Towards a Multi-Level Explainability Framework for Engineering and Understanding BDI Agent Systems

Elena Yan^{1,*}, Samuele Burattini^{1,*}, Jomi Fred Hübner^{2,*} and Alessandro Ricci^{1,*}

¹Alma Mater Studiorum, University of Bologna - Cesena Campus, Via dell'Universitá 50, 47521 Cesena (FC), Italy ²Federal University of Santa Catarina, 88040-900 Florianópolis (SC), Brasil

Abstract

Explainability is more and more considered a crucial property to be featured by AI-based systems, including those engineered in terms of agents and multi-agent systems. This property is primarily important at the user level, to increase e.g. system trustworthiness, but can play an important role also at the engineering level, to support activities such as debugging and validation. In this paper, we focus on BDI agent systems and introduce a multi-level explainability framework to understand the system's behaviour that targets different classes of users: developers who implement the system, software designers who verify the soundness and final users. A prototype implementation of a tool based on the JaCaMo platform for multi-agent systems is adopted to explore the idea in practice.

Keywords

BDI Agents, Explainable AI, Debugging

1. Introduction

Explainability emerged in recent years as an important desired property for AI-based systems [1]. Among these systems, autonomous agents and MAS play a prominent role [2]: explainability in that case mainly concerns the decisions and actions that agents autonomously make and perform in order to accomplish their tasks. The key concept is to provide a level of understanding of the behaviour through (usually text based) explanations that can better convey the decision making processes and enable users to validate the motivations of the choices taken by the AI system – being it an agent, a Convolutional Neural Network or a cutting-edge generative system.

Actually, explainability is a crucial property not only from a user perspective, but also from an *engineering* one, to support activities of developers and designers involved in debugging [3] and validation of the system behaviour [4, 5]. In this context, the model and architecture adopted for designing and programming agents can play a key role. In particular, high-level cognitive models/architectures such as Belief-Desire-Intention (BDI) [6] provide in principle a natural support to explainability *by design*. This has been clearly recognized in literature by different



WOA 2023: 24th Workshop From Objects to Agents, November 6-8, Rome, Italy

^{*}Corresponding author.

[†]These authors contributed equally.

[🛆] elena.yan@studio.unibo.it (E. Yan); samuele.burattini@unibo.it (S. Burattini); jomi.hubner@usfc.br (J. F. Hübner); a.ricci@unibo.it (A. Ricci)

D 0009-0000-6660-9378 (E. Yan); 0009-0009-4853-7783 (S. Burattini); 0000-0001-9355-822X (J. F. Hübner); 0000-0002-9222-5092 (A. Ricci)

^{© 02023} Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

approaches in the last two decade [7, 4], including works about exploiting explainability for debugging BDI agents [3, 8], and validating BDI agent behaviour [5].

Existing works based on BDI agent programming languages/platforms typically work on traces generated by the agent at runtime, and build explanations based on the specific operational semantics on which the specific agent programming language is based. The level of abstraction of the explanation is then the *one*, directly provided by the agent programming language adopted.

In this paper, we argue that explaining and understanding that behaviour and system could be important at *multiple levels of abstraction*, from developers working at the implementation level with e.g. some specific agent programming languages, up to designers and engineers that may want to abstract from details of the specific technologies used to implement agents focusing more at the architectural level, and finally the domain expert and user level, who focus on the functional and non-functional requirements of the system as a whole, abstracting from how the system has been built. In that perspective, in this paper we propose a *multi-level explainability framework*, a conceptual framework and a supporting prototype tool that make it possible to create *narratives* explaining the behaviour of an agent at multiple levels of abstractions, meant to be useful at different levels – developers, designers, users.

The remainder of the paper is organised as follows. In Section 2 we provide an overview of the conceptual framework, and then we discuss the three main levels that we identified: the implementation level (Section 3), the knowledge level (Section 4) and the domain level (Section 5). In Section 6 we describe a prototype implementation of a tool to generate such explanations, based on an existing MAS platform (JaCaMo [9]). Finally, in Section 7 we reflect on the open challenges to bring this approach to its full potential and future work.

2. Logs as Narratives to Explain and Understand - A Multi-Level Perspective

In this paper, we build on top of the idea of using logs to examine the behaviour of a software system by applying it to multi-agent systems with a novel angle which is to include multiple levels of explanation generated from the same set of logs. Commonly, explainability in agent systems is achieved focusing on a *single agent* that produces a *single explanation* for a *single purpose*. Our research introduces a different approach by presenting an explainability framework for agents and multi-agent systems that deals with multiple levels of abstraction that can be used with different purposes by different classes of users.

We start from the idea that explanations in the MAS community have been used either as support for comprehending the behaviour of a running system [10, 11] or for software engineering processes such as collaboration with designers and domain experts [12], debugging [8, 13, 14, 15, 16, 17], as well as testing and validation of agent behaviours [18, 19, 20, 12, 21]. All of these use cases require to deal with different levels of abstraction in the generated explanation since they target specific classes of users with different needs and objectives.

When developing explanations though, it is important that they are understandable and useful to users, the aim is to maximise the perceived usefulness and generate explanations that are user-friendly. These are key aspects that influence how users can benefit from the provided explanations. To achieve this, instead of providing explanations in a traditional technical format or presenting directly with the raw format of the log, explanations are often provided as text in natural language. In our approach we use the idea of a **narrative** to indicate a specific level of abstraction in which the explanation is provided. There are similar works in the literature that present the explanation in the form of dialogues [22] or stories at the agent level [20], system or user level [18, 19] to check the correctness of the system. The narrative is particularly effective in making the explanation accessible to a wider audience. Developers, designers, and users who are not fully familiar with the technical complexities of agent systems can easily understand the concepts and motivations when they are presented in a narrative, story-like form. It's interesting then to understand whether a unified approach can lead to better results, allowing users to switch between multiple possible levels of explanation when in need.

The first step to generate explanations is to collect data about the running system: a simple and valuable approach is derived from the system trace by a logging approach, such as in [3] and [15], where the authors export traces from the execution system and programmers can ask *why* questions for events that require explanations, following the idea of the *Whyline* approach [23].

Logging can be either *explicit* when the developer takes an active part in selecting which information are logged, or *implicit* when the overall execution platform captures and builds traces without external intervention. For our framework, we chose to rely on *implicit logging*, in order to be able to store, organise, and rely on the same kind of logs of agents behaviour to generate automatic explanations of the system's behaviour without bothering the developer with additional steps to be included in its code. On top of the data collected with this approach, we then generate different layers of explanation that target different classes of users with different purposes.

It is worth noticing that generally the level of explanation is more abstract when it deals with less technical users, and can be seen as hierarchically built on top of the lower ones. At the same time, in MAS engineering several dimensions are interesting when we move from a single agent to a multi-agent systems, namely the *interaction* dimension that concerns exchange of messages between agents, the *environment* dimension concerning the actions of multiple agents in a shared space and the *organisational* dimension which considers norms and social structures. Different levels of explanation are possible also considering these orthogonal dimensions, in order to target the specific dimension that better resonates with the user of the system. We refer to this approach as "**multi-level explainability**", its idea is represented in Figure 1, from the multi-level abstraction of the individual agent (Figure 1a) to multiple dimensions involving the multi-agent system (Figure 1b). The primary levels of abstraction identified in this study are as follows:

- **the Implementation Level** We start with a detailed narrative closely related to the Agent and Multi-Agent System at a technical level that facilitates the developers to understand the agent's behaviour. It could also be a good reference for the debugging and testing phases of the software engineering process.
- **the Knowledge Level** At this level, we propose using the Belief-Desire-Intention (BDI) model [6] to represent the cognitive aspect of agents. The BDI model is a valuable abstraction for explaining the agent's high-level decisions. BDI agents have *beliefs* their



(a) Multiple levels of abstraction in the agent dimension

(b) Multiple dimensions and levels of abstraction in a multi-agent system

Figure 1: The conceptual idea of multi-level explainability from multiple perspectives

knowledge about the world and themselves, *desires* – the objective that agents want to achieve, and *intentions* – agent's decision to perform actions for pursuing a particular goal. [24, 25] Through rational reasoning, the agent has a set of goals to achieve and uses its knowledge and beliefs to decide the actions to be taken. This is particularly conducive to the creation of a valid explanation at the knowledge level. Regardless of the specific implementation of the system. This level is a good abstraction for designers who want to comprehend the current behaviour system, to interact with other stakeholders and developers or to formulate new requirements for improving the system.

• **the Domain Level** The previous level can be extended to a higher one incorporating the domain-specific knowledge and insights. This vision provides a comprehensive understanding related to the system's high-level behaviour and requirements that involve end-users and domain experts. We believe this is the ultimate level of abstraction that can be reached, so that users that are expert in the domain can understand seamlessly the behaviour of the system as if it was described by other humans.

These abstraction levels are designed to be flexible, allowing different groups of users to access explanations from different perspectives. Each level of explanation includes all the relevant information necessary for a complete understanding of that specific abstraction. Consequently, these levels are considered fully *self-contained*. This means that all explanations, properties and tools necessary for the evaluation of behaviour at a given level are enclosed within that level, avoiding considering the details of other levels.

Explanations at higher levels of abstraction are built on top of explanations at lower levels. This modular process involves keeping or combining the most relevant explanations and presenting them in a more abstract, higher-level way, depending on the desired level of abstraction.

Although we include in the idea of multi-level explainability also the different dimensions of a MAS, in this first exploratory study we focused only on the Agent dimension. In the following

sections, the levels presented above will be explained in detail with examples of how they were defined for the Agent dimension and as such focusing on the explanation of the behaviour of each individual agent in the system.

3. Narratives for Developers - the Implementation Level

At the implementation level, the explanation of behaviour is presented in a very close way to the raw log produced with the system execution. It is a low, detailed, and technical level that follows the operational semantics of the programming language used to code the agents. The narrative at this level describes the entire history of the agent at a detailed level, including very technical aspects and the inner workings of the agent. The explanation of the agent can also be extended to a *multi-agent system* (MAS) perspective. Considering the fact that each agent is autonomous and operates with its own cycle and execution time, a multi-agent view requires associating a log trace for each agent in the system. For a more consistent and robust explanation, a log could also be associated with the environment and organisation. This narrative can be navigated by the developer and is an excellent tool for understanding intricate Agent and MAS code and for engineering phases such as debugging, validation, and testing.

In a debugging scenario, through the explanation of the agents' decisions and the motivation leading to such choices, the developer can follow the trace of decisions and actions and find the cause of the bug or failure in order to correct the problem effectively. In order to locate the cause of such behaviour, it is essential that the narrative explain *how* and *why* an event or action is occurring. Understanding the **reasons** behind certain decisions is extremely useful to help developers during the debugging phases. [3, 26] With an extension to the environment, the developer can check whether the problem is located in the agent's program or in the environment's program.

During the testing and validation phases, developers can use the explanation of the agents, take the narrative, and check the requirements to see if they meet them, or compare the narrative with the user stories of the system to ensure alignment. [19, 20]

Agent programming may be exploited even for classic cognitive agents and multi-agent systems that use logic-based languages such as *Jason* [27] for the agent side and *JaCaMo* [28, 9] for multi-agent systems with more dimensions. These agent programming languages have a higher level of abstraction centred on agents with a mental state and reasoning process consisting of concepts such as beliefs, goals, plans, intentions, actions, etc. In this case, the logging of the agent's behaviour follows its reasoning cycle events, providing a detailed trace of key *events* in the agent's decision-making process. It allows developers to keep track of every change in the agent's plan library and belief base, record every state of goals and intentions, register agents' decisions and actions in pursuit of their goals, and follow interactions with other agents in the multi-agent system.

We have identified the following types of events and their related narratives according to the concepts of *AgentSpeak* and *Jason*. Operating by the concepts, we can apply the same reasoning to another language and consider its semantics.

Belief Events – There are events that can change an agent's belief base, including the complete addition or removal of a belief or referring only to a source, a new perception from the environment, or a new message from another agent.

- Belief Added: "Added belief [belief]"
- Belief From Src Added: "Added belief [belief] from source source"
- Belief From Src Removed: "Removed belief [belief] from source source"
- Belief Removed: "Removed belief [belief]
- New Percept: "New [type] percept [percept] from [source]"
- New Speech Act Message: "New speech act message [type] from [source]: [message]"

Goal Events – Events related to a goal are based on their current state and the agent's commitment to achieving them. The initial state of a goal is *pending*, denoting that it is essentially a desire at this point. If an applicable plan is discovered, the goal becomes an **intention** and the agent fully commits to pursuing it, the state then moves to *executing*. Then, because of some events, the state can be waiting or resumed. Finally, a goal may conclude its lifecycle by moving into one of the three terminal states: *dropped* - by special internal action, *achieved* - successfully completed or *failed* - if there is no plan or by internal action.

- Goal Created: "Goal [goal] created, state: pending"
- **Plan Selected:** "Plan [goa1] selected, state: executing" when the agent selected a plan for committing the goal, the goal becomes an intention, and from there an intention event is created, and we can have these events:
 - Intention Created: "Intention [id] created, state: running, current step: [current_intention_step]"
 - Intention Waiting: "Intention [id] waiting because [reason], state: waiting, current step: [current_intention_step]"
 - Intention Suspended: "Intention [id] suspended because [reason], state: suspended, current step: [current_intention_step]"
 - Intention Removed: "Intention id removed because reason, state: undefined"
- Goal Suspended: "Goal [goa1] suspended because [reason]"
- Goal Removed: "Goal [goal] removed because the goal is [achieved/dropped/failed]"

Other *Jason* **events** Other significant events in *Jason* that contribute to the representation of the agent's behaviour are:

- Reasoning Cycle Started: "New reasoning cycle started: [number]"
- Plan Events events that refer to change in the agent's plan library:
 - Plan Added: "Plan [plan] added to the plan library"
 - Plan Removed: "Plan [plan] removed from the plan library"

- Action Events actions that the agent executes for pursuing its goal:
 - Internal Action Finished: "Internal action [action] finished"
 - External Action Triggered: "External action [action] triggered"
 - External Action Finished: "External action [action] executed"
 - Executed Deed: "Deed [deed] executed, type: [deed_type]"
- Speech Act Messages events relating to the exchange of messages with other agents:
 - Mailbox Messages: "Messages in mailbox: [type] message from [agent]: [message], [type] message from [agent]: [message], ..."
 - Selected Message: "Selected message: [message]"
 - New Speech Act Message: "New speech act message [type] from [agent]: [message]"
 - Send Message: Send [type] message to [agent]: [message]

4. Narratives for Designers - the Knowledge Level

The implementation level has too many technical and code-related aspects, and its narrative is not particularly suitable for users who are not familiar with programming. Designers, for example, who want to understand and improve the system are more interested in the aspects of the system and the agent than the programming aspects related to the specific language. This is the motivation for the need for a higher and more abstract level.

A good level of abstraction is the cognitive aspect of the agent, which we can exploit the Belief-Desire-Intention (BDI) abstraction [6] to represent at the knowledge level. This model is advantageous for generating explanations because it relies on concepts that closely resemble those used by humans to explain their actions. [5] These mentalistic explanations can follow human intuitions, as the agent's behaviour is attributed to the beliefs, desires, and intentions present in the system at a given time. Specifically, the agent has a set of *desires* that he wants to satisfy. Based on his observations of the environment and the features he perceives, the agent updates its *beliefs* and finds a more appropriate and favourable plan that enables him to pursue some desire. The agent then commits to an *intention* that the agent follows by performing actions that should leave to the desire's satisfaction. [24, 29, 25]

The narrative at this Knowledge level is centred around these concepts and reported in *first person* to make it more expressive, as if the agents themselves were narrating their experiences and thought processes. The explanation is more vivid and accessible, allowing users to enter into the agent's reasoning and possibly identify themselves with the agent in that specific situation as we do with other human beings. The events we have identified at this Knowledge level are shown below.

Belief Events For belief, we have identified three main events related to its possible operations: adding, removing, and updating.

- New Belief: "I believe [belief] because [reason]".
- Belief Removed: "I no longer believe in [belief]".
- Belief Updated: "I update the belief [old_belief] to [new_belief]".



Figure 2: Example of the connection of the Knowledge and Implementation levels by the mapping of events of the two levels

Desire Events Desire events follow their own life cycle: creation, commitment, satisfaction, or dropped.

- New Desire: "I have a new desire [desire] because it is an initial desire".
- **Desire Committed/New Intention:** "I committed to desire [desire], and it became a new intention [intention]".
- Desire Satisfied: "I have satisfied my desire [desire] because its intention [intention] has finished".
- **Desire Dropped:** "I gave up desire [desire] because its intention [intention] failed".

Intention Events Intention represents a commitment or plan formed by an agent to achieve a specific desired outcome.

- **Desire Committed/New Intention:** "I committed to desire [desire], and it became a new intention [intention]".
- Executing Action: "I executed action [action] because of intention [intention]".

The proposed narrative is simple and concise, but at the same time, it provides all the information needed to understand the agent's behaviour at the Knowledge level.

In our current research, we are focusing on the *Jason* technology. Given the events at the implementation level and knowing the operational semantics of *Jason* and AgentSpeak, we can devise some rules for patterns of events that can be mapped into a new event at the knowledge level. An illustrative example of this event mapping is presented in Figure 2. For instance, a Desire Committed event at the Knowledge level, in which the desire is chosen and the agent commits to it, is grounded by Select Plan, Plan Selected, and Intention Created events at the implementation level. While a Belief Updated event is associated with Belief Removed and Belief Added events at the implementation level. Such a mapping is crucial to build the multi-level explanation, we argue that given any implementation level, given a mapping it would be possible to generate a knowledge level explanation of the system, with the benefits that a BDI-based explanation has from a user understanding perspective.

Knowledge Level	Implementation Level
New Belief	Belief Added
"I believe <i>b</i> because I annotated it in my mind for future reference"	"Added belief <i>b</i> "
	New Percept
"I believe <i>b</i> because I perceived it from <i>c1</i> "	"New percept from <i>c1</i> : <i>b</i> "
	Belief Added
	"Added belief <i>b</i> "
	New Speech Act Message
"I believe <i>b</i> because <i>Alice</i> told me"	"New speech act message <i>tell</i> from <i>Alice</i> : <i>b</i> "
	Belief Added
	"Added belief <i>b</i> "
Belief Removed	Belief Removed
"I no longer believe in <i>b</i> "	"Removed belief <i>b</i> "
Belief Updated	Belief Removed
"I update the belief $b(1)$ to $b(2)$ "	"Removed belief <i>b(1)</i> "
	Belief Added
	"Added belief $b(2)$ "
	Audeu bellet b(2)

Table 1

Correspondence of Belief Events in Knowledge and Implementation level

It is important to also note that these events may not necessarily occur sequentially, there can be interleaving events, such as a new perception of changes in the environment.

To provide a more detailed overview of this mapping process, Table 1 illustrates the mapping of Belief related events from the Implementation level to the Knowledge level, while Table 2 illustrates the mapping of Desire and Intention related events.

5. Narratives for Users - the Domain Level

Increasing the abstraction further, we are abstracting from *how is done*, but we are focusing on *what we want*, specifically, we don't see how the system is behaving, we see what are the results. From the user's perspective, we need to have an even more abstract narrative on top of the knowledge level that concerns the *requirements* and *domain aspects* of the system. At this level, the narrative needs to be related to the domain specifics, similar to user stories [18], in a way that is more related to the user level. End-users or domain experts can explore this narrative to comprehend the system from a domain perspective. The narrative at this level is easy to understand and more accessible to users, even without being familiar with agent technologies or knowledge abstractions. So when the user asks for an explanation of why the

Knowledge Level	Implementation Level
New Desire	New Goal Created
"I have a new desire g because it is an initial desire"	"Goal g created"
"I have a new desire g2 because it is a desire created from g"	"Goal g2 created"
Desire Committed/Intention Created	SelectPlanEvent
"I committed to desire g, and it became a new intention 1 g"	"Plan options for g are [] The plan se- lected for g is g : (count(X) & (X < 3)) <- a1; !g2; a3."
	PlanSelected
	"Plan g selected"
	Intention Created
	"Intention 1 g created, state: running"
Executed action	Intention Created
"I'm executed action <i>a1</i> because of intention 1 <i>g</i> "	"Intention 1 g created, state: <i>running</i> , current step: <i>a</i> 1
	External Action Finished / Internal Action Finished / Execute Deed
	"External action / Internal Action / Deed <i>a1</i> exe- cuted"
Desire satisfied	Intention Removed
"I have satisfied my desire g because its inten- tion 1 g finished"	"Intention 1 removed, state: <i>undefined</i> "
	Goal Removed
	"Goal g removed because the goal is achieved"
Desire dropped	Intention Removed
"I gave up desire g because its intention 1 g failed"	"Intention 1 removed, state: <i>undefined</i> "
	Goal Removed
	"Goal g removed because the goal is failed"

Table 2

Correspondence of Desire and Intention Events in Knowledge and Implementation level

system does something, the answers should be related to the system's requirements and justified in terms of the user story and requirements, instead of the agent's desire or belief. For building explanations at this level, we may need to infuse knowledge about the description of the system requirements that make it possible to design the agents, and given the behaviour of agents at the knowledge level, we should be able to derive why some actions are done in terms of user and domain level. We need some formalisation of the use cases, and the user stories, this could



Figure 3: Overview of the main components of the framework

be exploited by works in [18, 20].

For domain experts, this level of abstraction can support them in collaborating, reducing the gap with developers [12], and verifying whether the developed system is consistent with the domain specification. For end-users, this narrative can help them understand the system, improving trust and transparency of the system. [30, 2]

This level is still being discussed and it is a future direction of the multi-level explainability framework.

6. Prototype Implementation

To put our conceptual framework into practice, we have experimented with the development of a prototype tool for this multi-level explanation idea, using the operation semantics of *Jason* as our starting point.

The main components of the multi-level tool that we built in this project are represented in Figure 3. Since the approach taken for creating explanations is through logging, an important component of our tool is the **logging component**, which is responsible for generating the execution trace for each entity in the system. This logging component captures events and useful information from the system execution at the lower level and then stores them in *file-based* formats that serve as the foundation for generating explanations.

A model component then includes the **explanation** generation algorithm, which processes the log and generates an explanation and narrative of the system's behaviour. This explanation is presented to the user through a *web application*. Different users, such as developers, designers, or end users, can access the application and explore the narrative at different levels for understanding and analysis.

Logging component For developing the logging component¹ we have extended the *Jason* interpreter with customisation agent architecture that allows logging functionality. This cus-

¹The logging component prototype is available at https://github.com/yan-elena/agent-logging, an example of utilisation is presented at https://github.com/yan-elena/domestic-robot-example.



Figure 4: Overall architecture of the logging component

tomisation primarily focuses on the Agent class and the AgArch, which have been extended respectively by the classes LoggerArch and LoggerAg.

The overall architecture of the logging component is presented in Figure 4. Specifically, the central element of this structure is the EventLogger, which orchestrates the management of logs from all agents in the system. It is responsible for publishing the events associated with a specific agent and subsequently saving the log to a file.

The workings of the EventLogger class exploit a map-based data structure, which houses the event history of all agents. The individual log of a given agent is represented by the Agent History component. The interface of this component provides methods for including events and storing the log. Events are encapsulated by the corresponding Event interface, which provides a uniform structure to represent different types of events. Every event at the Implementation Level outlined in 3 is instantiated as a class that implements the Event interface, adapting its structure to the specific characteristics of the event type.

Explanation component The *explanation* component² is responsible for presenting the explanation of the agent system on multiple levels in a comprehensible and user-friendly manner. We opted to develop a **web application** for its accessibility and ease of use. This web application serves as an interface for users to interact with and understand the complex reasoning and behaviour of the multi-agent system. Users can navigate explanations, view event narratives, and select the level of abstraction of the narrative.

The application provided functionality to upload the log files generated by the *logging component*. With these log files, the application generates a comprehensive multi-level explanation

²The explanation component prototype is available at https://github.com/yan-elena/agent-explanation, a deployed application can be accessed directly via this link: https://yan-elena.github.io/agent-explanation.



Figure 5: Web application - Narrative at the Knowledge level of the desire *get(beer)* of the agent *owner* in the classic example of the *domestic robot*

following the mapping identified in the design phases. When a user selects an agent for explanations, the application provides a view of all event narratives at the selected level. Figure 5 shows a narrative for the classic example of the *domestic robot*³ described in the *Jason* book [27]. The explanation of the desire get (beer) for the *owner* agent is shown at the Knowledge level.

In the application, there is also a component for *filtering* the events of interest, this functionality is very useful for visualising the entire life cycle of a particular event. In this case, we have shown the desire get(beer), and by filtering events, its lifecycle is immediately shown. In fact, we can see that the agent has an initial desire, then decides to commit it, actions are performed to achieve the desire, and in the end, it is satisfied.

7. Conclusion and Future Work

In this paper, we presented this new idea of *multi-level* explanation, identified three possible levels, and which relevant events are useful for generating the explanation while demonstrating how it is possible to generate and map these events from one level to the next, taking as a reference technology *Jason*.

In the current state, the implemented explainability tool can be used as a basis for comprehending multi-agent systems. However, we present below some valuable suggestions and possible directions for research to further improve the framework.

The first main direction is to move towards the domain level that is still in the discussion phase, as described in Section 5. We argue that to build this further level, we should include formal

³The example is available on GitHub, configured to be used with the logging component at https://github.com/ yan-elena/domestic-robot-example

descriptions of system requirements, use cases, and system stories for additional information about the domain. The narrative here should provide explanations at the system level and focus on the system requirements and objectives at the domain level.

In addition to expanding the level of abstractions, another direction is to expand the dimensions of explanation regarding multi-agent systems, encompassing both *environment* and *organisation* aspects, as cited in Section 2. This expansion will involve integrating dedicated logging components tailored to environmental artefacts and organisational specifications. We can also apply the idea of multi-level explanation to these dimensions. For example, a low-level explanation of the organisation can refer to the implementation and technical aspects, while an explanation at the knowledge level could include specifications of objectives and missions.

Another direction that needs to be improved in order to achieve a more robust framework is the integration of *cause-effect* relationships to sequences of events [26]. The complexity lies in identifying the relationships between the various events, *linking* them and creating causal chains. This feature further improves the explanation of the reason behind certain decisions and allows the root cause to be traced in a robust manner which is extremely useful both for debugging and improving the understanding of the system from a user point of view.

References

- [1] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, J. Zhu, Explainable ai: A brief survey on history, research areas, approaches and challenges, in: Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8, Springer, 2019, pp. 563–574.
- [2] A. Malhi, S. Knapic, K. Främling, Explainable agents for less bias in human-agent decision making, in: D. Calvaresi, A. Najjar, M. Winikoff, K. Främling (Eds.), Explainable, Transparent Autonomous Agents and Multi-Agent Systems, Springer International Publishing, Cham, 2020, pp. 129–146.
- [3] K. V. Hindriks, Debugging is explaining, in: I. Rahwan, W. Wobcke, S. Sen, T. Sugawara (Eds.), PRIMA 2012: Principles and Practice of Multi-Agent Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 31–45.
- [4] J. Broekens, M. Harbers, K. Hindriks, K. Bosch, C. Jonker, J.-j. Meyer, Do you get it? user-evaluated explainable bdi agents, in: Multiagent System Technologies: 8th German Conference, MATES 2010, Leipzig, Germany, September 27-29, 2010. Proceedings 8, volume 6251, Springer, 2010, pp. 28–39. doi:10.1007/978-3-642-16178-0_5.
- [5] M. Winikoff, G. Sidorenko, V. Dignum, F. Dignum, Why bad coffee? explaining bdi agent behaviour with valuings (extended abstract), in: L. D. Raedt (Ed.), Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 5782–5786. URL: https://doi.org/10.24963/ijcai.2022/810. doi:10.24963/ijcai.2022/810, journal Track.
- [6] M. Bratman, Intention, Plans, and Practical Reason, Cambridge: Cambridge, MA: Harvard University Press, 1987.
- [7] M. Harbers, K. van den Bosch, J.-J. Meyer, Design and evaluation of explainable bdi agents,

in: 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, volume 2, 2010, pp. 125–132. doi:10.1109/WI-IAT.2010.115.

- [8] M. Winikoff, Debugging agent programs with why? questions, in: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS '17, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2017, pp. 251–259.
- [9] O. Boissier, R. H. Bordini, J. Hübner, A. Ricci, A. Santi, Multi-agent oriented programming with JaCaMo, Science of Computer Programming 78 (2013) 747–761. URL: http://www. sciencedirect.com/science/article/pii/S016764231100181X. doi:10.1016/j.scico.2011. 10.004, special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.
- [10] T. Bosse, D. Lam, K. Barber, Tools for analyzing intelligent agent systems, Web Intelligence and Agent Systems 6 (2008) 355–371. doi:10.3233/WIA-2008-0145.
- [11] D. N. Lam, K. S. Barber, Comprehending agent software, in: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 586–593. URL: https: //doi-org.ezproxy.unibo.it/10.1145/1082473.1082562. doi:10.1145/1082473.1082562.
- [12] Á. Carrera, C. A. Iglesias, M. Garijo, Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development, Information Systems Frontiers 16 (2014) 169–182.
- [13] T. Ahlbrecht, An algorithmic debugging approach for belief-desire-intention agents, Annals of Mathematics and Artificial Intelligence (2023) 1–18. doi:10.1007/ s10472-023-09843-4.
- G. Pothier, E. Tanter, Back to the future: Omniscient debugging, IEEE Software 26 (2009) 78-85. doi:10.1109/MS.2009.169.
- [15] V. J. Koeman, K. V. Hindriks, C. M. Jonker, Omniscient debugging for cognitive agent programs, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 265–272. URL: https://doi.org/10.24963/ijcai.2017/38. doi:10.24963/ijcai.2017/38.
- [16] D. Poutakidis, L. Padgham, M. Winikoff, Debugging multi-agent systems using design artifacts: The case of interaction protocols, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2, AAMAS '02, Association for Computing Machinery, New York, NY, USA, 2002, pp. 960–967. URL: https://doi.org/10.1145/544862.544966. doi:10.1145/544862.544966.
- [17] M. Dastani, J. Brandsema, A. Dubel, J.-J. C. Meyer, Debugging bdi-based multi-agent programs, in: L. Braubach, J.-P. Briot, J. Thangarajah (Eds.), Programming Multi-Agent Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 151–169.
- [18] S. Rodriguez, J. Thangarajah, M. Winikoff, User and system stories: An agile approach for managing requirements in aose, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '21, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2021, pp. 1064–1072.
- [19] S. Rodriguez, J. Thangarajah, M. Winikoff, D. Singh, Testing requirements via user and system stories in agent systems, in: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, 2022, pp. 1119–1127.

- [20] S. Rodriguez, J. Thangarajah, M. Winikoff, A behaviour-driven approach for testing requirements via user and system stories in agent systems, in: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, 2023, pp. 1182– 1190.
- [21] E. E. Ekinci, A. M. Tiryaki, Ö. Çetin, O. Dikenelli, Goal-oriented agent testing revisited, in: M. Luck, J. J. Gomez-Sanz (Eds.), Agent-Oriented Software Engineering IX, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 173–186.
- [22] L. A. Dennis, N. Oren, Explaining bdi agent behaviour through dialogue, Autonomous Agents and Multi-Agent Systems 36 (2022) 29.
- [23] A. J. Ko, B. A. Myers, Debugging reinvented: Asking and answering why and why not questions about program behavior, in: Proceedings of the 30th International Conference on Software Engineering, ICSE '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 301–310. URL: https://doi.org/10.1145/1368088.1368130. doi:10.1145/ 1368088.1368130.
- [24] M. Wooldridge, Introduction to MultiAgent Systems, John Wiley and Sons, 2002.
- [25] A. S. Rao, M. P. Georgeff, Bdi agents: From theory to practice, in: International Conference on Multiagent Systems, 1995.
- [26] S. M. Khan, M. Rostamigiv, On explaining agent behaviour via root cause analysis: A formal account grounded in theory of mind, in: Proceedings of the 26th European Conference on Artificial Intelligence ECAI, 2023, pp. 30–09.
- [27] R. Bordini, J. Hübner, M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak Using Jason, volume 8, John Wiley & Sons, 2007. doi:10.1002/9780470061848.
- [28] O. Boissier, R. H. Bordini, J. Hubner, A. Ricci, Multi-agent oriented programming: programming multi-agent systems using JaCaMo, Mit Press, 2020.
- [29] R. Kowalski, F. Sadri, From logic programming towards multi-agent systems, Annals of Mathematics and Artificial Intelligence 25 (1999) 391–419. URL: https://doi.org/10.1023/A: 1018934223383. doi:10.1023/A:1018934223383.
- [30] D. Branley-Bell, R. Whitworth, L. Coventry, User trust and understanding of explainable ai: Exploring algorithm visualisations and user biases, in: M. Kurosu (Ed.), Human-Computer Interaction. Human Values and Quality of Life, Springer International Publishing, Cham, 2020, pp. 382–399.