characterised by few fundamental key-points [26]: (*i*) autonomy, (*ii*) interaction, and (*iii*) task. In other words, an agent may be thought as an *autonomous* software component which *interacts* with its environment in order to achieve its *tasks* [26].

Managing the interaction is the goal of coordination: the space of agent interaction is no longer to be seen as merely the space of communication, but also the space of coordination [26]. Coordination languages are meant to express the agent's observable behaviour and to design its interaction protocol; coordination models allow the interaction space to be shaped and ruled, while coordination architectures provide for patterns for the organisation of agent ensembles [26].

2.1.2 BDI

The model from which the BDI agents take inspiration is the human one. Humans tend to interpret the behaviour of an entity in terms of mental properties, as if it

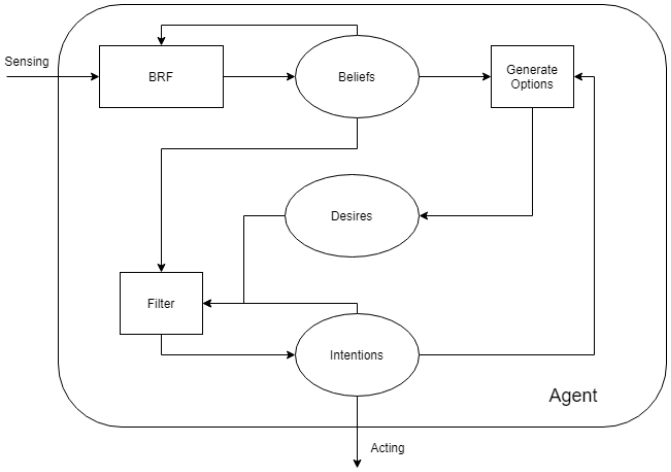

Figure 2.1: Basic architecture of a BDI agent.

were rational, this level of abstraction is defined as intentional stance.

In accordance with [28], one of the most popular and successful framework for Agent technology is that of Rao and Georgeff [72].

The agents of the BDI framework are called BDI agents, as the notions of Belief, Desire and Intention are central.

Beliefs represent the agent's current knowledge about the world, including information about the current state of the environment inferred from perceptions devices and messages from other agents, as well as internal information [28]. Desires represent a state which the agent is trying to achieve. Intentions are the chosen means to achieve the agent's desires, and are generally implemented as plans and post-conditions [28].

An example of a BDI architecture is shown in figure 2.1.

2.2 Spatial Reasoning

The way in which we represent space mutually affects the way in which we can reason about it. It may be either qualitative or quantitative, but is mostly approximated [33].

Mathematics and Logic can be exploited as tools to represent, analyze and reason on space, allowing to compute on it and on its organization, on different levels of efficiency.

Some different approaches to the representation of space and reasoning on space are presented next in brief, taking from [62].

2.2.1 Geometry

Geometry is an approach to the representation of space that abstracts from our perception of reality. For instance, we can use Euclidean Geometry to represent space and reason about it through axioms, theorems and proofs.

A further step can be taken from non-Euclidean geometry, with the aim of representing the *true* physical space that goes beyond our direct sensory and cognitive perceptions.

2.2.2 Logic

Modal logic have proved to be particularly interesting, as they are very specific and have a computational behaviour that is often decidable, they can be used to model, analyse and reason about space.

The modal operators are reinterpreted:

- $\Box$ (necessarily) as the interior operator.
- $\diamond$ (possibly) as the closure operator of a topological space.

Another example is S4 [19], sound and complete with respect to topological semantics [56] and represents the modal logic of any Euclidean space.

2.2.3 Mathematical Morphology

Mathematical morphology (MM) [2] analyses shape, spatial information and image processing. It comes from the similarity between the algebraic properties of MM operators and of modal operators.

It is efficient for spatial reasoning, for instance to guide the exploration of space, in a focus of attention process, and for recognition and interpretation of tasks.

2.3 Prolog

Prolog is a logic programming language that has its roots in first-order logic.

When stripped to the bare essentials, logic programming can be summarized by the following three features [3]:

- computing takes place over the domain of all *terms* defined over a "universal" alphabet.
- values are assigned to variables by means of automatically generated substitutions, called *most general unifiers*.

- the control is provided by a single mechanism: automatic *backtracking*.

Logic programming in its pure form supports declarative programming. A declarative program admits two interpretations [3]:

- The first one, called a procedural interpretation, explains how the computation takes place.
- The second one, called a declarative interpretation, is concerned with the question what is being computed.

Programs become executable specifications and the programmer will have to worry about what will be computed, while delegating how to the underlying machine.

Another important feature of logic programming is that it supports interactive programming. That is, the user can write a single program and interact with it by means of various queries of interest to which answers are produced [3].

The Prolog systems greatly support such an interaction and provide simple means to compute one or more solutions to the submitted query, to submit another query, and to trace the execution by setting up, if desired, various check points, all within the same "interaction loop", leading to a flexible style of programming [3].

2.3.1 tuProlog

The complexity of Internet-based system engineering calls for suitable infrastructures, meant to make the designers' and developers' task easier by providing commonly-required services to applications [31].

Easily deployable infrastructures are needed, which can (*i*) be easily configured to match the application needs, both statistically and dynamically, (*ii*) rule component and application interaction, and possibly (*iii*) encapsulate some form of intelligence to be exploited by applications [31]. In this scenario, where Software Engineering, Programming Languages, and (Distributed) Artificial Intelligence meet, logic-based languages are fighting to find a role to play [31].

tuProlog is a Java-Based Prolog designed to build Internet-based intelligent components, which are (*i*) easily deployable, (*ii*) lightweight, (*iii*) scalable, (*iv*) statically and dynamically configurable, and (*v*) interoperable [31].

tuProlog makes a core Prolog inferential engine available as a Java class, so that an unlimited number of tuProlog engines can be exploited at the same time by the same application or process [31].

Each engine can be configured independently, and integrated into a system according to the preferred/required interaction pattern: as a Java object, a Java

bean, via RMI or CORBA, or as an Internet Service [31]. Moreover, tuProlog integrates basic coordination capabilities, by providing logic tuple spaces as a coordination media. This makes tuProlog a good choice as an establishing technology for flexible and effective Internet infrastructures [31].

tuProlog natively supports multi-paradigm programming, so as to provide an integration model between Prolog and other object-oriented languages.

2.4 2P-Kt

2P-Kt is the natural evolution and modernisation of Prolog implementation [78], Kotlin-based, and provides a Prolog framework that natively supports multi-paradigm programming.

It aims at becoming an open ecosystem for Symbolic Artificial Intelligence (AI). For this reason, 2P-Kt consists of a number of incrementally interdependent modules aimed at supporting symbolic manipulation and reasoning in an extensible and flexible way [1].

2P-Kt currently focuses on supporting knowledge representation and automatic reasoning through logic programming, by featuring [1]:

- a module for logic terms and clauses representation, namely `core`
- a module for logic unification representation, namely `unify`
- a module for in-memory indexing and storing logic theories, as well as other sorts of collections of logic clauses, namely `theory`
- a module providing ISO Prolog resolution of logic queries, namely `solve`, coming with two implementations (i.e. `solve-classic` and `solve-streams`)
- a number of modules (i.e., the many `dsl-*` modules) supporting a Prolog-like, Domain Specific Language (DSL) aimed at bridging the logic programming with the Kotlin object-oriented & functional environment
- two parsing modules: one aimed at parsing terms, namely `parser-core`, and the other aimed at parsing theories, namely `parser-theory`
- two serialisation-related modules: one aimed at (de)serialising terms and clauses, namely, `serialize-core`, and the other aimed at (de)serialising terms theories, namely `serialize-theory`
- a module for using Prolog via a command-line interface, namely `repl`
- a module for using Prolog via a graphical user interface (GUI), namely `ide`

The modular, unopinionated architecture of 2P-Kt is deliberately aimed at supporting and encouraging extensions towards other sorts of symbolic AI systems than Prolog. Furthermore, 2P-Kt is developed as in *pure*, multi-platform Kotlin project, which brings two immediate advantages [1]:

1. it virtually supports several platforms, there including JVM, JS, Android, and Native (even if, currently, only JVM, JS and Android are supported),

2. it consists of a very minimal and lightweight library, only leveraging on the Kotlin *common* library, as it cannot commit to any particular platform standard library.

2P-Kt can either be used as a command-line program or as a Kotlin, JVM, Android or JS library.

2.4.1 Application scenarios

As explained in [78], 2P-Kt has been thought to explicitly address two main use scenarios: (*i*) its use by software programmers as a library to exploit the logic programming paradigm and (*ii*) its use as a basic component to be further extended by adding new Prolog libraries, primitives and so on. The first scenario is intended for "final users" that would like to inject in their software some logic programming. The secondary usage is as a code base to implement Prolog extensions, deviating from the Prolog Standard functionalities.

2.4.2 The motivation behind 2P-Kt

In this thesis we realized that the main focus was on how to manage logical information and consequently locate it spatially. What we needed then was to be able to take and manipulate logical knowledge directly. 2P-Kt allows us to do this, unlike for example Prolog or Jason, which must be taken in their entirety, as its modular structure allows us to exploit the concepts that were of interest to us, such as that of Theory, individually.

2.5 Tile38

In order to enable situated reasoning, in this thesis it was decided to adopt Tile38. Tile38 is an ultra fast geospatial database and geofencing server for location-based applications. It is open source (MIT licensed) and supports in-memory geolocation data store, spatial index, and realtime geofence [7].

Tile38 was adopted as it provides simple commands, which can be exploited directly to solve the issues posed by this thesis, such as geolocalizing data, recovering data belonging to a certain region of space and removing data from space. It is also open source, and can be supported by several client libraries, including one in Java already available, which will be exploited.

Among its characteristics we find:

- spatial index with search methods such as Nearby, Within and Intersects.
- support to different Object types, such as lat/lon, GeoJSON and XYZ tile.
- support for lots of Clients Libraries written in many different languages.
- server response in RESP or JSON.
- in-memory database that persists on disk.
- it also supports realtime geofencing through webhooks or publish subscribe channels, where a geofence is a virtual boundary that can detect when an object enters or exits the area.

2.5.1 Installation

Tile38 can be easily installed and run with Docker [57], with the following commands:

```
1 docker pull tile38/tile38
2 docker run -p 9851:9851 tile38/tile38
```

2.5.2 Object Types

Tile38 provides a series of object types that can be stored in a collection, with an exception to XYZ Tiles and QuadKeys that are reserved for the SEARCH keyword only.

The most basic one is a point, composed of a latitude and a longitude. An optional `z` may be used for auxiliary data such as elevation or a timestamp.

Another important type is the bounding box, which consists of two points: the first being the southwestern most point and the second is the northeastern most point.

It also supports Geohash, a string representation of a point, and GeoJSON, an industry standard format for representing a variety of object types, such as a point, multipoint, linestring and polygon. All coordinates are in Longitude, Latitude order.

2.5.3 Commands

Within the list of commands that Tile38 makes available, some examples are reported below:

```
SET key id [FIELD name value ...] [EX seconds] [NX|XX]
   (OBJECT geojson)|(POINT lat lon z)|(BOUNDS minlat
   minlon maxlat maxlon)|(HASH geohash)|(STRING value)
```

set the value of an id, if already associated it'll be overwritten,

```
GET key id [WITHFIELDS] [OBJECT|POINT|BOUNDS|(HASH
   geohash)]
```

get the object of an id, GeoJSON is the default output,

```
NEARBY key [CURSOR start] [LIMIT count] [SPARSE spread]
   [MATCH pattern] [DISTANCE] [WHERE field min max ...]
   [WHEREIN field count value [value...] ...] [WHEREEVAL
    script numargs arg [arg...] ...] [WHEREEVALSHA sha1
   numargs arg [arg...] ...] [NOFIELDS] [FENCE] [DETECT
   what] [COMMANDS which] [COUNT|IDS|OBJECTS|POINTS|
   BOUNDS|(HASHES precision)] (POINT lat lon meters)|(
   ROAM key pattern meters)
```

searches a collection for objects that are close to a specified point,

```
DROP key
```

remove all objects from specified key.

2.5.4 Client Libraries

Tile38 supports HTTP and telnet options, but it is recommended to use a client library or the Tile38 CLI. Tile38 uses the Redis RESP protocol natively, therefore all clients that support basic Redis commands will in turn support Tile38.

2.6 Lettuce

Among all the popular clients, for this thesis we opted for Lettuce [50], a scalable and thread-safe Redis client for synchronous, asynchronous and reactive usage, in Java.

Lettuce enables a connection to Tile38,

```
val client = RedisClient.create("redis://localhost
   :9851")
        val connection = client.connect()
        val sync = connection.sync()
        val codec = StringCodec.UTF8
```

on which it is possible to send commands, an example:

```
      sync.dispatch(
            CommandType.SET,
            StatusOutput(codec),
            CommandArgs(codec).addValues(
                "fleet",
                "truck1",
                "OBJECT",
                "{\"type\": \"Point\", \"coordinates\":
                   [-112.2693, 33.5123]}"
            )
        )
```

The command *set* sets the value of a key. The value in the example is an an object of type Point with coordinates in longitude, latitude order.

2.7 Related Works

Some related works are discussed in this section, in which the BDI model is integrated with spatial concepts.

2.7.1 Game Engines e MAS: Spatial Tuples in Unity3D

Game Engines and MAS: BDI & Artifacts in Unity

The purpose of the work of [71] was to propose a different, more expressive and intuitive design approach for the definition of AI based on the notion of autonomous agents in a simulated environment, powered by the Unity game engine.

The resulting language is Prolog based and Jason [16] inspired and provides high-level declarative behaviours for autonomous agents. The behaviour is definable through a Prolog file.

Acknowledging similarities between logical reasoning and agent's cycle of reasoning, the author decided to use a logical paradigm as foundation.

Spatial Tuples in Unity3D

Spatial Tuples [74] is an extension of the basic tuple-based model for distributed multi-agent-system where (a) tuples are conceptually placed in regions of the physical world and possibly move anchored to a mobile computational device, (b) the behaviour of standard Linda coordination primitives is extended so as to depend on the spatial properties of the coordinating agents, tuples, and the topology of space, and (c) the tuple space can be conceived as a virtual layer augmenting physical reality.

The master thesis [6] provides a model for Spatial Tuples in Unity3D that exploits the API produced in [71] and [23], allowing the programmer to use the Spatial Tuples model when programming a BDI Agent, using the primitives of Spatial Tuples directly inside the plan of an agent written in Prolog.

2.7.2 Introducing a novel model of belief-desire–intention agent for urban land use planning

The goal in the works [11] and [12] from the same authors, is to develop a BDI agent based model to handle spatial issues. The fundamental concepts of practical reasoning architecture such as belief, desire, intention, along with commitment ad interaction, have been combined with analyses and applications of GIS. In [11] a Desktop GIAgent software is introduced, with the advantage of using agents for spatial analysis. In [12] particular attention is given to land use planning defined by three main components: land areas, goals and actions, translated to a spatial BDI agent architecture.

2.7.3 A multi-agent architecture for geosimulation of moving agents

In [89] a novel architecture is proposed in which an axiomatic derivation system in the form of first-order logic facilitates declarative explanation and spatial reasoning. The objective is to describe and develop an architecture that combines a multi-agent system with GIS, logical deduction and qualitative reasoning. The systems integrates multiple moving agents and the concept of means-ends spatio-motional reasoning. The paper provides details about the architecture and simulation system for moving reasoner agents (SISMORA).

2.7.4 Using conceptual spaces for belief update in multi-agent systems

The work presented in [18] comprises the design and implementation of an integration of a conceptual-space level into the BDI agent architecture, with the aim of helping people who are blind or visually impaired to know where they are, where they want to go and how to get there safely and independently. This integration is developed on top of the resources of the Jason platform and the CSML API.

2.7.5 Solace a multi-agent model of human behaviour

The paper [10] presents SOLACE (SOciaL Attachment Crisis Evacuation), a multi-agent model of human behaviour during seismic crisis based on social attachment theory, where real geographic data are used to define the spatial context of the crisis environment, delimit mobility with barriers and constrain movement to free space. The novelty of this model must be sought in the integration of social attachment and GIS data. The model was implemented using the GAMA platform using a BDI approach.

Chapter 3

BDI & AI techniques integration: a Systematic LR

A research topic of particular relevance is that which concerns BDI (Belief-Desire-Intention) technologies. The BDI model is a well-known software programming model for programming intelligent agents. However, this model has some limitations: for example the lack of learning techniques, the fact that it does not take into account the emotional state of the agents and it does not involve spatial reasoning. These limitations can be addressed by integrating other techniques and technologies. This chapter summarizes a Systematic Literature Review that I conducted on this subject.

3.1 Systematic Literature Review

The notion of systematic literature reviews (SLR) basically develops in the healthcare domain [60], where the notion of meta-analysis gets early relevance [48, 64], giving rise to the need of a well-founded methodological approach to literature results.

A systematic review attempts to identify, appraise and synthesise all the empirical evidence that meets pre-specified eligibility criteria to answer a given research question. Researchers conducting systematic reviews use explicit methods aimed at minimising bias, in order to produce more reliable findings that can be used to inform decision making [39].

An SLR is divided in five stages:

- Research Questions and Goals: the goal of the research is defined and consequently the questions to be answered are determined.
- Search Strategy: it is necessary to motivate a search strategy that allows

to find as many pieces of scientific literature as possible, avoiding bias (the tendency of deviating from the standards, the use of subjectivity or preconceptions), and that can be reproducible.

- Study Selection: the criteria for exclusion and inclusion are defined on the basis of the research questions defined in the first phase.
- Quality Assessment: the primary studies that have been collected are evaluated.
- Data Extraction and Analysis: selected studies should be read and the information needed to answer research questions and draw conclusions should be extracted.

To summarize, a Systematic Literature Review (SLR) is a powerful technique that allows to collect and summarize the information in the scientific literature relating to a specific topic, using a rigorous and reproducible methodology.

3.2 A categorization of BDI integrations

This section summarizes the content of the SLR and is structured as follow: in 3.2.1 the goal of the SLR is presented, in 3.2.2 the method by which the SLR was performed is discussed, finally in 3.2.3 the results are shown.

3.2.1 Goal

This works aims to categorize the possible types of integration of AI techniques to the BDI model, in order to become a starting point for defining new research areas and future projects.

The research questions that have given rise to the need for this SLR are so defined:

- How can the BDI model be integrated with other AI techniques? (Goal)
- Which integration categories are the most widespread? (Research Question 1)
- Are there technologies to support these integrations or just the models are described? (Research Question 2)
- Are these technologies effectively used in real-world applications? (Research Question 3)

3.2.2 Method

The search process was conducted on some of the most relevant digital libraries: Google Scholar, IEEE Xplore, Springer Link, ACM Digital Library.

The following keywords and their alternatives were defined from the research questions:

- BDI, (belief-desire-intention, belief desire intention).
- Integration, (integrat*).
- AI, (artificial intelligence, artificial-intelligence).
- ML, (machine learning, learning*).
- Logic.

The sources were interrogated using predefined queries, obtained from the previously identified keywords:

- (model* OR technolog* OR technique* OR logic*) AND integrat* AND (BDI OR "belief-desire-intention" OR "belief desire intention") AND (architecture* OR logic* OR model* OR framework*)
- (BDI OR "belief desire intention" OR "belief-desire-intention") AND integrat* AND (AI OR "artificial intelligence" OR "artificial-intelligence" OR "machine learning" OR ML OR learning*)

With the exception of IEEE Xplore which supports fewer wildcards, the first query was limited as follows: (model* OR technolog* OR technique* OR logic*) AND integrat* AND (BDI OR "belief-desire-intention" OR "belief desire intention") AND (architecture OR logic* OR model* OR framework).

A total of 981 documents were collected and filtered with some constraints:

- Inclusion Criteria
  - Full Paper.
  - Sources discussing the topic of this research.
  - Sources answering the research questions.
- Exclusion criteria
  - Papers not discussing the subject of this SLR: the main check was carried out on the abstract.

- Papers not answering the research questions: the check is carried out on the full paper.
- Duplicate reports of the same study.
- Whole books.

Duplicates have been searched for and deleted, titles and abstracts were used as the first filter to check the correlation with the topic to be discussed and the complete papers have been reviewed in detail.

At the end of this selection process, the documents amount to 131. To obtain a more targeted selection, one more check on full paper was performed in the phase of quality assessment, with the result of filtering additional documents.

After the quality assessment phase, the number of remaining articles amounts to 72.

3.2.3 Results

The results of this SLR have been summarized in the tables 3.1, 3.2, 3.3. If two rows appear in correspondence with a category, you should read as follows: the upper row refers to the studies of the category, the lower one to the studies of other categories that also deal with that topic. For the lower ones, the technological correspondence is not evaluated because it has already been counted in the related category. In table 3.4 there is a legend of the terms used.

The table collects the integrations resulting from the SLR divided by category, and shows those for which a technology or a project is provided.

At the end of this research, what emerges is that there are various types of integrations applicable to the BDI model, but still few technology ready to use or projects available, especially with regards to real world applications.

There are related-works, for instance the work in [14] puts its focus on identifying a range of possible approaches to integrating AI into a BDI agent architecture by organizing the work in: AI in the sense phase, plan phase, act phase and discussing the architectural strategies for integrating AI in BDI agents.

Having said that, this work aims at a different and new objective, that of categorizing the types of integration in order to understand in what ways the BDI model can be augmented and in what fields it can be used. In this way, I believe this SLR could become the starting point for defining new research areas and future projects.

Analysis				
	N°	**Integration**	**Tech**	**Project**
Decision Making	[41][46][61] [88]	[41] = Coherence-driven adaptive [46] = Internal Performance [61] = Recognition-primed [88] = Inner decision	[41]	[41]
Emotions	[13][17][36] [70][84]	[13][17][36] = OCC [70] = Perception-Action Model [84] = Emotions and Engagement [35] = OCC [86] = Ellis's ABC Theory	[17][36]	[17]
	[35][86]			
Intention Selection	[15][90][91]	[15] = Design-To-Criteria Scheduler [90] = Plan coverage [91] = Enablement Checking, Low-coverage prioritisation	[15][91]	
Logic	[24][66][69] [83][93]	[24] = Coalition logic [66] = Propositional Dynamic Logic [69] = Many-sorted first-order logic [83] = Fuzzy logic [93] = Deontic epistemic action logic [44] = FODL, Priority Logic [68] = Extension of CTL*	[83]	
	[44][68]			
Machine Learning	[4][5][32] [59][82][87]	[4][87] = RL [5] = TDL, Q-learning, SARSA [32] = TDL, Q-learning [59] = LSTM [82] = Adaptive ML [34] = Linear Regression [54] = Intentional Learning, Manipulative Abduction, RL [81] = ACL, BUL	[4] [5] [32] [59] [87]	[4][87]
	[34][54][81]			

Table 3.1: Analysis of the results part 1

Analysis				
	N°	**Integration**	**Tech**	**Project**
Modeling	[43][20][55] [58][86][95]	[43] = CAB, FFM [20] = Linear Threshold [55] = Computational model of selective attention [58] = Biological Immune System [86] = Psychoterapeutic Model [95] = Petri Net	[43] [20] [86]	[20]
	[8][35][44] [69][80]	[8] = Event-based run-time model of institutions [35] = ReGreT [44] = BRS, ForTrust, ReGreT [69] = Repage [80] = Agent Based Model		
Planning	[30][34][54] [65][76][81] [94]	[30] = Case Based Planning [34] = Machine Learning [54] = Intentional Learning [65] = Specifications [76] = Genetic Algorithms [81] = Decision Tree [94] = First-Principles Planning	[34] [54] [65] [76] [94]	[65]
	[82]	[82] = Decision Tree		
Probabilistic	[9][25][29] [49][53][73] [79]	[9][73] = POMDPs [25] = Bayesian Networks, Influence Diagrams [29] = Bayesian Theories [49] = BBN, DFT, PDFS [53][79] = Bayesian Network	[9] [25] [49] [53] [79]	
	[30][36][63] [81]	[30] = Bayesian Networks [36] = Bayesian Decision Networks [63] = Bayesian Algorithm [81] = Probabilistic plan selection function		

Table 3.2: Analysis of the results part 2

Analysis				
	N°	**Integration**	**Tech**	**Project**
Reasoning	[8][35][42] [44][47][92]	[8] = Event-based run-time model of institutions [35] = Trust processes [42] = Organizational Reasoning [44] = Trust model [47] = NARS [92] = Case-based reasoning	[8] [42] [47] [92]	[42]
Semantic Web	[22][37][38] [40][51][52]	[22] = Linked Data [37] = Rule Interchange Format [38] = Description Logic [40] = OWL, RDF [51][52] = OWL-S	[22] [40] [52]	
Simulation	[63][80] [49]	[63] = High-level architecture [80] = Agent Based Simulation [49] = AnyLogic 6.0	[80]	[80]
Spatial Concept	[10][11][12] [18][89]	[10][11][12][89] = GIS [18] = CSML	[10][18]	
Extra	[21][27][67] [68][75][77]	[21] = Distributed Transactions [27] = Possibilistic framework [67] = Interactive Storytelling [68] = Policy [75] = Cloud computing [77] = Social Influence Theory	[27]	[27]

Table 3.3: Analysis of the results part 3

Legend	
OCC	Ortony, Clore, and Collins theory
FODL	First-Order Dynamic Logic
CTL	Computation Tree Logic
RL	Reinforcement Learning
TDL	Temporal Difference Learning
SARSA	State–Action–Reward–State–Action
LSTM	Long Short-Term Memory
ML	Machine Learning
ACL	Aggressive Concurrent Learning
BUL	Bottom-Up Learning
CAB	Culturally Affected Behavior
FFM	Five Factor Model
BRS	The beta reputation system
POMDPs	Partially Observable Markov Decision Processes
BBN	Bayesian Belief Network
DFT	Decision-Field-Theory
PDFS	Probabilistic Depth-First Search
NARS	Non-Axiomatic Reasoning System
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
RDF	Resource Description Framework
GIS	Geographic Information System
CSML	Conceptual space markup language

Table 3.4: Legend for tables 3.1, 3.2, 3.3

3.3 Thesis Direction

Among the categories identified in the SLR, some have less results than others and a lack of technology. Among these, the category of our interest is that linked to spatial concepts.

Intelligent agents are possibly the most suitable vessel for spatial reasoning [45], including languages for spatial representation and logic for spatial reasoning.

Cognitive abilities of intelligent agents can be in principle exploited for spatial reasoning [33], for instance the ability to separately handle and properly use epistemic knowledge is an obvious benefit when dealing with spatial information.

Diverse logic can be embedded into an agent architecture, so as to provide for different ways to reason about space.

Despite this, few works have been found in this SLR that attempt to integrate

spatial reasoning into the BDI model: this works are summarized in the related works section 2.7, with the exception of 2.7.1 which comes from a different study.

We decided therefore to fill this technological gap highlighted in this survey. The type of reasoning we want to deliver consists of providing a way to locate logical information in certain spatial areas and to be able to constrain reasoning on them. In this way it is possible to enable a form of situated spatial reasoning. This goal is supported by two technologies: *Tile38*, one of the most famous geospatial database, and *2P-Kt*, a multi-paradigm logic programming framework written in Java.

We propose *Geo2p*, a technological prototype based on *2P-Kt* that allows to query for situated information in tuProlog, enabling a form of spatial reasoning.

Chapter 4

Geo2p Design

This chapter presents *Geo2p* and explores the choices that have been made during this thesis.

The scenario is first defined in section 4.1, the organization is presented in 4.2 and then each further section discusses a different module.

4.1 Scenario

The main goal of this thesis is to define a form of spatial reasoning where logical information can be located in certain spatial areas and to be able to constrain reasoning on them.

In particular, we provide a way that will allow to query for situated information in tuProlog. Given a certain region of space where certain clauses are valid, the current theory will be constrained on the basis of what is true in that area.

It should be possible to geolocate the clauses of a Prolog program within a certain region of space. In this way, different clauses can be placed in different regions of space.

Given a region of space, one must be able to load the clauses that have been geolocated within that region. In such a manner, knowledge can be constrained to what is true in the selected region of space.

Finally, we must be able to remove selected clauses from a region of space.

It will be necessary to understand how to seamlessly integrate spatial concepts within 2P-Kt, so as not to have to modify what already exists.

4.2 Project Organization

The project is divided into two sub-projects, one dedicated to the Prolog theory (`theory-geo`) and the other to the Prolog solver (`solve-geo`). Both are structured

in modules, so that each module handles a different responsibility. Each module can contain a sub-module `impl` within which the implementations are defined.

The first project includes the following modules:

- **region**: it models the concept of a region of space.
- **geoshape**: it models the concept of a geometric shape that represents a region of space.
- **theory**: it models the concept of a Prolog theory that is obtained from a certain region of space. This theory is called `SpatialTheory`. Both mutable and immutable versions are provided.
- **tile38**: it provides the functionality to interact with Tile38. It also contains the **tile38object** sub-module which models the objects that Tile38 accepts.

The second project includes the **solver** module, which models the Prolog solver that supports the concept of `SpatialTheory`.

4.3 Region module

The first module deals with defining the representation of a region and its implementations.

The main problem is understanding how a region must be defined in order to use it to recover the data that is geolocated with Tile38. In fact, given a region of space, our goal is to retrieve the information that is located in it.

Among the commands that Tile38 provides, the following can be exploited for this scenario:

- Nearby: searches a collection for objects that are close to a specified point.
- Within: searches a collection for objects that are fully contained inside a specified bounding area.
- Intersects: searches a collection for objects that intersect a specified bounding area.

These commands provide two alternatives: *nearby* allows you to define a point and search for information around it by defining a certain radius, *intersects* and *within* require a bounding area, which delimits the area to be searched.

Taking a cue from these commands, at least two types of regions will be needed in this project: a **bounding area**, and an **area** around a point, see 4.1.

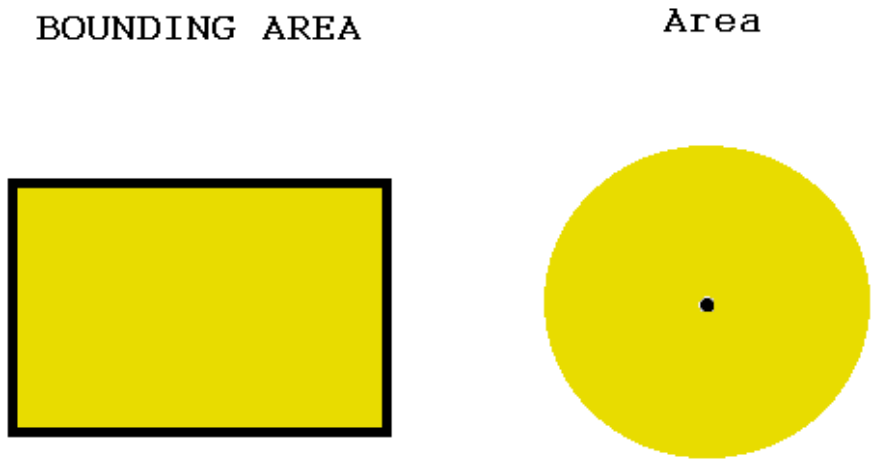

Figure 4.1: A bounding area and an area.

4.3.1 Geolocating a region

Since a `Region` itself could be geolocated (the reason behind this decision is explained in 4.6.4), each region must also provide a geometric shape, called `GeoShape`.

4.3.2 Summary

Region

Represents the concept of a region of space and it is the base interface for every region. Every region must expose a region type, an identifier and the geometric shape.

RegionType

The possible types of regions are defined with an `enum`. At least two different types are needed:

- The bounding area which is defined in Tile38 as a bounding box that consists of two points: the first being the southwestern most point and the second is the northeastern most point.
- The area around a point is a simple point with a given radius.

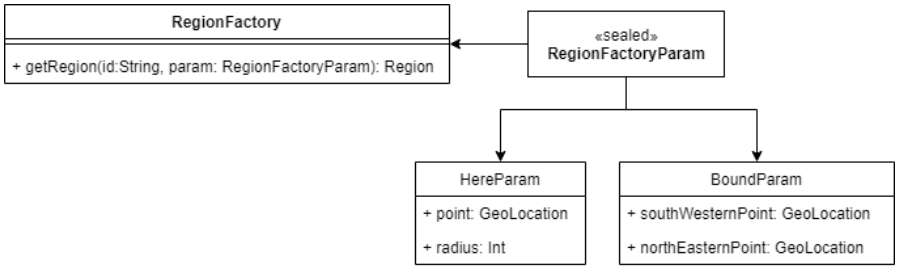

Figure 4.2: Factory pattern with sealed classes.

RegionFactory

Having planned the possibility of having regions of different types, the creation is delegated to a factory. Although it is reasonable to think that for different regions different data will be needed (a bounding area is defined by two vertices, an area around a point is defined by a point and a radius), Kotlin provides the possibility to use *sealed classes* as a solution, see figure 4.2.

4.4 GeoShape module

This module represents the concept of the geometric shape of a region, with all the information needed to be able to geolocate in Tile38. Each region will have its own geometric shape.

4.4.1 Summary

GeoShape

Represents the concept of a geometric shape that represents a region of space. Every shape must expose an identifier, a shape type and the coordinates.

- a method `convertToTile38Object()` is required to convert the shape into an object that can be accepted by Tile38.

4.5 Theory module

In this module it was necessary to understand how to integrate spatial concepts within 2P-Kt.

The goal is to allow to query a certain region of space where certain clauses are valid, so as to constrain a Prolog theory on the basis of what is true in the given area.

Two possible approaches were considered:

- Solver approach: the idea was to define a component that enables the interaction with Tile38, which managed by the solver, could be used to load the clauses from a certain region.
- Theory approach: the idea was to define a new implementation of `Theory`, which is located in a certain region of space, and consequently when an operation is requested on it, the changes are also applied to the space by interacting with a component that enables the interaction with Tile38.

The second approach is the best one, as it is better suited to the 2P-Kt project: it allows to extend it without modifying what is already present, and allows for a better division of responsibilities.

4.5.1 Theory methods

A Prolog program is basically a collection of *Clauses*. A theory, during execution, can be read and written (enabling meta-programming) and because of these features it can be referred as a `Clause` *database* [78].

The theory module implements the `ClauseDatabase` abstraction, and as a real database it has methods to read, write and remove `Clauses` [78]:

- `assertA`, adds the given clause before the others.
- `plus`, aliases the `assertZ` method
- `assertZ`, adds the given clause after others
- `contains`, checks for presence of matching clauses
- `get`, retrieves all matching clauses
- `retract`, deletes one matching clause and returns a `RetractResult`
- `retractAll`, deletes all matching clauses and returns a `RetractResult`

The matching is done using `structurallyEquals`, a method to enable structural comparison of Terms.

The `RetractResult` type holds the information of a retract operation (removal operation). It can be a `Success`, and hence contains the new database and the removed `Clauses`, or it can be a `Failure`, containing the same database with no modifications [78].

In order to seamlessly integrate situated reasoning to the current 2P-Kt project, we have decided to give a new implementation of `Theory`, which takes into account the fact of being situated in a region of space.

There are two versions that need to be implemented, one that is mutable and one immutable.

4.5.2 Caching clauses

Having chosen to adopt a new implementation of `Theory`, it was necessary to define how the loading of the clauses occurs. This can be done in two ways, one static and one dynamic, and it was decided for the latter.

A `SpatialTheory` when instantiated has no clauses in it. These will in fact be loaded only when an operation on the theory is performed, such as an assert, and only the first time. The clauses are cached and made available for subsequent operations. Subsequent requests on the same theory will no longer need to reload the clauses, making them lighter.

In this way instantiating a theory does not mean directly loading the clauses, and the operation is postponed to when is required.

4.5.3 Summary

SpatialTheory

Represents the concept of an immutable theory loaded from a certain region of space. The operations carried out on the theory must not have effects on it but return a new one on which the effects will be visible.

- a method `setRegion(region: Region)` is required to set the region from which the theory can load the clauses.
- a method `loadClausesIfNecessary()` is required to load the clauses from the specified region and cache them when requested for the first time.
- `assertA, assertZ, plus`, the assert methods will not simply have to return a new theory with the asserted clauses, but will also have to locate the clauses in the region of space.
- `retract, retractAll`, the retract methods will not simply have to return a new theory with the matching clauses retracted, but also remove the matching clauses from the region of space.

MutableSpatialTheory

Represents the concept of a mutable theory loaded from a certain region of space. The difference from the immutable version is that the changes will be applied directly to the theory. The theory itself is returned instead of a new one.

4.6 Tile38 module

As for this module, it provides a way to establish a connection with Tile38, it defines the list of commands that can be used to interact with it, and finally it models the concept of an object that can be geolocated in Tile38. In fact, as far as geolocation is concerned, it was decided to separate the concept of a region of space from that of objects in the space:

- A region of space represents an area on which at least one of the search methods offered by Tile38 can be used to retrieve the objects located in it. For example, a region could be a bounding box, on which the *intersect* command can be used, which searches a collection for objects that intersect that region.
- On the other hand, there can be different types of objects which can be geolocated, and therefore which can be searched within a region.

In particular it is necessary to understand which objects can be geolocated with Tile38 and how.

4.6.1 Geolocation of data

The geolocation of data is an important aspect. Tile38 was adopted as it provides simple commands which can be exploited directly to solve the issues posed by this thesis, such as geolocalizing data, recovering data belonging to a certain region of space and removing data from space.

The first problem to be faced concerns how to geolocate the data in Tile38, that is, the identification of the geographical position in the space of a given object.

The command that Tile38 makes available for this problem is the *set* command. This command set the value of an id, and if is already associated to that key/id, it will be overwritten. Given a key and an id, different objects can be located in the space:

- Point: a simple point defined by a latitude and a longitude.
- Bounds: a bounding rectangle defined by four values (southwest latitude, southwest longitude, northeast latitude, northeast longitude)

- Geohash: a way of expressing a location using a short alphanumeric string.
- GeoJSON: a GeoJSON object.

The choice fell on the GeoJSON format, as it represents an industry standard format and makes it possible to represent a variety of object types.

In this way, regardless of the implementation that will be given of an object, if it is convertible in GeoJSON format then it can be geolocated.

4.6.2 Associating clauses to objects

Since what is localized are GeoJSON objects, it is possible to decorate the object with a key-value pair, which associates the clauses as a value to a key common to all objects.

Clauses can be defined as a `String`, and 2P-Kt provides the ability to parse a `String` into a list of clauses.

Consequently, given a clause, it is possible to use its `String` representation as a value to decorate the objects, by associating it to the relative key.

When it is necessary to retrieve data from a certain region of space, it will be sufficient to retrieve the values associated with the common key, which decorate each object, and parse them into clauses.

4.6.3 Ordering of objects

The ordering of the objects that are loaded by Tile38 depends on the type of command being used. For this reason, an arrangement has been devised that is common to all.

Given the list of objects loaded from a region, an order has been defined in which the position acts as a discriminant. Objects are sorted from the leftmost to the rightmost.

In this way, the ordering mirrors the spatial one.

4.6.4 Geolocating Clauses

A `SpatialTheory` is a `Theory` located in a certain region of space. This region can be used to load the clauses associated to the localized objects that are enclosed within it.

A theory also provides for the possibility of asserting new clauses, that is, adding them before (or after) the others already existing. These clauses will not simply have to be added to the theory, but also localized within the region.

We need to define what it means to add a clause before (or after) the others, within the given region. There are two scenarios:

- As already stated in 4.6.3, objects that are loaded from a region are sorted by their position, from leftmost to rightmost to mirror the spatial ordering. The current theory is represented by the clauses associated to these objects. If any objects are already present inside the region, asserting a clause means taking the leftmost (or the rightmost) object, which has associated clauses, and adding it before (or after) them.

- Otherwise, if there are no objects, it is necessary to localize one to decorate with the clause. In this case, the region itself is converted into an object that is localized and to which the clause is associated.

Whoever loads the clauses from the same region will find them in the correct order.

4.6.5 A parser of results

Given the result of a command to search a region executed on Tile38, it will be necessary to parse it in order to obtain usable data.

A parser that given the result of a command, returns a list of objects will be provided. The objects will be sorted based on their position.

4.6.6 Summary

Tile38Object

Represents the concept of an object that can be converted in GeoJSON format so that it can be geolocated with the *set* command of Tile38. It will contain an identifier, the coordinates of the object, the list of clauses that the object contains and the common key to which the clauses will be associated as a value in the GeoJSON representation. It is the base interface for every object.

- a `convertToGeoJSON` method is required to convert the object in GeoJSON format, so that is supported by Tile38. In addition to the representation of the geometry, the GeoJSON must also contain a key-value pair where the value represents the clauses. The key is common to all objects.

- it will also be necessary to provide methods to add and remove clauses from the object list.

Tile38ObjectType

The possible types of object are defined with an `enum`.

Tile38ObjectFactory

As decided for the regions, the creation of the objects is delegated to a factory which will return an object of the requested type.

Comparator

A `Comparator` of objects is needed, which sorts them according to their position, from the leftmost to the rightmost.

Tile38Commands

A list of the possible commands used to interact with Tile38 is needed.

Tile38Parser

A parser of the results obtained from Tile38 commands.

Tile38Connection

An object that provides the method to reflect the operations applied to the theory on the region of space is required.

- `loadClausesInRegion(region:Region):List<Clause>`, a method to load all clauses of a region depending on the type of region.
- `putClauseFirst(clause:Clause, region:Region)`, a method that reflects the `assertA` and add the clause to the region of space before the other clauses.
- `putClauseLast(clause:Clause, region:Region)`, a method that reflects the `assertZ` and add the clause to the region of space after the other clauses.
- `retractClause(clause:Clause, region:Region)`, a method that retract a matching clause from the region of space.
- `retractAllClause(clause:Clause, region:Region)`, a method that retract all matching clauses from the region of space.

4.7 Solver module

The 2P-Kt solver module contains the main implementation of a `Solver`, called `ClassicSolver`, that provides a Prolog ISO Standard resolution. This implementation is State-Machine-Based. An instance of the solver can be created via the `ClassicSolverFactory` and it is also accessible via `Solver.classic`.

For this thesis the main idea is to define a solver that can work with the previously defined spatial theory without modifying what is already present in 2P-Kt. Knowing the region of space we want to query, the goal is to be able to provide it to the solver, so that it can perform a query bound to that area.

The strategy that has been decided to adopt is to exploit the transitions of the finite state machine: specifically the `solver-classic` module provides a particular sort of solution iterator supporting hijack of state transition, the `MutableSolutionIterator`. This allows to alter the execution context or the destination state of the transition, in particular what we need is to be able to get the current theory of the context and set it to the new region it belongs to. In this way, when the solver will query the knowledge base, it will depend on the region that was provided.

Consequently, it is necessary to define a new solver which adopts as the solution iterator the `MutableSolutionIterator`. Following the example of the classic solver, this solver is called `SpatialClassicSolver` and an instance of it can be created via the `SpatialSolverFactory` and it is also accessible via `Solver.classic`.

Chapter 5

Implementation

This chapter describes how the decisions made in the design are concretized in the implementation. Each section describes a different project module.

5.1 Region module

A `Region` is an interface, and each region implementation must provide an identifier, a region type and the geometric shape.

In this way, a region can be used both to retrieve information from space, but it can also be geolocated by exploiting its geometric shape.

Region

An interface that represents a region of space.

- `val type: RegionType`, the region type.
- `val id: String`, the region identifier.
- `val shape: GeoShape`, the region geometric shape.

As previously stated, two important regions have been identified: the bounding area and the area. The first has been renamed to bounds (`Bounds`). The second to here (`Here`), as it is designed to represent the current position of an agent with a certain coverage area. These types have been defined with an enum class.

```
enum class RegionType { BOUNDS, HERE }
```

5.1.1 Bounds

The `Region` implementation of the type `RegionType.BOUNDS` was called `Bounds`.

Bounds

A region of type `BOUNDS` is an area represented by a bounding box, which consists of two `GeoLocation` points: the first being the southwestern most point and the second is the northeastern most point. An additional parameter has been added, a Boolean value called `within` that determines whether the search on the region should be carried out considering only what lies within it, or also what intersects it. The region must also provide the geometric shape, in this case a `GeoRectangle`.

- `val shape: GeoShape`, a rectangle with the four vertices defined as follows:
  - Southwestern vertex: same as the southwestern vertex of the bounding box.
  - Northeastern vertex: same as the northeastern vertex of the bounding box.
  - Southeastern vertex: can be obtained by combining the longitude of the northeastern vertex of the box and the latitude of the southwestern one.
  - Northwestern vertex: can be obtained by combining the longitude of the southwestern vertex of the box and the latitude of the northeastern one.

A region of type `Bounds` can be used in conjunction with the *intersect* and the *within* commands to search for objects that intersect or are within the specified area, see 5.1.

5.1.2 Here

The `Region` implementation of the type `RegionType.HERE` was called `Here`.

Here

A region `Here` represents a point in space with a certain coverage area. Accordingly, it consists of a `GeoLocation` and an `Int` value that represents the radius which determines the area.

- `val shape: GeoShape`, a point that is located in the same `GeoLocation`. GeoJSON does not support the circle among its geometric figures, which would have been the most immediate representation for an area. For this reason, the `Region` is considered for loading geolocated data, which is supported by the previously explained command *nearby*, but the related `GeoShape` to which it can be converted is a point.

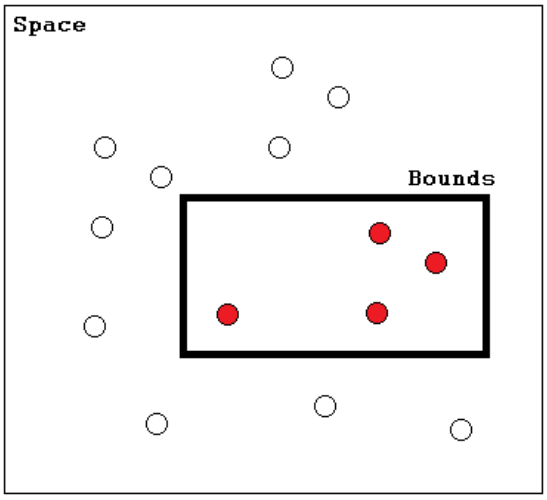

Figure 5.1: Using Bounds to find objects that intersect or are within the region.

A region of type `Here` can be used in conjunction with the *nearby* command to search for objects that are nearby the specified point, the search is limited by the radius, see 5.2.

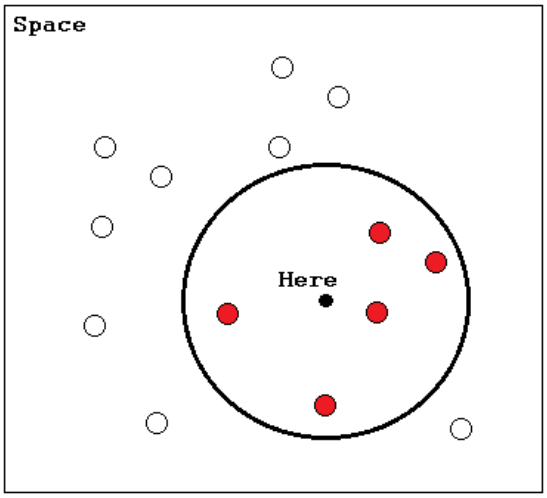

Figure 5.2: Using Here to find nearby objects.

5.1.3 RegionFactory

The creation of regions is implemented through a factory: based on the parameters that are supplied to the factory, it is possible to obtain the region you need.

```
object RegionFactory {
    fun getRegion(id: String, param: RegionFactoryParam)
       : Region {
        return when (param) {
            is RegionFactoryParam.HereParam -> Here(id,
               param.point, param.radius)
            is RegionFactoryParam.BoundParam -> Bounds(
               id, param.southWesternPoint, param.
               northEasternPoint,
            param.within)
        }
    }
}
```

A `sealed class` is provided which contains a `data class` for each region to manage the parameters difference, as explained in the design phase.

- `Bounds` requires as parameters the southwestern point and the northeastern point defined as `GeoLocation`. The Boolean `within` was added to the class with the default value set to false.
- `Here` requires as parameters a point defined as a `GeoLocation` and the radius in `Int`.

5.2 GeoShape module

A `GeoShape` is an interface, and each object implementation must define a method that convert the shape into a `Tile38Object`.

GeoShape

A representation of a shape that represents a region of space.

- `val id: String`, the identifier of the shape.
- `val coordinates: String`, the string representation of the coordinates of the object, defined in GeoJSON format.
- `fun convertToTile38Object(clauses:List<Clause>):Tile38Object`. A method that converts the shape into an object that can be converted in GeoJSON format.

5.2.1 GeoPolygon

An interface used to collect the `GeoShape` of type polygon.

5.2.2 GeoLocation

To represent a position, a specific class has been defined.

GeoLocation

A geolocation is a point defined by a longitude and a latitude.

- `class GeoLocation(val longitude: Double, val latitude: Double)`

5.2.3 GeoRectangle

A `GeoRectangle` is an implementation of a `GeoPolygon`.

GeoRectangle

A rectangle that is defined by four points, where each point is a `GeoLocation` and represents a vertex, and an identifier.

- `fun convertToTile38Object(clauses:List<Clause>):Tile38Object`. A method that returns an object that can be geolocated in Tile38.

5.2.4 GeoPoint

A `GeoPoint` is an implementation of a `GeoShape`.

GeoPoint

A point that is defined by a `GeoLocation`, which represents the position of the point and an identifier.

- `fun convertToTile38Object(clauses:List<Clause>):Tile38Object`. A method that returns an object that can be geolocated in Tile38.

5.3 Theory module

This module implements the two versions of `SpatialTheory`.

SpatialTheory

Is an interface that models a `Theory` that loads its clauses from a `Region` of space.

- `val region: Region`: the region from which the clauses are loaded.
- `setRegion(region: Region): SpatialTheory`, a method that returns a spatial theory of the specified `region`.

The implementation of this theory will have to be immutable: the methods will not modify the current theory, but will return a new one with the required changes.

MutableSpatialTheory

Is an interface that models a `Theory` that loads its clauses from a `Region` of space. The difference with a `SpatialTheory` is that the implementation will have to be mutable: the methods will directly modify the theory and return the theory itself.

- `setRegion(region: Region): MutableSpatialTheory`, a method to modify the current region of the spatial theory which is then returned.

The `val region` that was present in the `SpatialTheory` will be inserted in the implementation constructor as a variable, as it will have to be modifiable, but only by the `setRegion` method.

5.3.1 Tile38SpatialTheory

Having chosen Tile38 as a tool to geolocate the data, a specific implementation of `SpatialTheory` was provided for Tile38: `Tile38SpatialTheory`. In this way, if we wanted to use different technologies in the future, it would be possible to do so by adding new `SpatialTheory` implementations specific to them.

Each implementation must do two things: implement the methods defined by the `SpatialTheory` interface and those of `Theory`.

To create a connection to Tile38 an ulterior field is required: the Redis URI, a string which is used to create the connection. Therefore, being a dedicated implementation for Tile38, the field was added in the constructor.

Tile38SpatialTheory

Is an implementation of a `SpatialTheory` specific for Tile38. It is immutable.

- `setRegion(region: Region): SpatialTheory`, this method returns a new theory located in the new region, not modifying the current one, that is immutable.

- `loadClauseFromTile38IfNecessary()`:

```
/** Load clauses and put each one in cache*/
private fun loadClausesFromTile38() {
  cache.assertA(Tile38Connection.loadClausesInRegion
     (region, uri))
  loaded = true
}
/** Load clauses if not already done*/
private fun loadClauseFromTile38IfNecessary() =
  if (!loaded) loadClausesFromTile38() else Unit
```

  when a spatial theory is instantiated, the clauses are not loaded automatically, but are retrieved only when required by an operation. For this reason, the methods of the theory that require the clauses, will first have to execute the method to load them if necessary. At the first operation, the clauses are loaded and put in a cache.

  - `cache`, the cache is represented by a `MutableTheory` initialized empty. To insert the clauses in the cache it is sufficient, given the list of clauses obtained with the method provided by the connection object, to delegate an `assertA` operation to the `MutableTheory`, that inserts them in order.

- `assertA(clause: Clause): Theory`, this method adds the given clause before all other clauses. For a spatial theory implemented for Tile38, asserting a clause does not simply mean adding the clause to the theory, but also geolocating it. This is possible with the methods provided by the connection object. Being immutable, the modification is not directed on the current theory and a new one is returned. The new theory, which draws data from the same region, will be able to load the new clauses that have been geolocated, in addition to those already present.

```
/** Adds given clause before all other clauses in
   this theory */
override fun assertA(clause: Clause): Theory {
 loadClausesFromTile38IfNecessary()
 Tile38Connection.putClauseFirst(clause,region,uri)
 return Tile38SpatialTheory(region, tags, uri)
}
```

- `assertA(clauses:Iterable<Clause>):Theory`, this method adds the given clauses before all other clauses in this theory. It is easy to transform this case in the first one, by converting the `Iterable` into a `List` and iterating it in reverse order.

- `assertA(clauses:Sequence<Clause>):Theory`, this method adds the given clauses before all other clauses in this theory. It is easy to convert this case to the second, by creating an iterable instance that wraps the original sequence.

- `assertZ`, adds the given clauses after all other clauses in this theory. The implementation is analogous to that of `assertA` and three methods are provided.

- `plus(theory: Theory): Theory`, aliases the `assertZ` method.

- `get(clause:Clause):Sequence<Clause>`, retrieves matching clauses from the cache.

- `contains(clause: Clause): Boolean`, checks if a given clause is contained in this theory. It uses the `get` method and checks if the sequence contains at least one element.

- `retract(clause: Clause): RetractResult<Theory>`, this method tries to delete a matching clause from this theory. The retract operation is performed on a a mutable copy of the current theory, which must not be changed directly. In this way the retract methods are delegated to the already existing methods provided by the implementation of `MutableTheory`. If it is a `Success`, then the deleted clause is also removed from the region. Unlike the assert, the retract returns a `RetractResult` type that holds information of the retract operation.

```
1 /** Tries to delete a matching clause from this
      theory */
2 override fun retract(clause: Clause): RetractResult<
      Theory> {
3  loadClausesFromTile38IfNecessary()
4  val newTheory = ClauseQueue.of(clauses)
5  return when (val retracted = newTheory.
       retrieveFirst(clause)) {
6   is RetrieveResult.Failure ->
7    RetractResult.Failure(this)
8   else -> {
```

```
9       (retracted as RetrieveResult.Success).clauses.
            forEach{ Tile38Connection.retractClause(it,
            region, uri)}
10      RetractResult.Success(
11       Tile38SpatialTheory(
12        region,
13        tags,
14        uri
15       ),
16       retracted.clauses
17      )
18     }
19    }
20  }
```

- `retract(clauses:Iterable<Clause>):RetractResult<Theory>`, tries to delete the matching clauses from this theory. The approach is similar to that of the previous case, but is done for each clause of the `Iterable`.
- `retract(clauses:Sequence<Clause>):RetractResult<Theory>`, tries to delete the matching clauses from this theory. This case can be traced back to the previous by creating an `Iterable` instance that wraps the original sequence.
- `retractAll(clause:Clause):RetractResult<Theory>`, this method tries to delete all matching clauses from this theory, and returns a `RetractResult`. In case of success, the deleted clauses are also removed from the space. The difference from the previous case is in the method provided by the connection object, which instead of removing the first matching clause from the region, deletes them all.

5.3.2 Tile38MutableSpatialTheory

An implementation of a `MutableSpatialTheory` specific for Tile38 is given. The theory is directly modified and all methods that used to return a new theory in the immutable case, now return the theory itself. Only the methods for which there are additional differences besides the latter will be discussed.

- `setRegion(region: Region) : MutableSpatialTheory`, this method will not simply return a new theory with the new region, but it will also change the current region and reset the cache. The value of loaded is set to false, so the clauses can be loaded from the new region when necessary.

```
override fun setRegion(region: Region):
   MutableSpatialTheory {
  return this.also {
    this.cache = MutableTheory.empty()
    this.region = region
    this.loaded = false
  }
}
```

- `AssertA, AssertZ`, the assert methods follow the same logic as those of the immutable version, the difference is that the `cache` is directly modified and what is returned is the theory itself, in this case a `MutableTheory`. We rely on the fact that the `cache` is an implementation of `MutableTheory` and the assert method is delegated to it. In the same way as in the immutable version, the clauses are also geolocated.

- `Retract`, in the mutable case, creating a mutable copy of the current cache on which to make the retrieval of the clauses is not necessary, as it can be directly applied to the current cache, which can be modified. As in the mutable case, if the `RetractResult` ends successfully, the clauses are removed from the relative region of space.

5.4 Tile38 module

This section covers how interaction with Tile38 has been enabled.

5.4.1 Tile38Object

A `Tile38Object` is an interface that represents the concept of an object that can be converted in GeoJSON format so that it can be geolocated.

- `val id: String`, the identifier of the object.

- `val coordinates: String`, the coordinates of the object prepared for the GeoJSON format.

- `val clauses: List<Clause>`, the list of clauses associated to the object.

- `val key: String`, the common key between the objects to which the clauses are associated when converted in GeoJSON format.

- `convertToGeoJSON(): String`, convert the object into a string in GeoJSON format.
- `addClauseFirst(clause: Clause): Tile38Object`, returns a new object with the given clause in the first position of the list.
- `addClauseLast(clause: Clause): Tile38Object`, returns a new object with the given clause in the last position of the list.
- `retractClause(clause: Clause): Tile38Object`, returns a new object with the first occurence of the given clause removed.

The possible types of objects have been defined with an enum class:

```
enum class Tile38ObjectType(val type: String) {
    POINT("Point"),
    POLYGON("Polygon")
}
```

5.4.2 Tile38Point

The implementation of the object type `Tile38ObjectType.POINT`.

Tile38Point

An object of type point consists of an identifier, its coordinates and the list of clauses.

- `convertToGeoJSON(): String`, the object is converted into a GeoJSON string representing a point.

The implementation of the other methods followed that dictated by the design phase.

5.4.3 Tile38Polygon

The implementation of the object type `Tile38ObjectType.POLYGON`.

Tile38Polygon

What was said for the point is valid, but the conversion operation transforms it into a GeoJSON string representing a polygon.

5.4.4 Tile38ObjectFactory

The creation of the objects is implemented through a factory: based on the type that is supplied to the factory, the needed object is obtained.

```
object Tile38ObjectFactory {
    fun getObject(type: String, id: String, coordinates:
        String, clauses: List<Clause>): Tile38Object {
        return when (type) {
            Tile38ObjectType.POINT.type -> Tile38Point(
               id, coordinates, clauses)
            else -> Tile38Polygon(id, coordinates,
               clauses)
        }
    }
}
```

5.4.5 Tile38Commands

Lettuce provides two alternatives to defining commands for interacting with Tile38:

- implement the `ProtocolKeyword` interface through an `enum`, where each item of the enumerator is a command keyword, for example *NEARBY*. Then Lettuce provides an API to dispatch commands.
- using command interfaces that provide you with an higher level of abstraction by declaring command methods on a Java interface. A method signature matches the command to invoke. For example:

```
@Command("SET region ?0 OBJECT ?1")
fun set(
    key: String,
    geoJSON: String
): List<Any?>?
```

In this case a command interface has been chosen, as it provides a dynamic way for type-safe Redis command invocation. It is also less verbose than invoking a Redis command.

Tile38Commands

An interface which collects the possible usable command:

- `set`: set the value of an id. If a value is already associated to that key/id, it'll be overwritten. This command requires a key (`String`) and the associated object, defined as a GeoJSON `String`.

- `intersects`: searches a collection for objects that intersect a specified area. This area is defined as a bounding box. This command requires an identifier (`String`) of a collection of objects (a common identifier for all geolocated objects was defined), the southwestern point and the northeastern point latitude and longitude, both defined as `Double`.

- `within`: same as intersects, but searches a collection for objects that are within the specified area.

- `nearby`: searches a collection for objects that are close to a specified point. The distance is determined by a given radius. This command requires an identifier (`String`) of a collection of objects, the latitude and longitude in `Double` of the point and the radius (`Int`).

- `drop`: remove all objects associated to a specified key. This command doesn't require any parameters and is applied to the common identifier of all geolocated objects.

Accessing these commands can be done through a `RedisCommandFactory` that exposes all the defined commands.

```
1 private val factory = RedisCommandFactory(connection)
2 private val commands = factory.getCommands(
    Tile38Commands::class.java)
```

5.4.6 JSONUtils

Through the intersect/within and nearby commands it is possible to obtain the list of geolocated objects: as explained before, these objects are in GeoJSON format and contain a common key to which the clauses are associated. To provide an ordering of the clauses it was decided to define an ordering depending on the longitude, from the leftmost position to the rightmost. To do so, a Comparator has been defined.

GeoJSONComparator

A Comparator of `JSONArray`.

- `compare(a: JSONArray, b: JSONArray)`, given two array, each contains a `JSONObject`, for each object the longitude values are taken and sorted from smallest to largest. Then the minimum values for each object are taken and compared to find which is the leftmost.

A method `sortGeoJSONArray(geoJsonArray:JSONArray):List<JSONArray>` has been defined, which sort a `JSONArray` with the `GeoJSONComparator`.

5.4.7 Tile38Parser

The *intersect*, the *within* and *nearby* commands defined in the command interface return a `List<Any?>?`. A parser is then set up to manipulate the result into a more convenient one.

Tile38Parser

A parser of Tile38 results.

- `parseResult(result:List<Any?>?):List<Tile38Object>`, given a Tile38 command result, a list of `Tile38Object` is returned. The list is sorted with the Comparator of GeoJSON.

5.4.8 Tile38Connection

A `Tile38Connection` is an object that exposes the methods needed to interact with Tile38.

Tile38Connection

Represents the link between the `SpatialTheory` and Tile38, and provides the necessary methods to enable interaction.

To interact with Tile38, Lettuce provides the ability to define a client on which a connection can be established. Consequently, each method, given the `uri` address passed by parameter, checks if a connection has already been established on that address, otherwise it creates it. In this way the connection is established when requested, and changed if requested on a different address.

Having used a Kotlin object, `Tile38Connection` is a `singleton` that exposes the necessary methods.

- The `Theory` interface requires two types of assertion methods, `assertA` and `assertZ`. The first method adds the given clause before the others, the latter adds the given clause after the others. From the spatial point of view,

the choice made was to order the spatial information based on their position, from the leftmost to the rightmost, using longitude as discriminant. Accordingly, an assert operation on the current theory is defined spatially:

- `assertA`: adding a clause before the others means that the clause should be located in the *leftmost* position available, in order to maintain the spatial order.
- `assertZ`: adding a clause after the others means that the clause should be located in the *rightmost* position available, in order to maintain the spatial order.

Two methods have therefore been made available: one that locates the clause in the leftmost position, one in the rightmost position.

- `putClauseFirst(clause:Clause, region:Region, uri:String)`. A method that given a clause and a region of space, locates the clause in the leftmost position. This operation is done as follows:
  - If the supplied region is of type `Bounds`, all localized objects that intersect or are within the bounding area are loaded. If the region is of type `Here`, all localized objects that are close to the region are loaded.
  - If the previous operation doesn't load any objects, the region itself is located by situating the corresponding `GeoShape`.
  - Otherwise it will be sufficient to take the clauses defined in the first object, which is the leftmost, and add the clause to be asserted before them. Then the object is updated and put in Tile38.

  In this way, whoever requests the clauses of the current region will obtain a list of clauses in which the first is the one just inserted.

  Considering the possibility of multiple invocations of the same method, the methods have been designed to be thread safe. The main reason this is necessary is to avoid the lost update problem. In order to add a clause to an object located with Tile38, this method first retrieve the list of objects and then make the change on the correct one which is then updated. If different threads load the objects list at the same time and then update the object, only the last update will be made, as it will be applied to the old object, losing the change made by others. For this reason, all methods that perform an update are accompanied by the **Synchronized** keyword which guarantees one access to the method at a time.

  - `putClauseLast(clause:Clause,region:Region,uri:String)`, given a clause and a region of space, locates the clause in the rightmost position. This method works as the previous one, but the clause is associated to the last object after the existing clauses.

- `retractClause(clause:Clause,region:Region,uri:String)`, remove the first matching clause from the region.

- `retractAllClause(clause: Clause, region: Region, uri:String)` remove all matching clauses from the region.

- `loadClausesInRegion(region: Region, uri:String):List<Clause>`. A method that given a region, returns the list of clauses located in the region. Based on the type of the region, the clauses are loaded in two different ways:

  - `Bounds`: in the case of a region of type `Bounds`, the loading is done through the intersect command, that searches a collection for objects that intersect the region, or the within command.
  - `Here`: in the case of a region of type `Here`, the loading is done through the nearby command, that searches a collection for objects that are close to the specified point.

  The result is parsed and a list of `Tile38Object` is obtained. The clauses are then retrieved from the objects and the list is returned.

- `dropAll()`, remove all objects from a specified key. Is a function useful mainly for testing, which allows to remove all objects from a specified key by using the Tile38 `DROP` command on the `region` key, which is the one common to all the located objects. This feature is used to remove all geolocated objects from the space, so as to start from a basic situation.

5.5 Solver module

This module implements the `SpatialClassicSolver`.

5.5.1 SpatialClassicSolver

The `SpatialClassicSolver` implements the abstract class provided by 2P-Kt for classic-like solver, the `AbstractClassicSolver`. The peculiarity lies in the type of solution iterator it uses, in particular the `MutableSolutionIterator`.

```
override fun solutionIterator(
    initialState: State,
    onStateTransition: (State, State, Long) -> Unit
) = MutableSolutionIterator.of (
    initialState,
    this::hijackStateTransition,
    onStateTransition
)
```

This iterator allows to alter the execution context or the destination state of the Prolog State Machine, and this is done through the following method:

```
    private fun hijackStateTransition(
        source: State,
        destination: State,
        index: Long
    ): State
```

Whenever the destination state of the transition is that of `StateRuleSelection`, which is the one that precedes each access to the knowledge base, the destination context is altered. Within the context, two knowledge bases are defined, a static one that hold all static predicates loaded upon construction, a dynamic one for changeable knowledge base. The method takes care of setting the region of the `SpatialTheory` that defines the static knowledge base and of the `MutableSpatialTheory` that defines the dynamic one to the current region.

Among the methods the solver provides for solving a goal, the `solveOnce` method accepts resolution options and it is exploited to provide the region of interest.

- `solveOnce(goal:Struct,option:SolveOptions):Solution`, this method checks if among the custom options, defined as a key-value map, there is the *Region* key, which is associated with the region on which to perform the resolution. If present, the region associated to the key replaces the current one.

- `private var region: Region`, the current region that the solver is to query. When the solver is instantiated the region is set to a default value defined in the factory.

The creation of the solver is delegated through a factory, following the example of the `ClassicSolver`, and the extensions called `SpatialSolverExtensions` are provided.

- `Solver.Companion.spatialSolver`, it makes the solver also accessible via `Solver.classic`. The creation is delegated to the factory method with the same name which allows to instantiate the solver.
- `Solver.Companion.spatialSolverWithDefaultBuiltins`, like the previous case, but adds the default builtins.

Chapter 6

Conclusions and Future Works

The work that has been done in this thesis is summarised in section 6.1. Finally, some possible future works are discussed in section 6.2.

6.1 Summary

This thesis finds its place in the context of BDI agents and aims to enable a form of situated spatial reasoning.

A survey proposed regarding the BDI model and the possible techniques and technologies that can be integrated to provide a form of spatial reasoning highlighted a lack of technology that we have decided to fill.

We wanted to define a type of spatial reasoning that consists of providing a way to locate logical information in certain spatial areas and to be able to constrain reasoning on them. In this way it is possible to enable a form of situated spatial reasoning. This goal was supported by two technologies: *Tile38*, one of the most famous geospatial database, and *2P-Kt*, a multi-paradigm logic programming framework written in Java.

As a result, in this thesis we have proposed *Geo2p*, a technological prototype based on *2P-Kt* that allows you to query situated information in tuProlog, enabling situated spatial reasoning: given a region of space where certain Clauses are valid, a Theory can be defined, constraining the knowledge on what is true in the selected area. The functionalities that were required by the scenario discussed in the design phase have been respected: given a certain region of space where certain clauses are valid, the current theory is constrained on the basis of what is true in that area. Prolog clauses can also be geolocated within a certain region of space or removed from it.

6.2 Future Works

In this section are listed some possible future works which serve to consolidate the thesis work or to take it in new directions.

Other Technologies

One possible direction is to define new implementations based on a technology different from Tile38.

Performance

One aspect that has not been considered in this thesis is that of performance. One might also think about comparing the performance of this Tile38 specific implementation with one that uses a different technology.

BDI in 2P-Kt

As already stated, the idea behind this thesis was to provide something that could be exploited to perform spatial reasoning. An idea would be to integrate BDI concepts into *2P-Kt*, so that *Geo2p* could be directly exploited. It would be interesting to see this happening and how the agents will benefit from *Geo2p*.

Bibliography

[1] 2P-Kt. `https://github.com/tuProlog/2p-kt`. Last access: March 14, 2021.

[2] Marco Aiello, Guram Bezhanishvili, Isabelle Bloch, and Valentin Goranko. Logic for physical space. from antiquity to present days. *Synthese*, Volume 186:pp 619–632, 06 2012.

[3] Krzysztof R Apt. The logic programming paradigm and prolog. *arXiv preprint cs/0107013*, 2001.

[4] Dejanira Araiza-Illan, Anthony G. Pipe, and Kerstin Eder. Intelligent agent-based stimulation for testing robotic software in human-robot interactions. In *Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering*, MORSE '16, page 9–16, New York, NY, USA, 2016. Association for Computing Machinery.

[5] C. Badica, A. Becheru, and S. Felton. Integration of jason reinforcement learning agents into an interactive application. In *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 361–368, 2017.

[6] Alessandro Bagnoli. Game Engines e MAS: Spatial Tuples in Unity3D. Master's thesis, Alma Mater Studiorum Università di Bologna, 2018.

[7] Josh Baker. Tile38. `https://github.com/tidwall/tile38`, 2020.

[8] T. Balke, M. De Vos, J. Padget, and D. Traskas. Normative run-time reasoning for institutionally-situated bdi agents. In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 1–4, 2011.

[9] Kim Bauters, Kevin McAreavey, Jun Hong, Yingke Chen, Weiru Liu, Lluís Godo, and Carles Sierra. Probabilistic planning in agentspeak using the pomdp framework. In Ioannis Hatzilygeroudis, Vasile Palade, and Jim

Prentzas, editors, *Combinations of Intelligent Methods and Applications*, pages 19–37, Cham, 2016. Springer International Publishing.

[10] J. Bañgate, J. Dugdale, E. Beck, and C. Adam. Solace a multi-agent model of human behaviour driven by social attachment during seismic crisis. In *2017 4th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, pages 1–9, 2017.

[11] Saeed Behzadi and Ali Alesheikh. Hospital site selection using a bdi agent model. *International Journal of Geography and Geology*, 2:36–51, 01 2013.

[12] Saeed Behzadi and Ali A. Alesheikh. Introducing a novel model of belief–desire–intention agent for urban land use planning. *Engineering Applications of Artificial Intelligence*, 26(9):2028 – 2044, 2013.

[13] Mouna Belhaj, Fahem Kebair, and Lamjed Ben Said. Agent-based modeling and simulation of the emotional and behavioral dynamics of human civilians during emergency situations. In Jörg P. Müller, Michael Weyrich, and Ana L. C. Bazzan, editors, *Multiagent System Technologies*, pages 266–281, Cham, 2014. Springer International Publishing.

[14] Rafael Bordini, Amal Seghrouchni, Koen Hindriks, Brian Logan, and Alessandro Ricci. Agent programming in the cognitive era. *Autonomous Agents and Multi-Agent Systems*, 34, 05 2020.

[15] Rafael H. Bordini, Ana L. C. Bazzan, Rafael de O. Jannone, Daniel M. Basso, Rosa M. Vicari, and Victor R. Lesser. Agentspeak(xl): Efficient intention selection in bdi agents via decision-theoretic task scheduling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3*, AAMAS '02, page 1294–1302, New York, NY, USA, 2002. Association for Computing Machinery.

[16] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007.

[17] Mathieu Bourgais, Patrick Taillandier, and Laurent Vercouter. An Agent Architecture Coupling Cognition and Emotions for Simulation of Complex Systems. In *Social Simulation Conference*, Rome, Italy, September 2016.

[18] J. M. L. Brezolin, S. R. Fiorini, M. de Borba Campos, and R. H. Bordini. Using conceptual spaces for belief update in multi-agent systems. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, pages 178–181, 2015.

[19] Robert Bull and Krister Segerberg. *Basic Modal Logic*, pages 1–88. Springer Netherlands, Dordrecht, 1984.

[20] C. Bulumulla, J. Chan, and L. Padgham. Enhancing diffusion models by embedding cognitive reasoning. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 744–749, 2018.

[21] Paolo Busetta, James Bailey, and Kotagiri Ramamohanarao. A reliable computational model for bdi agents. In *1st International Workshop on Safe Agents. Held in conjunction with AAMAS2003*. Citeseer, 2003.

[22] Y. E. Cakmaz, O. F. Alaca, C. Durmaz, B. Akdal, B. Tezel, M. Challenger, and G. Kardas. Engineering a bdi agent-based semantic e-barter system. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 1072–1077, 2017.

[23] Mattia Cerbara. Game engines and MAS: tuplespace-based interaction in Unity3D. Master's thesis, Alma Mater Studiorum Università di Bologna, 2018.

[24] Qingliang Chen, Kaile Su, Abdul Sattar, Xiangyu Luo, and Aixiang Chen. A first-order coalition logic for bdi-agents. *Frontiers of Computer Science*, 10, 10 2015.

[25] Yingke Chen, Jun Hong, Weiru Liu, Lluís Godo, Carles Sierra, and Michael Loughlin. Incorporating pgms into a bdi architecture. In Guido Boella, Edith Elkind, Bastin Tony Roy Savarimuthu, Frank Dignum, and Martin K. Purvis, editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, pages 54–69, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[26] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Multiagent system engineering: The coordination viewpoint. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, pages 250–259, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[27] Célia da Costa Pereira and Andrea G. B. Tettamanzi. An integrated possibilistic framework for goal generation in cognitive agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, page 1239–1246, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[28] Aniruddha Dasgupta and Aditya K. Ghose. BDI agents with objectives and preferences. In Andrea Omicini, Sebastian Sardina, and Wamberto Vasconcelos, editors, *Declarative Agent Languages and Technologies VIII*, volume 6619 of *Lecture Notes in Computer Science*, pages 22–39. Springer Berlin Heidelberg, 2011.

[29] Darryl N. Davis and Hossein Miri. Probabilistic bdi in a cognitive robot architecture. *International Journal of Computer Science and Artificial Intelligence*, pages 1–10, 09 2012.

[30] Juan F. De Paz, Manuel Pablo Rubio, and Angélica González. Dynamic planning with bayesian network applied in mas. In Yves Demazeau, Frank Dignum, Juan M. Corchado, Javier Bajo, Rafael Corchuelo, Emilio Corchado, Florentino Fernández-Riverola, Vicente J. Julián, Pawel Pawlewski, and Andrew Campbell, editors, *Trends in Practical Applications of Agents and Multiagent Systems*, pages 113–121, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[31] Enrico Denti, Andrea Omicini, and Alessandro Ricci. tuprolog: A light-weight prolog for internet applications and infrastructures. In I. V. Ramakrishnan, editor, *Practical Aspects of Declarative Languages*, pages 184–198, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[32] Meenakshi D'Souza and Rajanikanth N. Kashi. Avionics self-adaptive software: Towards formal verification and validation. In Günter Fahrnberger, Sapna Gopinathan, and Laxmi Parida, editors, *Distributed Computing and Internet Technology*, pages 3–23, Cham, 2019. Springer International Publishing.

[33] Soumitra Dutta. Approximate spatial reasoning. In *Proceedings of the 1st international conference on Industrial and engineering applications of artificial intelligence and expert systems-Volume 1*, pages 126–140, 1988.

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

[35] T. Â. Gelaim, R. A. Silveira, and J. Marchi. Towards a model of cognitive agents: Integrating emotion on trust. In *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, pages 80–86, 2015.

[36] João Carlos Gluz and Patricia Augustin Jaques. A probabilistic approach to represent emotions intensity into bdi agents. In Béatrice Duval, Jaap

van den Herik, Stephane Loiseau, and Joaquim Filipe, editors, *Agents and Artificial Intelligence*, pages 225–242, Cham, 2015. Springer International Publishing.

[37] Yiwei Gong, Sietse Overbeek, and Marijn Janssen. Business rules for creating process flexibility: Mapping rif rules and bdi rules. In Dickson K. W. Chiu, Ladjel Bellatreche, Hideyasu Sasaki, Ho-fung Leung, Shing-Chi Cheung, Haiyang Hu, and Jie Shao, editors, *Web Information Systems Engineering – WISE 2010 Workshops*, pages 142–155, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[38] T. G. Halac, E. E. Ekinci, and O. Dikenelli. Description logic based bdi implementation for goal-directed semantic agents. In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 62–65, 2011.

[39] Julian Higgins and Sally Green. *Cochrane Handbook for Systematic Reviews of Interventions*. John Wiley & Sons, Ltd, 2008.

[40] D. Holmes and R. Stocking. Augmenting agent knowledge bases with owl ontologies. In *2009 IEEE Aerospace conference*, pages 1–15, 2009.

[41] S. Isçi, O. Topçu, and L. Yilmaz. Extending the jadex framework with coherence-driven adaptive agent decision-making model. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 3, pages 48–55, 2014.

[42] Andreas Schmidt Jensen, Virginia Dignum, and Jørgen Villadsen. The aorta architecture: Integrating organizational reasoning in jason. In Fabiano Dalpiaz, Jürgen Dix, and M. Birna van Riemsdijk, editors, *Engineering Multi-Agent Systems*, pages 127–145, Cham, 2014. Springer International Publishing.

[43] Philip Kerbusch, Jeffrey Schram, and Karel van den Bosch. Modeling cultural behavior for military virtual training. In *Proceedings of the NATO MSG107 meeting. Held at: Orlando, Florida, 1-9*, 2011.

[44] Andrew Koster, Marco Schorlemmer, and Jordi Sabater-Mir. Opening the black box of trust. *Journal of Logic and Computation*, 23:25–58, 02 2013.

[45] Christian Kray. The benefits of multi-agent systems in spatial reasoning. In *FLAIRS Conference*, pages 552–556, 2001.

[46] R. Lang, S. Kohlhauser, G. Zucker, and T. Deutsch. Integrating internal performance measures into the decision making process of autonomous agents. In *3rd International Conference on Human System Interaction*, pages 715–721, 2010.

[47] Francesco Lanza, Patrick Hammer, Valeria Seidita, Pei Wang, and Antonio Chella. Agents in dynamic contexts, a system for learning plans. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 823–825, New York, NY, USA, 2020. Association for Computing Machinery.

[48] Joseph Lau, Elliott M. Antman, Jeanette Jimenez-Silva, Bruce Kupelnick, Frederick Mosteller, and Thomas C. Chalmers. Cumulative meta-analysis of therapeutic trials for myocardial infarction. *New England Journal of Medicine*, 327(4):248–254, 1992. PMID: 1614465.

[49] Seungho Lee, Young-Jun Son, and Judy Jin. An integrated human decision making model for evacuation scenarios under a bdi framework. *ACM Trans. Model. Comput. Simul.*, 20(4), November 2010.

[50] Lettuce. Lettuce. `https://github.com/lettuce-io/lettuce-core`, 2020.

[51] W. Lian, Y. Liang, and Q. Zeng. Integrating semantics and agent technology to automatic web service composition. In *2010 IEEE 2nd Symposium on Web Society*, pages 201–206, 2010.

[52] Chih-Hao Liu, Jason Jen, and Yen Chen. Using ontology-based bdi agent to dynamically customize workflow and bind semantic web service. *Journal of Software*, 7, 04 2012.

[53] Emiliano Lorini and Michele Piunti. Introducing relevance awareness in bdi agents. In Lars Braubach, Jean-Pierre Briot, and John Thangarajah, editors, *Programming Multi-Agent Systems*, pages 219–236, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[54] Wulfrano Arturo Luna Ramirez and Maria Fasli. Plan acquisition in a bdi agent framework through intentional learning. In Jan Ole Berndt, Paolo Petta, and Rainer Unland, editors, *Multiagent System Technologies*, pages 167–186, Cham, 2017. Springer International Publishing.

[55] Luis Macedo. A computational model for forms of selective attention based on cognitive and affective feelings. In *Proceedings of the international conference on cognitive modelling (ICCM 2012)*, pages 145–150, 2012.

[56] John Charles Chenoweth McKinsey and Alfred Tarski. The algebra of topology. *Annals of mathematics*, pages 141–191, 1944.

[57] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[58] Salima Mnif, Saber Darmoul, Sabeur Elkosantini, and Lamjed Ben Said. Integration of immune features into a belief-desire-intention model for multi-agent control of public transportation systems. In Francisco Javier Martínez de Pisón, Rubén Urraca, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems*, pages 459–470, Cham, 2017. Springer International Publishing.

[59] Sara Montagna, Stefano Mariani, Emiliano Gamberini, Alessandro Ricci, and Franco Zambonelli. Complementing agents with cognitive services: A case study in healthcare. *Journal of medical systems*, 44:188, 09 2020.

[60] Cynthia Mulrow. Systematic reviews: Rationale for systematic reviews. *BMJ (Clinical research ed.)*, 309:597–9, 10 1994.

[61] Emma Norling, Liz Sonenberg, and Ralph Rönnquist. Enhancing multi-agent based simulation with human-like decision making strategies. In Scott Moss and Paul Davidsson, editors, *Multi-Agent-Based Simulation*, pages 214–228, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[62] Andrea Omicini, Stefano Mariani, and Viroli Mirko. Spatial Multi-Agent Systems. `https://www.slideshare.net/andreaomicini/spatial-multiagent-systems`, 2016.

[63] I Ourdev, Hua Xie, and Simaan Abourizk. An intelligent agent approach to adaptive project management. *Tsinghua Science & Technology*, 13, 10 2008.

[64] A. Oxman, D. Sackett, and G. Guyatt. Users' guides to the medical literature. i. how to get started. the evidence-based medicine working group. *JAMA*, 270 17:2093–5, 1993.

[65] Lin Padgham and Dhirendra Singh. Situational preferences for bdi plans. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1013–1020, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[66] Shamimabi Paurobally, Jim Cunningham, and Nicholas R. Jennings. A formal framework for agent interaction semantics. In *Proceedings of the Fourth*

International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '05, page 91–98, New York, NY, USA, 2005. Association for Computing Machinery.

[67] Federico Peinado, Marc Cavazza, and David Pizzi. Revisiting character-based affective storytelling under a narrative bdi framework. In Ulrike Spierling and Nicolas Szilas, editors, *Interactive Storytelling*, pages 83–88, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[68] Y. Peng, L. Ye, Z. Zheng, J. Xiang, J. Gao, J. Ai, Z. Lu, Y. Jin, and X. Jiang. Policy enhanced grid computing. In *2009 International Conference on Education Technology and Computer*, pages 133–136, 2009.

[69] Isaac Pinyol, Jordi Sabater-Mir, Maria Pilar Dellunde, and Mario Paolucci. Reputation-based decisions for logic-based cognitive agents. *Autonomous Agents and Multi-Agent Systems*, 24:175–216, 01 2012.

[70] J. Polajnar, B. Dalvandi, and D. Polajnar. Does empathy between artificial agents improve agent teamwork? In *IEEE 10th International Conference on Cognitive Informatics and Cognitive Computing (ICCI-CC'11)*, pages 96–102, 2011.

[71] Nicola Poli. Game Engines and MAS: BDI & Artifacts in Unity. Master's thesis, Alma Mater Studiorum Università di Bologna, 2018.

[72] Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In Bernhard Nebel, Charles Rich, and William R. Swartout, editors, *3rd International Conference on Principles of Knowledge Representation and Reasoning (KR '92)*, pages 439–449, Cambridge, MA, USA, 25-29 October 1992. Morgan Kaufmann. Proceedings.

[73] Gavin Rens and Thomas Meyer. A hybrid pomdp-bdi agent architecture with online stochastic planning and desires with changing intensity levels. In Béatrice Duval, Jaap van den Herik, Stephane Loiseau, and Joaquim Filipe, editors, *Agents and Artificial Intelligence*, pages 3–19, Cham, 2015. Springer International Publishing.

[74] Alessandro Ricci, Mirko Viroli, Andrea Omicini, Stefano Mariani, Angelo Croatti, and Danilo Pianini. Spatial tuples: Augmenting reality with tuples. *Expert Systems*, 35(5):e12273, 2018. e12273 10.1111/exsy.12273.

[75] Sara Rodríguez, Dante I. Tapia, Eladio Sanz, Carolina Zato, Fernando de la Prieta, and Oscar Gil. Cloud computing integrated into service-oriented

multi-agent architecture. In Ángel Ortiz, Rubén Darío Franco, and Pedro Gómez Gasquet, editors, *Balanced Automation Systems for Future Manufacturing Networks*, pages 251–259, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[76] Gail Shaw and Etienne van der Poel. Genetic algorithms as a feasible replanning mechanism for belief-desire-intention agents. In *Proceedings of the 2015 Annual Research Conference on South African Institute of Computer Scientists and Information Technologists*, SAICSIT '15, New York, NY, USA, 2015. Association for Computing Machinery.

[77] Aaron XL Shen, Christy MK Cheung, Matthew KO Lee, and Huaping Chen. How social influence affects we-intention to use instant messaging: The moderating effect of usage experience. *Information Systems Frontiers*, 13(2):157–169, 2011.

[78] Enrico Siboni. *2p-Kt: A Kotlin-based, Multi-Platform Framework for Symbolic AI*. PhD thesis, Ingegneria e Architettura, 2019.

[79] D. G. Silva and J. C. Gluz. Agentspeak(pl): A new programming language for bdi agents with integrated bayesian network model. In *2011 International Conference on Information Science and Applications*, pages 1–7, 2011.

[80] Dhirendra Singh, Lin Padgham, and Brian Logan. Integrating bdi agents with agent-based simulation platforms. *Autonomous Agents and Multi-Agent Systems*, 30, 11 2016.

[81] Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Stéphane Airiau. Learning context conditions for bdi plan selection. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, page 325–332, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[82] Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Geoff James. Integrating learning into a bdi agent for environments with changing dynamics. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[83] Rosalvo Ermes Streit and Denis Borenstein. An agent-based simulation model for analyzing the governance of the brazilian financial system. *Expert Systems with Applications*, 36(9):11489 – 11501, 2009.

[84] L. Subramainan, M. A. Mahmoud, M. S. Ahmad, and M. Z. M. Yusoff. A conceptual emotion-based model to improve students engagement in a classroom using agent-based social simulation. In *2016 4th International Conference on User Science and Engineering (i-USEr)*, pages 149–154, 2016.

[85] Katia P Sycara. Multiagent systems. *AI magazine*, 19(2):79–79, 1998.

[86] Y. Sánchez, T. Coma, A. Aguelo, and E. Cerezo. Applying a psychotherapeutic theory to the modeling of affective intelligent agents. *IEEE Transactions on Cognitive and Developmental Systems*, 12(2):285–299, 2020.

[87] Ah-Hwee Tan, Yew-Soon Ong, and Akejariyawong Tapanuj. A hybrid agent architecture integrating desire, intention and reinforcement learning. *Expert Systems with Applications*, 38(7):8477 – 8487, 2011.

[88] Cecilia Sosa Toranzo, Marcelo Errecalde, and Edgardo Ferretti. On the use of agreement technologies for multi-criteria decision making within a bdi agent. In Ana L.C. Bazzan and Karim Pichara, editors, *Advances in Artificial Intelligence – IBERAMIA 2014*, pages 54–65, Cham, 2014. Springer International Publishing.

[89] Mohammad H. Vahidnia, Ali Alesheikh, and Seyed Kazem Alavi Panah. A multi-agent architecture for geosimulation of moving agents. *Journal of Geographical Systems*, 17, 10 2015.

[90] Max Waters, Lin Padgham, and Sebastian Sardina. Evaluating coverage based intention selection. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '14, page 957–964, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[91] Max Waters, Lin Padgham, and Sebastian Sardina. Improving domain-independent intention selection in bdi systems. *Autonomous Agents and Multi-Agent Systems*, 29, 07 2015.

[92] Y. Weihong. Active guidance mechanism of university public opinion based on cbr and bdi agent. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 1307–1311, 2018.

[93] Vincent Wiegel and Jan Berg. Combining moral theory, modal logic and mas to create well-behaving artificial agents. *I. J. Social Robotics*, 1:233–242, 08 2009.

[94] M. Xu, Kim Bauters, Kevin McAreavey, and W. Liu. A formal approach to embedding first-principles planning in bdi agent systems. In *SUM*, 2018.

[95] Y. Zhang and W. Wu. Flight mission modeling based on bdi petri net. *Journal of Systems Engineering and Electronics*, 28(4):776–783, 2017.