Towards Explainable BDI Agents for End-users

Marcel Mauri[0000−0002−4135−1945] and Mirjam Minor[0000−0002−6592−631X]
{mauri, minor}@cs.uni-frankfurt.de

Department of Computer Science, Goethe University Frankfurt, Germany

Abstract. Explainable agency (XAg) aims at providing users with insights about the reasoning and decisions taken by an agent. Most of the newer XAg approaches are particularly useful as explanations for developers and researchers. In contrast, our novel XAg framework presented in this paper intends to answer explanation demands of end-user, including domain experts and lay users. It is a challenging task since this kind of users is not familiar with the methodological and technical aspects of agency. We propose a representation for end-user questions and potential explanatory answers in both a verbal and a formal description as well as a mapping structure of questions to multiple possible explanations. We develop a pattern-based approach to extract explanatory content from an execution log and to validate potential answers to a user question which is based on the TriQPAN decision patterns from the literature [15]. We organize the novel concepts in a four-layered architecture with layers for end-user questions, validation logic, TriQPAN patterns, and answer text generation. A running sample from a Jadex-BDI project on autonomous mobility on demand provides a demonstration scenario to illustrate some data structures and pseudocode. Further, it highlights the plausibility of our novel XAg framework.

Keywords: BDI Agent, XAI, XAg, BDI-ABM Framework, TriQPAN, Traffic Simulation, Jadex, Agent Development Framework

1 INTRODUCTION

Agent-oriented Programming (AOP) [16] has a rich research tradition of implementing intelligent behavior in complex environments. The decisions of cognitive agents are transparent and well explainable by established notions of agenthood in AI, such as beliefs, desires, intentions, plans or norms. The state of an agent can be 'read' directly from its data structures. Thus, Bordini et al. [3] consider BDI-based approaches [13] per se as *Explainable AI (XAI)* and argue that the intelligibility of agent behavior by end-users and other stakeholders is their key-contribution. However, with the advent of larger amounts of data and more complex decision processes involved in modern agent approaches, this transparency claim does not fully hold any more.

Recently, the field of *explainable agency* (XAg) has evolved [1, 6, 15, 9, 20, 21]. XAg describes the ability of agents "to explain their decisions and the reasoning that produces their choices" [6]. Today, the main target groups of XAg

approaches are researchers and developers who have a sound scientific and technological understanding. Ribera and Laprediza [14] categorize explainees of AI systems in three main groups, namely developers and AI researchers, domain experts, and lay users. The end-user perspective (domain experts and lay users) has not yet been addressed in depth in XAg. There is especially a lack of methods to communicate generated explanations to lay users. Mualla et al. [9] have published some preliminary work on enriching visual simulations of BDI agent behaviour by summarized beliefs and alerts. Yan et al. [21] recognize that explanations of agent-based systems can be used by different kinds of users but do not yet achieve domain level explanations for end-users in their prototypical framework. There is a research gap on developing XAg methods for end-users, including their evaluation in real application scenarios. It is a challenging research topic to bridge the gap between the explanations demands from an end-user perspective and the agent decisions that have been developed from an agent design and problem-solving perspective.

In this paper, we introduce a novel framework for XAg that takes the end-user perspective into consideration. It builds on TriQPAN design patterns [15] for recording agent decisions in an execution trace. The TriQPAN design patterns are particularly useful for expert users (MAS developers) as explanations. In our model, we integrate them with additional layers for the end-user perspective. This includes a mapping between the user's information needs and potential answers, a validation layer for (multiple) potential answers, and a verbalization layer with different degrees of granularity for communicating the explanations to the end-users. The novel framework is demonstrated by means of a running sample in a Jadex-BDI environment. A fully functional implementation in Java is currently under development. The sample explanation scenario is taken from an Autonomous Mobility on Demand (AMoD) project called *ATRIAS* where a fleet of autonomous e-trikes is simulated [8]. The demonstration serves as a proof-of concept for the novel representation forms and validation mechanisms.

The main contributions of this paper are threefold:

- To develop an *XAg framework targeting end-users*. It integrates an existing design pattern approach for expert users with the aim to let also lay users get insights about the reasoning of an agent.
- To integrate TriQPAN with Jadex agents in a *preliminary implementation concept*.
- To demonstrate a further *application scenario for TriQPAN with typical decisions of a fleet of autonomous e-trikes*.

The remainder of this paper is structured as follows: Section 2 presents the related work. Necessary background is covered in Section 3. The concept design of our explanation model is introduced in Section 4. In Section 5, we present a first feasibilty check of our framework. Using an example, we first show the necessary representation forms to capture all information for future explanations. Then we show how that information can be extracted and validated for generating

explanations. Further typical end-user questions will be explained by the means of a table with several examples.

Section 6 draws a conclusion and discusses future work.

2 RELATED WORK

Langley et al. [6] describe elements of explainable agency (XAg) in a position paper. The elements comprise representation forms for content that supports explanation, an episodic memory of target agents, as well as methods to access and extract content from episodic memory. Our approach has been inspired by this work in the sense that it uses execution traces from a simulation as episodic memory. Events and decision patterns will provide a structured form of representation in this episodic memory, allowing relevant content to be retrieved and extracted for explanation.

Anjomshoae et al.'s [1] literature review discusses application scenarios, main drives, social science and psychological background, platforms and architectures, explanatory granularity, presentation and evaluation of XAg. The authors state that most of the studied works either lack evaluations or conduct a user study for relatively simple scenarios. The findings provide a further incentive for our intended work on developing methods for the end-user perspective in XAg, including their evaluation.

Mualla [9] provides visual explanation also for end-users of a parcel delivery service. However, the explanations are still at an atomic level which makes it rather difficult to grasp the relation to the user's demand for information.

There is a body of XAg work on BDI agents where the agents inform human observers about their internal reasoning such as intentions [5, 10], recent actions [4], or decision processes [15, 19, 20]. Some of the work consider building blocks for explanations with a formal setting. Dennis & Oren's framework [4] uses predicate dictionaries to provide natural language substitutes in semi-formal explanations for domain experts with technical expertise. Winikoff et al. [20] generate explanations of the behavior of BDI agents from goal trees. A goal tree is a tree of nodes, where leaves are actions, and inner nodes are goals that can be decomposed using AND, SEQ or OR. The children of an OR decomposition are options that are selected at run time based on valuings, that means which outcome the agent prefers most in the current situation. The valuings approach has been evaluated in a sandbox scenario with end-users to assess the believability, acceptability, and comprehensibility of explanations [19]. Rodriguez et al. [15] describe design patterns for developing explainable-by-design agents. The aim is to explain the agent's reasoning and decision processes based on patterns that have a well-defined structure called TriQPAN. Since our approach uses TriQPAN patterns as a formal setting they are described in more detail below. As an extension to the design approach discussed in the literature [15], our work has a scope on the explanatory demands of end-users in an application scenario with real world data.

The issue of creating explanatory narratives for end-users of BDI agents has been discussed in the literature [21] by means of a domestic robot running example. It is part of a multi layered framework for different user perspectives. Two perspectives namely the implementation view and the BDI design view are formalized and prototypically implemented. The third layer, the domain view, has not yet been formalized and implemented.

3 BACKGROUND

3.1 ATRIAS system

ATRIAS [8] for *Autonomous trikes as a service* is a framework that connects BDI agents, implemented with *Jadex* [11] with the traffic simulation platform *MATSim* [2]. It was built upon the BDI-ABM interface [17]. Every Jadex vehicle agent is assigned to an agent in MATSim where the Jadex agents act as the decision making component (brain) where the MATSim counterpart is limited to the execution of actions and perception in the simulation environment (body). This allows complex reasoning capabilities to be combined with a feature-rich simulation platform. The intended scope of ATRIAS is all types of AMoD scenarios including ride-hailing, last-mile delivery or waste disposal logistics. The main focus is on the name giving autonomous trikes which process incoming trip requests on demand. Within ATRIAS, the area of operation is divided into several sub-areas, each with its own *area agent*. Incoming customer requests are sent by the area agents to the *vehicle agents* located closest to the customer in their area. These vehicle agents are designed to work in a decentralized manner for easier scalability and are completely self managed. Following the initial allocation of a customer request, an evaluation is conducted in order to calculate a utility score. The purpose of this is to determine how well an agent is suited to execute this request. If it will be below a certain threshold vehicle agents will use the *contract-net-protocol* (CNP) [18] to try to find a better suited agent for that customer request. When their batteries are low they will also drive to charging stations on their own.

The decision logic of the vehicle agents is designed following the BDI paradigm. Several goals have been specified to handle incoming customer requests. Beliefs about incoming customer requests are stored inside the `DecisionTaskList`. The goal `ManageJobs` processes the `DecisionTasks` depending on their current status and decides about the next action to be executed (evaluate it, negotiate with other agents to find a more suitable vehicle for execution, commit to it, etc.). When the status of a `DecisionTask` is set to 'commit' it will cause the creation of a `customerTrip` which will be stored alongside all other scheduled trips inside the `TripList`. The goal `BatteryLoaded` watches the current battery level and can create a `chargingTrip`. The trips inside the `TripList` then will be executed by the goal `TripService` which sends the corresponding drive operations to MATSim.

ATRIAS has been designed as a framework to simulate AMoD scenarios and to test and evaluate agent behavior. The plan is to port the ATRIAS agents

to control a fleet of autonomous trikes that will be operating on the campus of Goethe University Frankfurt. When the development of the ATRIAS framework started, XAg was not part of the project. Therefore, a possible future integration of XAg methods was not considered when designing the agent architecture. This makes it an interesting test case to show how an XAg model designed for end users can be adapted to such a non-optimized architecture.

3.2 TriQPAN patterns

TriQPAN [15] (Trigger, Query, Process, Action and Notify) are XAg design patterns that can be used to explain the behavior of agents to expert users like agent developers. In this paper we want to use TriQPAN as a part of an explainability model to extend the scope of applicability to end-users. TriQPAN processes are designed to work with an underlying event store, a log database in which all related events are stored. It has recently been implemented directly into the Sarl programming language [15] in which the capturing of these logs is directly integrated. The use of the pattern itself is not limited to a specific agent language or architecture. To apply the pattern, it is necessary to adapt it to the respective agent architectures specific processes. Therefore these patterns have to be modeled manually.

Every TriQPAN process starts with a `Trigger`. A `Trigger` might comprise perceptions, an update of a belief or the activation of a goal or plan. During the `Query` step the agent retrieves all information needed to execute an action. The `Action` step contains all initiated processes. These can relate both to belief updates as well as movements of the agent within the environment. The final step `Notify` will list all the changes made during the `Action` step and inform the used components which can trigger other TriQPAN patterns. After a TriQPAN process is completed, all the information it contains is captured within an XAgentProcess and stored together with the other logs generated at runtime at the event store.

Explanation approaches like those using TriQPAN patterns are well suited to explain how an agent's decision was made. This makes them a good tool for agents developers. Since they know the agent architectures and their reasoning mechanisms well, they can directly ask for agent internals such as goals/plans to get the information they need. They are also able to understand answers containing technical vocabulary and extract the information they need from them.

To make them better usable also for end-users (lay users), there are some research gaps that have to be closed. First end-users of agents do not possess knowledge about the internals of an agent. A question asked by an end-user does not necessarily stand in a direct relation to an agent component/decision or can contain situation specific aspects. Another problem is that the generated outputs can contain too many technical, agent-specific vocabulary or would not directly relate to the question. Therefore, a transformation of the facts given by the patterns into an answer in everyday language is required.

4 XAG MODEL FOR END-USER ALIGNMENT

The novel XAg model addresses the information demands of end-users. The explanations shall be understandable without technical background knowledge. It aims to fill the gap between the direct explanation of the decisions of the agents and the more general questions raised by end-users. In a preliminary survey of potential end users of the ATRIAS system [12], users were confronted with a mobile application to interact with our ATRIAS framework. A brief evaluation of the user experience with the app revealed, among other things, a few questions they would like to ask the system. The questions which have been formulated are "Why is my trike late?", "What are alternative transport modes?" or "What are the CO2 savings when using the ATRIAS trikes instead of driving by car?".

Answering such questions requires mapping the user's query to appropriate patterns that allow explaining the rationale behind an agent's decision relevant to the question, validating their truth within the event store, and verbalising a textual explanation for the patterns in everyday language. Some of the questions require additional knowledge beyond the agent decisions, including the vehicle's consumption of electrical power, which might be recorded in the agents' belief base or even be retrieved from external sources. We have developed a model for the end-user questions that are explainable by means of the agents' decision. The model builds on the XAg framework TriQPAN to extract the information on the agents' decision behavior from an event store.

Figure 1 depicts the architecture of the novel model for XAg with an end-user alignment. It comprises four main vertical layers (from the right to the left): End-user questions, validation logics, TriQPAN patterns, and answer text generation. First, the end-user can select a question from a list of possible questions. The questions are formulated as texts using the vocabulary of potential end-users and are free from technical terms (that are only understood by agent experts). In the second step the selected question will be processed by the validation logic. This component will retrieve a list of possible answers fitting to the user question. These possible answers are based on the functionality of the agent. A possible answer to the initial user question can depend on multiple different decisions the agent has made in the past. These decision points are captured by the TriQPAN patterns designed for the agent and stored alongside changes made to beliefs inside the event store. The patterns in the event store can be reused to answer different user questions. This is because they can refer to the same patterns.

During this step, a full list of possible answers will be calculated. These can be independently validated. Every possible candidate answer will have a test criterion which will be validated by the use of TriQPAN patterns. Every agent will store all events and already occurred patterns inside an event store. The validation algorithm will then access the event store to validate the corresponding possible answers by means of pre-defined test criteria. The validation logic then returns a list of identified technical answers to the user's question, along with the corresponding events. In the final step an answer for the end-user will be generated. Therefore we will use an *LLM/RAG system* [7] to create an easy to understand answer which will contain all the information given by the validation

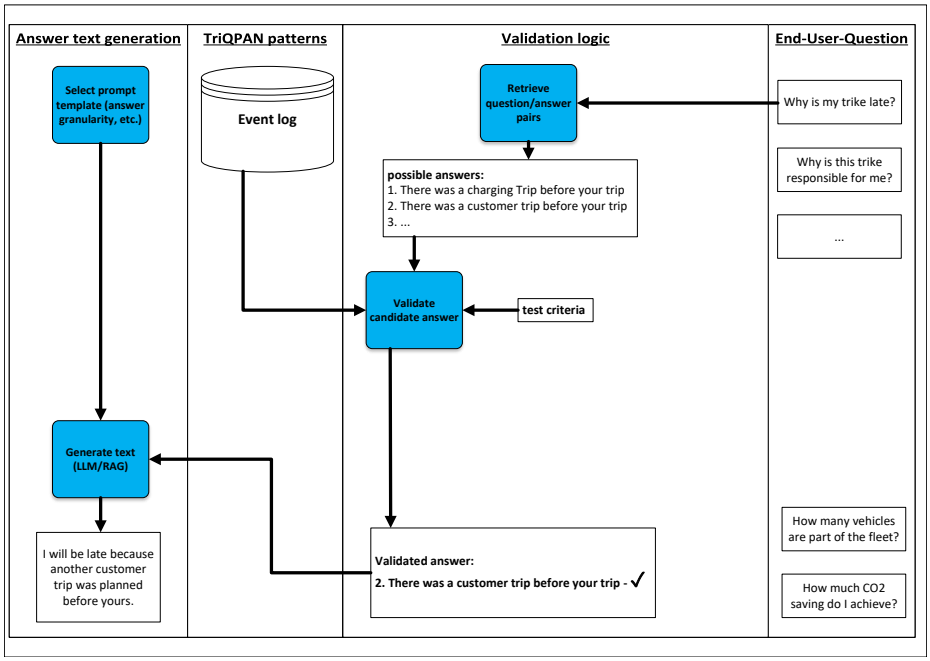

Fig. 1: Four-layered architecture of the novel XAg model.

logic. As customers may have different requirements for the level of detail of the answers, there is also the option to select a prompt template to specify the intended granularity of the answer.

5 XAG MODEL FEASIBILTY CHECK

In the following we will carry out a feasibilty check by discussing the data structures and pseudo code for a running sample within our ATRIAS framework. It will serve as a preliminary proof of concept of the model presented above.

We will use a typical sample question from end-users of our ATRIAS framework: "Why is my trike late?". Based on this question we will explain the measures necessary to log the information required. Then we will explain how we can extract the necessary information out of our logs to generate an answer. Finally, we will look at some of the other questions that end users may ask and the ways in which logs can be reused to generate answers for them.

5.1 Information recording

The question "Why is my trike late?" does not directly relate to an internal goal or plan which could give a direct answer to that question. To answer that question it has first to be connected to internal components from which the

necessary information can be extracted. The knowledge base of the XAg system comprises data structures for events and TriQPAN patterns that are assigned to the possible answers.

For the running sample in our ATRIAS scenario, a delay in the arrival of a vehicle can have various causes (possible answers). In the current state of our system, external factors like traffic jams are not yet considered. Thus, a delay is always caused by internal events or decisions made by the agent. In this scenario, the delay results from another trip that has been committed before the recent `customerTrip` but was not completed on time.

We have modeled two possible answers that can be validated independently from each other:

1. There is a `chargingTrip` before your trip, that does not finish in time.
2. There is a `customerTrip` before your trip, that does not finish in time.

A couple of events and TriQPAN patterns are assigned to each of the potential answers. In the following example, we will focus on the events and patterns for the case where another `customerTrip` has caused the delay (possible answer 2). This includes recording the previous `customerTrip` the agent is willing to serve, the agent's decision to commit to the current `customerTrip`, and the estimated time the previous trip will be completed by the agent. The two commit decisions for the previous and the recent customer can be recorded as two instances of the same pattern called `customerTripCreation`.

First, we define the trigger event for `customerTripCreation` named `Decision-TaskCommit`. The event will be logged everytime the status of a `DecisionTask` is set to 'commit'. It records the state before and after the change. These and other events relating to changes made to beliefs can be easily integrated into a Jadex agent. Therefore, every write access to a belief will be coupled with the execution of a log operation. The creation of the log has to be coupled to the method that creates new trips. Changes to the `TripList` will be captured by `TripList_BeliefUpdated` events (Fig. 2). For better understanding the content of the `TripList` is simplified. Every line in Fig. 2 represents a trip which contains the `TripID`, the `StartTime` and the currently expected `EndTime`. The latter is of special importance for the validation (see below). Its value can still change during future updates of the `TripList`.

```
TripList_BeliefUpdated = {
    Old value = [[Trip1, StartTime, Endtime, ...]]
    New value = [[Trip1, StartTime, Endtime, ...],
                 [Trip2, StartTime, Endtime, ...]]
}
```

Fig. 2: Log of the changes made to the `TripList` (simplified)

```
XAgProcess = {
    name = { customerTripCreation }
    Trigger = { DecisionTaskCommit }
    Queries = { DecisionTaskList }
    Criterion = { A DecisionTask which status equals commit
                  will cause the creation of a customerTrip }
    Actions = { Set DecisionTask.status = commited,
                Create new CustomerTrip = (TripID, StartTime,
                EndTime, ...),
    Notification = {DecisionTaskCommitted,
                    TripList_BeliefUpdated}
    }
}
```

Fig. 3: Sample TriQPAN pattern to be recorded in the event store.

Figure 3 depicts the pattern for `customerTripCreation`. The value of `DecisionTaskList` is recorded within the `Queries` part of the pattern since the agent has used it to take the decision. The `Criterion` part describes the conditions under which the `Actions` take place. It is specified in our example as follows. When a `DecisionTask` within the `DecisionTaskList` has the status 'commit' it will cause the action to create a `customerTrip` and set the status of the `DecisionTask` to 'committed'. The changes made during the execution will cause several notifications which can also be used for other explanations.

For an integration into an Jadex agent, the first step is to identify the decision points within the agent architecture.

The decision logics is complex and each decision comprises several steps that we call decision points. For instance a commit decision to serve a customer trip is part of a sequence of decisions subsumed under the goal EvaluateDecisonTask which processes every step in the live cycle of an customer request. This includes the decision whether a contract net protocol should be started or an request should be delegated as a result of the outcome of a contract net protocol. Not all of these micro decision are of interest with respect of the user demands. Thus, some of them are explicitly annotated as decision points.

Decision points are derived from state changes of beliefs, goals or plans.

When the decision points are identified you can place a logging event at every possible outcome of the decision point. Figure 4 shows a snippet of one of these outcomes in the source code of a vehicle agent. The decision point shown represents the case where the vehicle agent decides to commit a customer request and creates a customer trip. The eventTracker is used to log all the information about that event into a .json file.

This procedure is not limited to Jadex-based agents, but should be applicable to all, or at least many, agent development frameworks that implement reasoning agents, as long as the decision points are clearly identifiable.

```
case COMMIT: {
    //Decision point: customerTripCreation
    // creationof a new customer trip
    Trip newTrip = new Trip(currentDecisionTask,
        currentDecisionTask.getJobID(), "CustomerTrip",
        currentDecisionTask.getVATimeFromJob(),
        currentDecisionTask.getStartPositionFromJob(),
        currentDecisionTask.getEndPositionFromJob(), "NotStarted");
    trikeAgent.tripList.add(newTrip);
    //excecution of the logger
    eventTracker.addEvent(event, trikeAgent.tripList,"trike_events/Trike"
        + trikeAgent.agentID + ".json");
```

Fig. 4: A log functionality integrated into a decision point in a Jadex agent

5.2 Information extraction and validation

The validation algorithm can now search for relevant information inside the event store to validate the possible answer according to the specified test criterion.

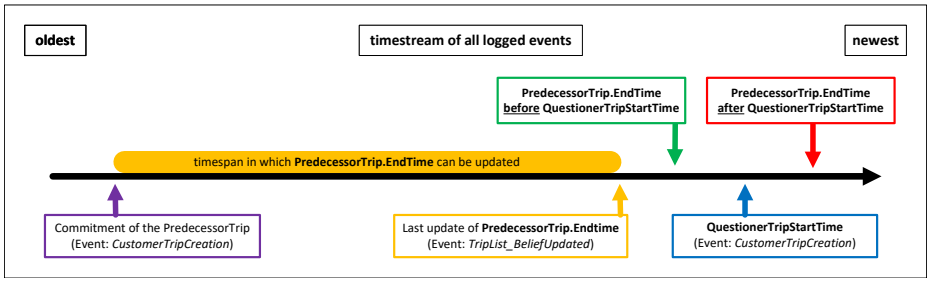

Fig. 5: important events for the validation visualized as a timestream.

We assume that the `TripID` that the customer's question refers to is known and that the entries inside the event store are sorted chronologically. A sample timestream of the events which can be a part of the validation is illustrated in Fig. 5. Below on the left side there is the `customerTripCreation` event of the `predecessorTrip`. This is followed by a timespan during which the `EndTime` of that trip can be updated with the final `TripList_BeliefUpdated` at its end. The second `customerTripCreation` event shows the commitment of the questioners trip. The two alternative time points (before or after) of the `EndTime` for the `predecessorTrip` are shown on the upper right corner.

The code depicted in Fig. 6 shows the validation algorithm. It searches the event store from the newest to the oldest entry. It will first search for the trip of the questioner, an event with the name `customerTripCreation` where the `TripID` equals the questionerTripID (line 3-9). With its time stamp (`EventTime`)

```
1  INPUT: QuestionerTripID
2  CauseOfDelay = false
3  # search for of the trip of the customer
4  FROM (EventTime: newest TO older){
5      IF ((name == CustomerTripCreation) &&
6          (TripID == QuestionerTripID)){
7          EventTimeOfQuestionerTripCreation = EventTime
8          BREAK
9      }
10 }
11 # search for the predecessor trip
12 FROM (EventTime: EventTimeOfQuestionerTripCreation TO older){
13     IF (name == customerTripCreation){
14         PredecessorTripID = TripID
15         EventTimeOfPredecessorTripCreation = EventTime
16         BREAK
17     }
18 }
19 # search for the most recent endtime of the predecessor trip
20 FROM (EventTime: newest TO PredecessorCreationTime){
21     IF ((name == TripList_BeliefUpdated) &&
22         (contains PredecessorTripID)){
23         # validation criterion
24         IF (PredecessorTrip.EndTime > QuestionerTripStartTime){
25             CauseOfDelay = true
26         }
27         BREAK
28     }
29 }
30 RETURN: CauseOfDelay
```

Fig. 6: Pseudocode with test criterion to validate if another `customerTrip` caused a delay.

we can narrow down the scope of the further search. From that `EventTime` we search for the `PredecessorTrip` (the next oldest `customerTripCreation`) to get its `TripID` and `EventTime` (line 10-17). For a validation of the potential answer we need the `EndTime` of the `PredecessorTrip`. The `EndTime` of a trip is stored inside the `TripList`, which is logged by `TripList_BeliefUpdated` events (Fig. 2). Should the `EndTime` of that trip have been changed during the runtime of the agent, every change will be stored within a `TripList_BeliefUpdated` event. To get the most recent one we look up the last `TripList_BeliefUpdated` event where the `PredecessorTrip` was mentioned (line 18-20). If the `EndTime` of the `PredecessorTrip` will be later than the `StartTime` of our `QuestionerTrip` we expect a delay and the `CauseOfDelay` is set to true (validation criterion, line 21-23).

Similarly to this algorithm, other possible causes for a delay can also be validated and a list of all positively validated causes can be created.

5.3 Answer verbalization

The short answer to the running sample discussed above would be: "There is a `customerTrip` before your trip." It hides details like the `EndTime` of the previous `customerTrip`, the commit time of the previous customer, belief updates and so on. It is still ongoing work to formulate different prompts for degrees of granularity that fit to the particular end user's demand for information. Promising solution ideas are to design a dialogue with the user to request further details and to integrate a dialogue management component into a RAG or plain LLM system. It is a further open issue how to deal with multiple causes that have been positively validated for the same question. A naive solution would be to let the LLM generate a couple of sentences for a conjunction of different causes.

5.4 Pattern reuse

Figure 7 gives an overview of some other questions of possible end-users of ATRIAS that can be answered by our model. The question "Why is this trike responsible for me?" refers to the concept of "responsibility", which is unknown to the agent. Responsibility can be inferred from various decisions made by the agent. For this special case there would be three possible answers. The agent is responsible for the trip because it "was delegated from the taxi control center and got a high utility score", it "was delegated from the taxi control center, got a low utility score, but was still committed after a CNP" and that it "was delegated by another trike after a CNP". In order to be able to validate the circumstances under which the trike committed the trip, the underlying reasoning processes have to be modeled by TriQPAN patterns. These will be "commitNewCustomerRequest", "CommitDespiteCNP", "CommitAsCNPparticipant".

The validation algorithm shown in Figure 6 was quite complex and had to be designed specifically for this single question. In contrast, many other possible answers can be validated by a simple approach that can be reused for different questions. For a validation of the above questions, it is sufficient to search for the listed patterns or logs within our event store. If such a pattern or log exists and relates to the `TripID` the possible answer can be validated.

Some other user questions just refer to a simple belief inside the agent and do not even require a reasoning process to refer to. "When will I reach my destination?", "What is your position at the moment?", "When will you arrive?" just require a mapping to the corresponding belief which stores that information.

As the number of possible end-user questions we consider increases, it becomes apparent that the events we need to log and the validation algorithms we need begin to repeat themselves (e.g. `TripList_BeliefUpdated`).

With a increasing amount of user questions, we expect this to become even clearer.

User Question	Possible Answer	Pattern & LOGs	Validation Algorithm and Test Criterion
Why is my trike late?	There is a `customerTrip` before your trip, that does not finish in time.	`customerTripCreation`, `DecisionTaskCommit`, `TripList_BeliefUpdated`	See Figure 6, (criterion in line 21-23)
	There is a charging trip before your trip, that does not finish in time.	`chargingTripCreation`, `estimateBatteryAfterTIP_-BeliefUpdated`, `TripList_BeliefUpdated`	Similarly to example shown in Figure 6
Why is this trike responsible for me?	The trip was delegated from the taxi control center and got a high utility score	`commitNewCustomerRequest`	Check IF (Pattern & LOGs) exist and contain ID
	The trip was delegated from the taxi control center, got a low utility score, but was still committed after a CNP	`TripList_BeliefUpdated`	Check IF (Pattern & LOGs) exist and contain ID
	The trip was delegated by another trike after a CNP	`CommitDespiteCNP`	Check IF (Pattern & LOGs) exist and contain ID
When will I reach my destination?		`TripList_BeliefUpdated`	Take the most recent (Pattern & LOGs) which contain ID
What is your position at the moment?		`AgentPosition-_BeliefUpdated`	Take most recent (Pattern & LOGs)
When will you arrive?		`TripList_BeliefUpdated`	Take most recent (Pattern & LOGs) which contain ID

Fig. 7: Overview of end-user questions and the requirements to our model to answer them.

6 FUTURE WORK AND CONCLUSION

In this paper we have presented a XAg model for End-user alignment. It was built upon an existing solution which was aimed at experts and expanded it to this new area of application. Further we carried out a feasibilty check by discussing the data structures and pseudo code for a running sample within our ATRIAS framework.

For the future, we are planning an empirical study with a potential user base to evaluate which questions users would really ask our agents. With a list of potential questions we can do a full implementation of our four-layered architecture to test the theory described in this paper in practice. This prototype then can be used to evaluate the strengths and limitations of our approach and think about further improvements. One possible refinement could be to allow the user to formulate questions freely, which are then mapped by an LLM or RAG system to a list of predefined questions that our system can answer.

Acknowledgements The authors would like to thank: Sebastian Rodriguez, who showed us examples of how TriQPAN patterns can be adapted to ATRIAS. Mariam Rahimi who conducted a preliminary survey on the interests of potential customers of our ATRIAS framework as part of her master's thesis.

References

1. Anjomshoae, S., Najjar, A., Calvaresi, D., Främling, K.: Explainable agents and robots: Results from a systematic literature review. In: 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019. pp. 1078–1088. International Foundation for Autonomous Agents and Multiagent Systems (2019)
2. Axhausen, K.W., ETH Zürich: The Multi-Agent Transport Simulation MATSim. Ubiquity Press (Aug 2016)
3. Bordini, R.H., El Fallah Seghrouchni, A., Hindriks, K., Logan, B., Ricci, A.: Agent programming in the cognitive era. Autonomous Agents and Multi-Agent Systems **34**, 1–31 (2020), publisher: Springer
4. Dennis, L.A., Oren, N.: Explaining BDI Agent Behaviour Through Dialogue. Auton. Agents Multi Agent Syst. **36**(1), 29 (2022), https://doi.org/10.1007/s10458-022-09556-8
5. Koeman, V.J., Dennis, L.A., Webster, M., Fisher, M., Hindriks, K.: The "Why Did You Do That?" Button: Answering Why-Questions for End Users of Robotic Systems. In: Dennis, L.A., Bordini, R.H., Lespérance, Y. (eds.) Engineering Multi-Agent Systems, vol. 12058, pp. 152–172. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-51417-4_8, http://link.springer.com/10.1007/978-3-030-51417-4_8, series Title: Lecture Notes in Computer Science
6. Langley, P., Meadows, B., Sridharan, M., Choi, D.: Explainable agency for intelligent autonomous systems. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 31, pp. 4762–4763 (2017), issue: 2

7. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T.: Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems **33**, 9459–9474 (2020)
8. Mauri, M., Erduran, Ö., Minor, M.: Jadex BDI Agents Integrated with MATSim for Autonomous Mobility on Demand. In: Engineering Multi-Agent Systems, 12th International Workshop, EMAS 2024, Auckland, New Zealand, May 6–7, 2024, Revised Selected Papers, vol. LNCS 15152, pp. 125–143. Springer (2024)
9. Mualla, Y., Kampik, T., Tchappi, I.H., Najjar, A., Galland, S., Nicolle, C.: Explainable Agents as Static Web Pages: UAV Simulation Example. In: Calvaresi, D., Najjar, A., Winikoff, M., Främling, K. (eds.) Explainable, Transparent Autonomous Agents and Multi-Agent Systems. pp. 149–154. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-51924-7
10. Persiani, M., Hellström, T.: The Mirror Agent Model: a Bayesian Architecture for Interpretable Agent Behavior. In: 4th International Workshop on EXplainable and TRAnsparent AI and Multi-Agent Systems (EXTRAAMAS 2022), Online via Auckland, NZ, May 9-10, 2022 (2022)
11. Pokahr, A., Braubach, L., Jander, K.: The Jadex Project: Programming Model. In: Multiagent Systems and Applications: Volume 1:Practice and Experience, pp. 21–53. Springer, Berlin, Heidelberg (2013)
12. Rahimi, M.: Explainable Agency for Users in Mobility on Demand. Master's thesis, Goethe Universität Frankfurt, Frankfurt am Main, Gemrany (March 2024)
13. Rao, A.S., Georgeff, M.P.: BDI agents: from theory to practice. In: ICMAS. vol. 95, pp. 312–319 (1995)
14. Ribera, M., Lapedriza, A.: Can we do better explanations? A proposal of user-centered explainable AI. In: IUI Workshops. vol. 2327, p. 38 (2019)
15. Rodriguez, S., Thangarajah, J., Davey, A.: Design Patterns for Explainable Agents (XAg). In: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems. pp. 1621–1629 (2024)
16. Shoham, Y.: Agent-oriented programming. Artificial intelligence **60**(1), 51–92 (1993), publisher: Elsevier
17. Singh, D., Padgham, L., Logan, B.: Integrating BDI Agents with Agent-Based Simulation Platforms. Autonomous Agents and Multi-Agent Systems **30**(6), 1050–1071 (Nov 2016). https://doi.org/10.1007/s10458-016-9332-x
18. Smith: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers **C-29**(12), 1104–1113 (1980)
19. Winikoff, M., Sidorenko, G.: Evaluating a mechanism for explaining BDI agent behaviour. In: Calvaresi, D., Najjar, A., Omicini, A., Aydogan, R., Carli, R., Ciatto, G., Mualla, Y., Främling, K. (eds.) Explainable and Transparent AI and Multi-Agent Systems. pp. 18–37. Springer Nature Switzerland, Cham (2023)
20. Winikoff, M., Sidorenko, G., Dignum, V., Dignum, F.: Why bad coffee? Explaining BDI agent behaviour with valuings. Artificial Intelligence **300**, 103554 (2021)
21. Yan, E., Burattini, S., Hübner, J.F., Ricci, A.: A multi-level explainability framework for engineering and understanding BDI agents. Autonomous Agents and Multi-Agent Systems **39**(1), 9 (Jan 2025). https://doi.org/10.1007/s10458-025-09689-6