Modeling and Reasoning about Explanation Requirements using Goal Models

Sotirios Liaskos[1] ✉, John Mylopoulos[2], Alex Borgida[3], and Shakil M. Khan[4]

[1] School of Information Technology, York University, liaskos@yorku.ca
[2] Department of Computer Science, University of Toronto, jm@cs.toronto.edu
[3] Department of Computer Science, Rutgers University, borgida@cs.rutgers.edu
[4] Department of Computer Science, University of Regina, shakil.khan@uregina.ca

Abstract. As modern software-intensive systems grow in complexity and autonomy there is a demand for them to incorporate services that explain their actions. Such explanation services allow stakeholders, especially end-users, to develop trust that the corresponding system complies with its objectives and limitations. Before such explanation services can be designed, their requirements need to be analyzed, in terms of both the kinds of stakeholder questions such services can entertain and the content of the answers they respond with. We propose a framework for goal-oriented modeling and analysis of explanation requirements. Explanation requirements are captured as stakeholder goals and are subsequently analyzed into specific explanation tasks and explanation interaction templates. By further extending the models with temporal and causal constraints, the resulting conceptual model can be used to generate answers to certain families of stakeholder questions. Formalization of the model allows both simulations of the explanation service and, under restrictive assumptions, a working prototype thereof.

Keywords: Goal Modeling · Software Systems Explainability · Golog

1 Introduction

We are interested in software systems that can explain their actions. Such systems include an autonomous vehicle (AV) that can explain its driving actions to its passengers, such as *"Why did you turn?" "Because I need to get gas"*, or a meeting scheduler (MS) that answers *"Why was I invited to this meeting?"* with *"Because of your expertise on the topics to be discussed"*. Explainability as a quality of software systems came to prominence with the advent of AI-intensive systems that play an important role in our daily lives and need to be transparent about their actions so that they can be justifiably [22] trusted by their stakeholders, and particularly their end-users [4,24]. The concern, however, extends to any socio-technical system – AI-powered or not – whose size and complexity makes the rationale of actions performed within its context difficult to comprehend without assistance.

Explanations are aimed at allowing an explainee to understand how and why an aspect of a system is as it is in a given context [4,20]. In one of their common forms, explanations are assignments of causal responsibility for a phenomenon [21] (as cited in [33]) [43]. In our case, the phenomenon is an action that a system has performed. For such phenomena, causes can be goals the system is trying to achieve. In the AV example, the overarching goal g_0 := *"Drive to Pittsburgh by 6pm"* is viewed as the cause of the corresponding system subgoals g_1 := *"Enter the highway"*, g_2 := *"Drive towards Pittsburgh"*, g_3 := *"Exit the highway"*, g_4 := *"If you are low on gas fill up"* and g_5 := *"Ensure safety"*, and as such constitutes an explanation for the pursuit of these subgoals. We call such causes *teleological* [33] and [9] (as cited by [19]). Alternatively, causes can be found in the design rules of the plan such as g_2 is enabled by g_1, so the answer to the question *"Why did you enter the highway?"* may be answered with *"Because that allows me to Drive towards Pittsburgh"*. We call these causes *canonical* as they are based on design rules.

For a system to be capable of providing such explanations, the requirements for the corresponding explanation functionality need to first be identified and modeled. Such functional requirements can be captured as *explanation goals* which express what domain *explainees* want to have explained. Explanation goals are, in turn, analyzed into explanation tasks. The latter describe the textual format and content of explanation interactions with explainees in order to fulfill the explanation goals. At the same time, when a model of the domain in question is present, it can serve as a basis for constructing certain additional explanations.

Based on these ideas, we present a goal-oriented framework for analyzing, modeling, simulating, and implementing explanation requirements. Functional and non-functional system requirements are captured using the i^* goal-modeling language [12,44], appropriately extended to capture temporal and causal relationships between goals and actions. Then, explanation requirements are captured in the form of explanation goals, assigned to various explainee roles. These explanation goals are operationalized into actions that capture run-time explanation information or render explanation interactions. The textual format of the latter is described through templates. In addition, a set of axioms are introduced that utilize the information of the extended model to construct responses to ad hoc explainee questions. Subsequent formalization of said templates and axioms allows design-time simulation of the envisioned explanation interactions, supporting its early assessment and improvement. Further, when the envisioned system is implemented in accordance to the goal model and its actions are logged, our formalization can serve as the basis of the run-time explanation engine.

The main contribution of our work is a model-driven approach for systematically discovering and specifying explanation requirements within an established goal-oriented requirements framework, while at the same time, thanks to the proposed formalization and axiomatization, allowing simulation and prototyping of the specified explainability service.

The rest of the paper is organized as follows. In Section 2 we describe the extensions to i^* that are needed for capturing explanations. In Sections 3 and

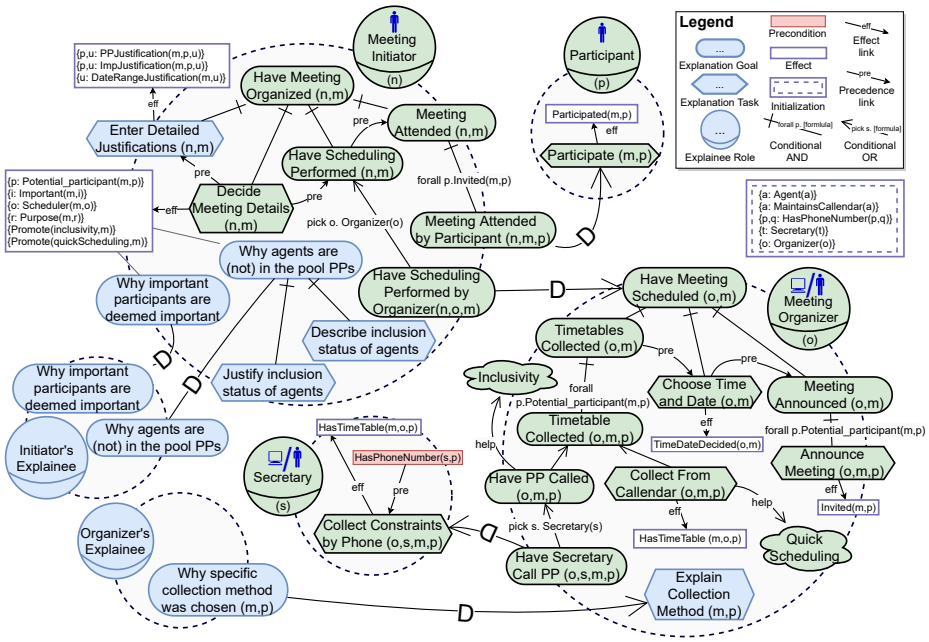

Fig. 1. Augmented Goal Model (for meaning of baseline iStar 2.0 elements see [12])

4 we describe explanation templates and explanation axioms for custom and generic explanations, respectively. In Section 5 we sketch how the above are formalized to allow for design-time and run-time reasoning, described in Section 6. We present related work in Section 7 and conclude in Section 8.

2 Capturing requirements in i^+

The i^* requirements modeling language [44] captures requirements in terms of *goals* that actors (stakeholders) need to fulfill and the actions (aka *tasks* in i^*) through which their goals can be fulfilled. Given an i^* model of stakeholder requirements, we are interested in extracting explanations that stem from *teleological causes* of actions, that is, what goal the action serves, as well as *canonical causes* of actions, that is, what rules or constraints lead the actor to perform a specific action. Teleological causes are readily captured in i^* through modeling tasks as the result of recursive goal refinement. However, to capture canonical causes, we need to extend i^* with the appropriate constructs for representing temporal and causal constraints. We adopt and extend a recent derivative of i^*'s successor *iStar 2.0* [12] for representing action theoretic aspects of goal models [27]. We call the resulting extension i^+. An example of an i^+ model appears in Figure 1. Figure 2 shows a meta-model with the concepts introduced in i^+ as they relate to the iStar 2.0 standard.

The standard iStar 2.0 elements, such as roles, goals, tasks, AND- and OR-refinements and dependencies [12] are augmented with elements that represent

constraints to the order by which tasks can be performed. At the core of the additional elements are *domain predicates*, such as *Agent* or *HasPurpose*, which, applied to *domain objects*, such as *John* or *GrantAdjudication*, are used to build *domain literals* to represent statements about the world, e.g., *Agent(John)* or *HasPurpose(Meeting, GrantAdjudication)*. Each *effect* (unshaded rectangles) contains lists, or descriptions (more below) of lists of domain literals. Connecting a task with an effect through an *effect link* shows that performance of the task has the effect of turning the literals listed or described in the effect box true. Further, *preconditions* contain closed formulae and are connected with tasks through *precedence* links, which, in turn, signify that the origin of the link must be satisfied before the destination of the link can be attempted.

Goals and tasks contain parameters that can be bound to domain objects. By convention, the first parameter is a reference to the agent that pursues the goal or performs the task. Thus, *Have Meeting Scheduled(o,m)* signifies that the goal *Have Meeting Scheduled* is pursued by an agent *o* for a distinct meeting *m*. This way i^+ allows us to reason with multiple *instances* of goals and tasks, each uniquely identified by its parameter instantiations. When a goal is (unconditionally) *refined* into a subgoal or *operationalized* into a task, the parameters of the subgoal/subtask must be a subset of the parameters of the parent goal. An exception is *conditional AND* and *conditional OR* refinements, which allow extension of an AND- or OR-refinement respectively to all objects that fulfill a stated condition (a formula of domain literals), and may result in an additional parameter for the subgoal or subtask. In Figure 1, *Timetables Collected (o,m)* is refined into as many instances of the goal *TimeTable Collected (o,m,p)* as the known potential participants *p* for meeting *m*. We signify this through the formula *forall p. Potential_participant(m,p)* on the refinement link. Likewise, goal *Have PP Called (o,m,p)* (*PP* is, henceforth, shorthand for *Potential Participant*) is OR-refined into an instance of task *Ask Secretary to Call PP (o,s,m,p)*, whose performance for some *s* that is a secretary suffices for the fulfillment of the goal. This is indicated by formula *pick s. Secretary(s)* on the refinement link.

The parameters of the literals listed in an effect must be a subset of the parameters found in the task that is connected to the effect. Often, however, tasks such as *Decide Meeting Details (n,m)* generate new instance-level information and we wish to represent this generation process. We use *assertion descriptions* for the purpose. They come in the form {*params: Predicate(params)*} which is a shorthand for the list of literals *Predicate(*$Param_1$*), Predicate(*$Param_2$*), ...*, where $Param_i$ are objects of the domain. Thus, the description {*p: Potential_participant(m,p)*} found in the effect of task *Decide Meeting Details (n,m)*, can be instantiated to list *Potential_participant (m, Alice)*, *Potential_participant (m, John)*. The description denotes that, as a result of performing the task, the meeting initiator will come up with instances of potential participants (Alice, John, etc.), that are unknown at the time of the analysis.

Further, we define *(task) histories tH* to be sequences of leaf level tasks, which have all their parameters bound to domain objects, and which can be executed in the given sequence in compliance with the precedence constraints of

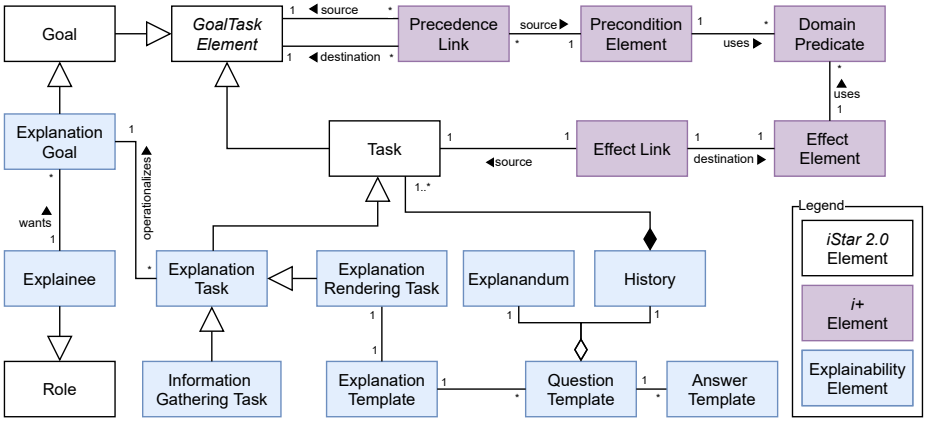

Fig. 2. Meta-model of *i+* and explainability extensions

the model. A history is, further, *goal satisfying* with respect to a goal g, if after all tasks are executed, g is satisfied according to the AND/OR decomposition structure. Finally, *initialization elements* containing assertion descriptions are added to the model to describe what domain literals need to be asserted as true or false prior to calculating task histories.

3 Modeling Explanation Requirements

The i^+ extension is used for developing models of stakeholder goals pertinent to a requirements problem at hand. When we are interested to also model stakeholder requirements for acquiring explanations on why actor actions are chosen and performed, i.e., *explanation requirements*, the models need to be augmented accordingly. We propose to approach explanation requirements as functional ones, stemming from explanation needs of *explainees* in the domain. Explainees are actors that inquire about the performance of tasks by other actors. We model explainees as *i* roles*. Depending on the application, explainee roles may reflect specific organizational roles and positions, e.g., mandated regulatory, quality control, or customer advocacy officers, or they may be generic roles that any actor in the domain may play. Explainee roles may also be dedicated to explaining the actions of one particular other actor. In the example of Figure 1, two explainee roles are included, one concerned with the actions of the initiator (Initiator's Explainee) and one concerned with the actions of the organizer (Organizer's Explainee). Thus, a participant may play the initiator's explainee role when they have the goal to understand why, e.g., their own names have been included or excluded from a participant list.

Explainees have *explanation goals*. These goals reflect states of the world in which a desired explanation about the actions of an actor has been acquired. For example, the initiator's explainee has the explanation goal to *"[Have] why important participants are deemed important [explained]."* – we omit the first

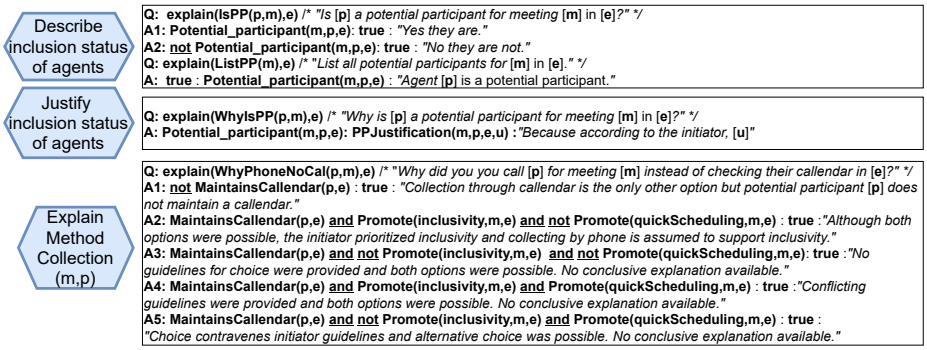

Fig. 3. Explanation Rendering Tasks and the associated Explanation Templates.

and last word when writing such goals. Top level explanation goals like this are delegated or refined like any other *i** goal. In our example, the explanation goal *"Why agents are (not) in the pool of potential participants"*, is delegated to the initiator, as she is the one who makes the corresponding decisions. The delegated goal is further analyzed into specific tasks that are required for satisfying the corresponding explanation goal. This includes verifying the inclusion status of various agents in the list of potential participants and offering an explanation as to why the status is as such. We call such tasks that gather or render explanation information *explanation tasks*. In our example, to be able to perform the explanation task that explains the potential participant status of each agent, explanations must be registered by the meeting initiator at an earlier time. Thus, the *information gathering task Enter Detailed Justifications (n,m)* with the corresponding effect is added as one of the initiator's tasks. According to the effect, at run time, the task results in a number of literals of the form *PPJustification(m,p,u)* to be generated, each assigning a message u as justification for inclusion of an agent p in the PP list for meeting m. This informs the designers that a facility for adding message u for each p when deciding meeting details for m must be available to the initiator. Thus, the analysis of explanation goals leads to the discovery of additional requirements for functions that are necessary for capturing explanation information. Such information capture is the first of two aspects of operationalizing explanation goals. The second aspect is the rendering of explanations which is described using explanation templates.

3.1 Explanation Templates

Explanations are offered in response to appropriately formulated explanation questions posed by the explainees. Part of the requirements specification process is to precisely describe the required format of both the question and the answer to the question, as well as the content of the answer based on information available in the system, including information acquired through other tasks in the model. Each *explanation rendering task* is, hence, associated with an *explanation*

template to describe the format of question and answer that the task entails as well as how the content of the answer can be retrieved or constructed.

Three explanation templates can be seen in Figure 3, each corresponding to one of the explanation tasks for Figure 1. An explanation template consists of a set of *question templates* and, for each such, one or more alternative *answer templates*. Question templates are of the form:

explain(explanandum(q_1,q_2,...),e)

where *explanandum(q_1,q_2,...)* is a goal or task whose parameters are to be bound at the time that the question is posed, and e is a task history offering the context within which the explanation is required. For example, consider the scenario in which an explainee studies the task history that unfolded for the scheduling of a meeting in a university setting. A question they may ask is of the form:

explain(WhyIsPP(p,m),...,e)

which, in plain language, asks why agent object m is treated as a potential participant for meeting m in history e. At the time of posing the question, examples of specific objects are used, such as p = *"Anita"* and m = *"the promotion adjudication meeting"*. The answer templates are of the form:

guard/(q_1,q_2,...,e) : query/(q_1,q_2,...,a_1,a_2,...,e) : render/(a_1,a_2,...)

The *guard* is a logical expression grounded on domain literals that must be true for the answer to be provided. Variables q_1,q_2,... are a subset of those contained in the question. The *query* part consists of a logical expression that is grounded on domain literals and that contains parameters q_1,q_2,...,e that are, again a subset of those of the question, as well as free parameters a_1,a_2,.... The *render* part is a function that presents the output parameters a_1,a_2,... in a human readable format through, e.g., instantiating parameterizable text templates as seen below. For example, the following can be an answer template to the above question:

Potential_participant(m,p,e) : PPJustification(m,p,u,e) : "Because, according to the initiator, [u]."/(u)

At run time the answer is instantiated as follows. First the guard condition is instantiated using the parameters of the question. In the example above, the guard condition is *Potential_participant("the promotion adjudication meeting", "Anita", e)*. If the guard is true, an answer can be rendered. Firstly, a value for u will be sought such that the following holds:

PPJustification("the promotion adjudication meeting","Anita",u,e)

Recall that, as a result of analysing explanation goals, the model contains task *Decide Meeting Details (n,m)* in which initiator n creates such literals. In our case, u = *"Anita has been elected as recording secretary"* may have been entered by the initiator and, as such, satisfy the query expression. The final response instance is simply a human-friendly rendering of the query results. If multiple u's satisfy the query, multiple answers will be provided. Notice also that a specific task history e is present in all relevant literals.

In Figure 3, additional examples of explanation templates are provided. The one addressing explanation task *Explain Method Collection (m,p)* tackles the question why a certain child of an OR-decomposition is chosen. The explain-

ability analyst designs various alternatives based on different guard conditions. The first answer observes that the alternative is not feasible. The second answer suggests that the choice is compliant to quality guidelines offered by the initiator and does not contradict any other guidelines. The remaining choices contextualize the choice but do not offer a definitive explanation. In the particular case, the analyst does not fully determine how variability is bound at the cost of being able to only provide plausible and not definite explanations.

4 Ad hoc Explanations and Explanation Axioms

Explanation goals, tasks, and templates help analysts to identify and represent specific explanation requirements and designs thereof (templates) that can be known at the requirements stage. This analysis ensures that the requirements specification includes requirements for explanation information tracking and answer-rendering functions that are specific to the domain in question.

However, there may be explanation requirements that evade capture during analysis. In that case, we may still be able to exploit information embedded in the i^+ model – in the form of intentional (refinements, operationalizations, contributions) and canonical (precedences, effects) relationships – to provide *ad hoc explanations*, that is, explanations not captured during requirements analysis. Given an observed action to be explained, such ad hoc explanations can render (a) the motivation for the action's performance in terms of actor goals they may support, (b) the action's potential role in enabling subsequent necessary steps, (c) the possible absence of feasible alternatives for the action, or (d) compliance of the action with certain preferences over quality goals.

For generating such explanations, we utilize a set of general *explanation axioms* that describe how such relationships can be combined to explain performance of tasks. Axioms are of the form:

$$explains(y, x, e) \longleftarrow f(y, x, e)$$

where y is an explanation, x is an explanandum (what needs to be explained), e the history relative to which the explanation is sought, and $f(y, x, e)$ is a formula based on structural characteristics of the i^+ model and, where applicable, the content of e. The explanandum x is a task that appears in e and the explainee desires to have explained, or a goal that has been rendered as an explanation to an earlier query. Likewise, explanations y are other goals or tasks. In a typical application of the axioms, the explainee will start by asking what explains a task within a history, which will return various other tasks or goals, which are, in turn, the explananda of follow-up queries. Below we present the axioms in detail and in later sections we show how they can be used.

Operationalizations. The first axiom is based on the observation that in an operationalization relationship, i.e., the (AND- or OR-) refinement of a goal into a task, the goal being operationalized constitutes a possible explanation as to why the task was performed. Hence, in Figure 1 task *Collect From Calendar (o,m,p)* is one of the tasks that operationalize goal *Timetable Collected (o,m,p)*. Clearly, question *"Why did organizer [o] collect [p]'s constraints from their cal-*

endar for meeting [m]?" can be answered with *"Because [o] wanted to collect [p]'s timetable for [m]"*. Hence our first axiom can be as follows, where x is a task, y is a goal, and e is a task history:

$explains(y, x, e) \longleftarrow occurredIn(x, e) \wedge operationalizedBy(y, x)$ (Axiom 1)

where $occuredIn(x, e)$ holds if task x is included in e.

Refinements. The above can be extended to refinements between goals. In the example, g_3 := *"Timetable Collected (o,m,p)"* (the explanation of the previous question, which now becomes the new explanandum) is a subgoal of g_2 := *"Timetables Collected (o,m)"*, which is, in turn, a subgoal of g_1 := *"Have Meeting Scheduled(o,m)"*. Each g_i can be explained by its parent g_{i-1}. The axiom is then as follows, where both x and y are goals and $satisfiedIn(x, e)$ holds iff the task occurrences included in e satisfy x according to AND/OR structure.

$explains(y, x, e) \longleftarrow satisfiedIn(x, e) \wedge refinedBy(y, x)$ (Axiom 2)

Dependencies. The initiator for a meeting is the original actor who wants to schedule it. However, the scheduling is delegated to the meeting organizer. Accordingly, a question about meeting organizer o, *"Why did [o] want to have meeting [m] scheduled?"* can be reasonably answered with *"Because the initiator [n] asked [o] to"*. In reference to Figure 1, *Have Scheduling Performed by Organizer(n,o,m)* explains why *Have Meeting Scheduled (o,m)*, where n is the initiator, o is the chosen organizer, and m is the meeting.

Let $dependency(r, d, y, x)$ denote a dependency between actors, whereby a depender actor r depends on a dependee actor d to fulfill depender goal y which becomes dependee goal or task x. Then:

$explains(y, x, e) \longleftarrow \exists(r, d).dependency(r, d, y, x) \wedge delegated(r, d, y, x, e)$ (Axiom 3)

where $delegated(r, d, y, x, e)$ holds if, in history e, actor r delegated goal y to actor d, which goal became goal or task x for d.

Enablement. Consider the question *"Why did [o] collect timetables for meeting [m]?"*, now answered with *"Because [o] needed to choose a time and a date for meeting [m]"*. The specific answer constitutes a canonical explanation rather than a teleological one. It explains performance of a task on the basis of it being necessary for allowing some other task to be executed later in the history of tasks. In the example, this is established by the precedence link that prevents performance of task *"Choose Time and Date (o,m)"* unless *"Timetables Collected (o,m)"* is first fulfilled. We formulate the above with the axiom:

$explains(y, x, e) \longleftarrow enables(x, y) \wedge includedIn(y, e) \wedge includedIn(x, e)$ (Axiom 4)

In the above, y and x are goals or tasks and $includedIn\ (z, e) \longleftrightarrow occuredIn\ (z, e) \vee satisfiedIn(z, e)$. Furthermore, $enables(x, y)$ means that there is an explicit or implicit precedence constraint between x and y, such that prior performance, achievement, or satisfaction of x is necessary (but not necessarily sufficient) for performance (task) or satisfaction (goal) of y to be possible.

An explicit constraint exists when there is a direct precedence link from x to y. Implicit constraints emerge when precedence links originate from high-level goals and are, hence, inherited by their successors in the refinement hierarchy.

Specifically, if goal g_1 targets goal g_2 with a precedence link, every task that is a descendant of g_1 can be seen as an enabler of g_2. This is formalized as follows:

$$enables(x, y) \longleftarrow pre(x, y) \quad \text{(Axiom 4.1)}$$

$$enables(x, y) \longleftarrow ancestorOf(w, x) \wedge enables(w, y) \quad \text{(Axiom 4.2)}$$

In the above, $ancestorOf(w, x)$ denotes that there is a chain of refinements and/or operationalizations from high-level goal w down to goal or task x. In Figure 1, for example, a precedence link is drawn from goal *"Timetables Collected (o,m)"* to task *"Choose Time and Date (o,m)"*. According to the axiom, satisfaction or performance of any descendant of *"Timetables Collected (o,m)"* contributes to the enablement of *"Choose Time and Date (o,m)"*. As such, goal *"Have Secretary Call PP (o,s,m,p)"* enables *"Choose Time and Date (o,m)"*. Intuitively, Axiom 4.2 represents that when an element (goal, task, precondition) enables a high-level goal through a direct precedence link, every task under that direct enabler element is also an indirect enabler. We note, however, that multiple such indirect enablement relations may target the same goal. Hence, satisfaction of one of the origins is a necessary but not sufficient condition for the actual enablement of the target.

OR-refinements. A final set of axioms offers explanations on why a child of an OR-refined goal, henceforth: OR-sibling, is chosen over some other child, as evident in a history e. In goal analysis, identifying optimal choices for OR-refinements is typically done by a global optimization process whose details are not part of the model (e.g., [27,28]). Even if the optimization algorithm is known, explaining local decisions within the context of a complex global optimization task can be hard. To make an analogy, asking why a specific OR-sibling is chosen over another is similar to asking in the knapsack problem context why an object of a specific weight and size is chosen for inclusion in a knapsack. Nevertheless, in our case, *plausible* explanations for OR-sibling choices made by the external optimizer can be possible. Specifically, a plausible explanation can be produced when (a) at the time of the choice of an OR-sibling no other sibling was possible, or when (b) in the presence of more than one possible alternatives, the alternative chosen is the only one that contributes to qualities for which express statement(s) of preference exist. To produce the corresponding axioms we first define the following auxilary predicates:

$$onlyFeasibleIn(x, e) \leftrightarrow \forall y_i.[\text{OR\_}sibling(y_i, x) \rightarrow \neg poss(y_i, e)] \quad \text{(Aux. 1)}$$

$$preferred(x) \leftrightarrow \exists q1.[contr(x, q_1, plus) \wedge promote(q_1)] \wedge \neg\exists q_2.[contr(x, q_2, minus) \wedge promote(q_2)] \quad \text{(Aux. 2)}$$

$$onlyPreferredIn(x, e) \leftrightarrow \forall y_i.[\text{OR\_}sibling(y_i, x) \rightarrow \neg(prefered(y_i) \wedge poss(y_i, e))] \quad \text{(Aux. 3)}$$

The first predicate, $onlyFeasibleIn(x, e)$, holds if none of x's OR-siblings y_i are possible ($poss(y_i, e)$) at the end of e. Predicate $preferred(x)$ holds if x sends a positive contribution to some quality q_1 ($contr(x, q_1)$) and that quality has been declared to be a preferred quality ($promote(q_1)$), while x does not contribute negatively to some other preferred quality. Predicate $onlyPreferredIn(x, e)$ holds when there is no alternative y_i to x that is both feasible in history e and preferred. We note that $\text{OR\_}sibling(y, x)$ holds both for direct OR-siblings and for

cases in which, e.g., y is an OR-sibling of an ancestor of x, which also makes y and x alternatives. For example, *Collect Constraints by Phone (o,s,m,p)* is an alternative to *Collect From Calendar (o,m,p)*. We omit here the exact definition for brevity. The explanation axioms are then as follows:

$explains(x, e) \longleftarrow onlyFeasibleIn(x, e_{sel})$ (Axiom 5)

$explains(x, e) \longleftarrow onlyPreferredIn(x, e_{sel})$ (Axiom 6)

where e_{sel} is a prefix $t_1, t_2, \ldots, t_n$ of e such that t_{n+1} is either x or, if x is a goal, a leaf-level successor of x but no t_i is any of the two for $i \leq n$. Thus, e_{sel} marks the moment in e where the choice to perform/satisfy x is made.

According to Axiom 5 selection of an OR-sibling is explained if none of its other siblings was possible at the time e_{sel} when goal or task x was selected. Axiom 6, on the other hand, explains OR-child selection on the basis that none of the competing alternatives is possible and preferred with respect some quality.

5 Formalizing Explanations

So far we have introduced an extension to the standard i^+ notation, explanation templates, as well as axioms for ad-hoc explanations. We now sketch a semantics of these extensions and constructs that pave the way for automatically calculating run-time explanations in a simulated or real environment. The semantics is based on Golog [26], a Situation Calculus [25] based language for specifying dynamic domains and agents. We specifically present a set of rules for translating i^+ models, templates, and axioms into Golog as well as auxiliary logic programs, that allow both generation of simulated histories and query answering over such histories.

5.1 Situation Calculus and Golog

Golog's main concepts are *fluents*, *actions* and *situations*. Fluents describe what is true in the domain at a given situation. They are represented using n-ary predicates such as `timeAndDateChosen(matilda,hiringCmt,s)`, with a situation term `s` as their last argument. A situation is a first-order term that represents a sequence of actions from an initial situation $\mathtt{S}_0$ where no action has been performed. The reserved function symbol `do(a,s)` denotes the situation which results from performing action `a` in situation `s`. Situations emerge from nested application of the function, as in $\mathtt{do(a}_n\mathtt{,do(a}_{n-1}\mathtt{,\ldots,\ do(a}_1\mathtt{,S}_0\mathtt{)\ldots)))}$. Finally, a special predicate `poss(a,s)` is used to state that it is possible to perform action `a` in situation `s`.

To describe a domain a set of axioms are defined over the above constructs. The most important are *action precondition axioms* and *successor state axioms*. The former are defined for each action `a` and describe the conditions under which actions can be performed. For example to represent that action `chooseTimeAndDate(o,m)` is possible in `s` only if fluent `sat_timeTablesCollected (o,m,s)` is possible in that situation we write:

`poss(chooseTimeAndDate(o,m),s)` $\leftrightarrow$ `sat_timeTablesCollected(o,m,s)`

Successor state axioms are defined for each fluent and describe how the fluent's truth value changes due to the performance of tasks. For example, the following axiom says that fluent `perf_chooseTimeAndDate(o,m)` is true in situation `do(a,s)` if it was true in the previous situation or action `a` made it true.

```
perf_chooseTimeAndDate(o,m,do(a,s)) ↔ perf_chooseTimeAndDate(o,m,s)
                                      ∨ (a = chooseTimeAndDate(o,m))
```

Finally, Golog defines constructs for programming agent behavior that is consistent with the action theory defined by the axioms. Programming constructs of interest are *sequences* and *nondeterministic choices* of actions which are both used in the context of *procedures*, written as `proc([name],[body])`. The following two procedures contain in their bodies a sequence of three and a nondeterministic choice between two other procedures and/or actions, respectively:

```
proc(haveMeetingScheduled(o,m), timeTablesCollected(o,m) :
                        chooseTimeAndDate(o,m) : meetingAnnounced(o,m))
proc(timeTableCollected(o,m,p), havePPCalled(o,m,p) #
                                    collectFromCallendar(o,m,p))
```

Procedure bodies can define loops using `while([condition],[body])` as in the following example in which procedure `timeTableCollected(p)` is invoked for all `p` that fulfill the condition of the first argument – in our case `some([var], [expression])` which is true if there is an instance of variable `[var]` for which `[expression]` is true:

```
proc(timeTablesCollected(o,m,p),
     while( (some(p,potential_participant(p),¬ timeTableCollected(p))),
            pick(p,?¬timeTableCollected(p),collectTimeTables(p)))
```

The expression in the body of the `while` is of the form `pick([var], ?[expression], [procedure/action])`, which invokes `[procedure/action]` for a non-deterministic choice of `[var]` that satisfies `[expression]`.

5.2 From i$^+$ to Golog

The semantics of i^+ is based on a set of translation rules of i^+ constructs into Golog constructs. We present the most important of them below using illustrative examples based on Figure 1:

Actors, Goals, Tasks, and Effects. Every actor in the goal model is mapped to an objects in the Golog domain – e.g. `meetingOrganizer` or `potential_participant`. Every goal such as *Timetables Collected (o,m)* is translated into (a) a satisfaction fluent, `sat_timeTablesCollected(o,m,s)`, and (b) a procedure name `timeTablesCollected(o,m)`. Every task in the goal models, such as *Choose Time and Date (o,m)* is translated into (a) a performance fluent (`perf_chooseTimeAndDate(o,m,s)`), (b) an atomic action (`chooseTimeAndDate(o,m)`). Every effect is translated into as many fluents (*effect fluents*) as the predicates that are listed in the effect. For example, the effect containing *hasTimeTable(m,o,p)* is translated to fluent `hasTimeTable(m,o,p,s)`.

Goal refinements and delegations. In all cases, each goal in the goal model is translated into a Golog procedure with the corresponding name. If the goal is OR-refined, the procedure's body contains a non-deterministic choice of the procedures (resp. actions) corresponding to each sub-goal (resp. task):

```
proc(timeTableCollected(o,m,p),  havePPCalled(o,m,p)
                                 # collectFromCallendar(o,m,p))
```

Moreover, an axiom for the satisfaction fluent of the goal is introduced based on the disjunction of the satisfaction fluents of the children like so:

```
sat_timeTableCollected(o,m,p,s) ← sat_havePPCalled(o,m,p,s)
                                  ∨ sat_collectFromCallendar(o,m,p,s)
```

If the goal is AND-refined, the procedure's body contains a non-deterministic choice of all permutations of sequences of procedures/actions corresponding to the sub-goals/sub-tasks. For example:

```
proc(haveMeetingScheduled(o,m),
     (timeTablesCollected(o,m) : chooseTimeAndDate(o,m) : meetingAnnounced(o,m))
     # (timeTablesCollected(o,m) : meetingAnnounced(o,m) : chooseTimeAndDate(o,m))
     # ... [four more permutations] )
```

For AND-refinments the satisfaction fluent is a simple conjunction of the satisfaction fluents corresponding to the children:

```
sat_haveMeetingScheduled(o,m,s) ←
        (sat_timeTablesCollected(o,m,s) ∧ sat_chooseTimeAndDate(o,m,s) ∧
         sat_meetingAnnounced(o,m,s))
```

Conditional AND- and OR-decompositions are translated into a procedure whose body contains a `while` or a `pick`, respectively. For example, for conditional AND-decompositions (syntactically simplified here):

```
proc(meetingAnnounced(o,m)
     while(some(p,potential_participant(p) ∧ ¬ invited(o,m,p)),
           pick (p,?(potential_participant(p) ∧ ¬ invited(o,m,p)),
                 announceMeeting(o,m,p)))
```

Thus, while there are objects `p` identified as potential participants for which the meeting has not been announced, (i.e., they have not been invited), the procedure picks a `p` that satisfies this property and performs `announceMeeting(o,m,p))`. For conditional OR-decompositions are of a similar format where `while` is replaced by `pick`.

```
proc(havePPCalled(o,p),
     pick(t, ?(secretary(t)) : haveSecretaryCallPP(o,t,p)))
```

Satisfaction fluents of conditional refinements are constructed by checking that every (resp., that there exists one) object fulfilling the condition (resp., for which) the child goal is satisfied. For an AND-refinement, for example:

```
sat_meetingAnnouncedToall(o,m,s) ← ¬ (∃ p.(invited(p,s) ∧
                                           ¬ perf_announceMeeting(m,p,s)))
```

A conditional OR-refinement is as above with the negations removed:

```
sat_meetingAnnouncedToSome(o,m,s) ← (∃ p.(invited(p,s) ∧
                                        perf_announceMeeting(m,p,s)))
```

If the goal is delegated, i.e., is the depender goal of a dependency, the procedure is of the form:

```
proc(depender_goal(delegator,...),
     pick (p, ?dependeeActorType(p),
              (delegate(delegator,p,depender_goal,dependee_goal) :
              dependee_goal(p,[delegator],...))))
```

The procedure picks an actor object that satisfies the dependee type, it performs a reserved `delegate` action and invokes the procedure action corresponding to the deependee element.

Effect and Performance Fluents. For each effect and performance fluent introduced above, a successor state axiom is introduced of the form:

$$\texttt{eff\_fluent}_i\texttt{(do(a,s))} \leftarrow \texttt{eff\_fluent}_i\texttt{(s)} \vee \texttt{(a = action}_1\texttt{)} \vee \texttt{(a = action}_2\texttt{)} \vee \ldots$$

Where $\texttt{action}_i$ represent the tasks for which there exists an effect link that points to an effect that includes $\texttt{eff\_fluent}_i\texttt{(s)}$. In the case of performance fluents, $\texttt{action}_i$ represents the associated performance task (i.e., `perf_[task name](...,s)`).

Precedence Links. Firstly, we replace each precedence link that targets a goal with a set of precedence links with the same origin and target every leaf task that is a descendant of the original target. Then, for each action (which, recall, represents a task at the i^+ level) we develop a precondition formula by (a) collecting all precedence links that target the task, (b) forming the conjunction of the satisfaction/performance fluents representing the origin elements. The resulting formula is used to construct the action precondition axiom. In the Figure 1 example, assuming there was an additional precedence from *Timetables Collected (o,m)* to *Meeting Announced (o,m)*:

```
poss(meetingAnnounced(o,m),s) ← sat_chooseTimeAndDate(o,m,s) ∧
                                sat_timeTablesCollected(o,m,s)
```

Situations as Traces. Finally, we observe that a situation $\texttt{do(a}_n\texttt{,do(a}_{n-1}\texttt{,} \ldots \texttt{,do(a}_1\texttt{,S}_0\texttt{)...)))}$ in Golog terms corresponds to a history $[t_1,t_2, ...,t_n]$ in the i^+ ontology, where each task t_i corresponds to Golog action a_i, and the initial situation $\texttt{S}_0$ corresponds to a state in which no task has been performed.

5.3 Translating Explanation Templates

We now turn our focus on the semantics of explanation templates with respect to the Golog specification. Recall that a custom explanation template is of the form: *guard/*$(q_1,q_2,...,e)$ *: query/*$(q_1,...,e,a_1,...)$ *: render/*$(a_1,a_2,...)$. As we saw, instances of explanation templates are relativised to histories e, which, in turn, map to situations `s`. Accordingly, the guard condition corresponds to a boolean expression constructed with fluents relativized to the situation `s` corresponding to history e. The parameters in the expression are all bound by the user to specific objects. The query is then another such expression with free parameters

to be bound to values that make the expression true in `s`. For human readability, these parameters are then used to construct the answer through a reserved `render` predicate which concatenates constant text with parameter instantiations to construct the answer in string `a`. For example, the second template of Figure 3 is translated to formula:

```
explain(whyIsPP(p,m,a),s) ← potential_participant(p,m,s) /* guard */
                          ∧ ppJustification(p,m,u,s)  /* query */
                          ∧ render("Because...",u,a)  /* rendering */
```

In the above, explanation is requested for inclusion of participant `p` to meeting `m` in a situation `s`. If, indeed, `p` is a potential participant (guard fluent holds), a value for `u` is sought such that `ppJustification(p,m,u,s)` holds as well. The render function simply delivers `u` in a human readable format.

5.4 Translating Generic Explanations

We finally sketch how the Axioms 1-6 are formalized. Axioms 1-4 are straightforward rules which depend on predicates reflecting structural characteristics of the i^+ model plus predicates that verify the inclusion of the explanatum and/or explanation are part of the history, i.e., *occured(x,e)* and *attempted(x,e)*. These are translated into Golog's reserved `holds(x,s)`, which holds if `x` is true in `s`, according to initial situation and successor state axioms. In our case `s` corresponds to history e and `x` is the satisfaction (for goals) or performance (for tasks) fluent.

Translation of Axioms 5 and 6 depends on the situation for which the explanation question is posed. Auxiliary predicates *onlyFeasible*(x, e), and *onlyPreferred* (x, e) are translated into the corresponding fluents `onlyFeasible(x,s)`, and `onlyPreferred(x,s)` whose definition follows directly from Axioms *Aux. 1* and *Aux. 3*, where `s` is, again, the Golog situation corresponding to history e.

With these defined, translating Axioms 5 and 6 reduces to constructing a predicate `getPrefix(x,s,`$\mathtt{s}_{sel}$`)` for extracting situation $\mathtt{s}_{sel}$ that corresponds to history prefix e_{sel}. The predicate holds (a) if `x` holds in $\mathtt{S}_0$, in which case $\mathtt{s} = \mathtt{s}_{sel} = \mathtt{S}_0$ or (b) $\mathtt{s}_{sel}$ is a prefix of `s` such that if `do(a,`$\mathtt{s}_{sel}$`)` is also a prefix for `s`, then `x` does not hold in $\mathtt{s}_{sel}$ and `x` holds in `do(a,`$\mathtt{s}_{sel}$`)` – i,e., performance of `a` turns `x` from false to true. A simple logic program (omitted here for simplicity), replays `s` from initial situation $\mathtt{S}_0$ until the first `a` is found that brings about this transition to the truth value of `x` and the corresponding prefix $\mathtt{s}_{sel}$ is returned. Hence Axioms 5 and 6 are translated into the following rules:

`explains('Only Feasible',x,s)` ⟵ `getPrefix(x,s,`$\mathtt{s}_{sel}$`)` ∧ `onlyFeasible(x,`$\mathtt{s}_{sel}$`)`

`explains('Only Preferred',x,s)` ⟵ `getPrefix(x,s,`$\mathtt{s}_{sel}$`)` ∧ `onlyPreferred(x,`$\mathtt{s}_{sel}$`)`

6 Simulation Analysis and Implementation

The formalized models are useful for both simulating the explanation services of the system-to-be in order to support the process of discovering and specifying explanation requirements and, assuming a compliant implementation of the

required system, forming the backbone of an explanation service prototype. A prototypical implementation of the explanation engine and the example of Figure 1 is available at the associated online code repository [30].

Exploring Explanations via Simulation. Simulation is based on generating possible action histories and asking explanation questions based on them. Specifically, Golog's predicate `FindSit(p(...),S0,s)` renders situations `s` which result from executing procedure `p(...)`, starting from the initial situation S_0. By setting `p(...)` to be the procedure that maps to the top-level goal, the resulting situation `s` maps to an i^+ history that captures one way of fulfilling the corresponding goal. The situations can be subsequently passed as the input situation of the explanation templates and rules. To allow such simulations, analysts will need to instantiate assertion descriptions – including these in the intialization element – with concrete objects (see Section 2).

Returning to our example of Figure 1, let us hypothesize that *Abdul*, the meeting initiator, wants to ask organizer, *Matilda*, to schedule a meeting with three participants, *Xing*, *Amr*, and *Naya*. The first two have their on-line calendars up-to-date, and *Alex*, the executive secretary, has the phone numbers of the last two. We further know that the organizer wants to promote *Quick Scheduling*.

After having Golog generate a hypothetical history `e1`, as seen in the top frame of Figure 4, we observe that Alex collected (at the behest of Matilda) Naya's constraints by phone (task #07). We request an ad-hoc explanation for this action, in the first question (`Q1`) at the bottom frame of Figure 4. As ad-hoc explanations are agnostic to the purpose or context of the sought explanation, predicate `explain` will consider all possible ways by which axioms can be instantiated to provide a response. The outcomes are answers `A1.1 - A1.3`. The first (`A1.1`) is a teleological explanation making use of dependency Axiom #3 to inform that said action took place at the request of the organizer, Matilda. The second explanation (`A1.2`) is a canonical one, and points to a subsequent task (#08 in the upper frame) whose performance requires prior performance of the task in question. The third (`A1.3`) informs that this is the only possible alternative; indeed Naya does not maintain an on-line calendar and there is no other way to collect constraints. Out of these answers, the explainee may focus on the one they intended with their question. Note also that the natural language in the answer is due to simple templates translating the atomic formulae in parentheses into more accessible descriptions.

The second question (`Q2`) of Figure 4 offers similar answers for the question why Amr's constraints were collected from the calendar. In this case, we have an additional canonical explanation `A2.3` stemming from the precedence between an ancestor of the task in question, namely *Have Scheduling Performed(Abdul)*, to goal *Meeting Attended(Abdul)*. In other words, collecting constraints – in the case of Amr from his calendar – is necessary for the meeting to be eventually attended. This explanation is likely too distant from the explanandum to be of interest. In general, since ad-hoc questions lack context, they may include several uninteresting explanations in response. For canonical explanations, specifically, the level of ancestry at which explanation producing precedence constraints are

Simulated History e1

```
– 01: s0
– 02: decideMeetingDetails(abdul)
– 03: delegate(abdul,matilda,haveSchedulingPerformedByOrganizer(abdul,matilda)
                                                    haveMeetingScheduled(matilda))
– 04: collectFromCallendar(matilda,xing)
– 05: collectFromCallendar(matilda,amr)
– 06: delegate(matilda,alex,haveSecretaryCallPP(matilda,alex,naya),
                                  collectConstraintsByPhone(matilda,alex,naya))
– 07: collectConstraintsByPhone(matilda,alex,naya)
– 08: chooseTimeandDate(matilda)
– 09: announceMeeting(matilda,xing)
– ...
```

Explanations

```
Q1: explain(collectConstraintsByPhone(matilda,xing,naya), e1)
  – A1.1: Teleological Explanation: Because organizer matilda wanted to
    have secretary alex call participant naya to collect their constraints.
    (haveSecretaryCallPP(matilda,alex,naya)) [dependency (axiom 3)]
  – A1.2: Canonical Explanation: Because organizer matilda wanted to choose time and
    date for the meeting. (chooseTimeandDate(matilda)) [enablement (axiom 4)]
  – A1.3: Choice mandatory, no other feasible alternative. (...) [only feasible
    (axiom 5)]

Q2: explain(collectFromCallendar(matilda,amr), e1).
  – A2.1: Teleological Explanation: Because organizer matilda wanted to collect
    amr's timetable. (timetableCollected(matilda,amr)) [operationalization (axiom
    1)]
  – A2.2: Canonical Explanation: Because organizer matilda wanted to choose time and
    date for the meeting. (chooseTimeandDate(matilda)) [enablement (axiom 4)]
  – A2.3: Canonical Explanation: Because organizer abdul wanted to have the meeting
    attended. (meetingAttended(abdul)) [enablement (axiom 4)]
  – A2.4: From the available choices this is the only preferred one. (...) [only
    preferred from feasible (axiom 6)]

Q3: explain(whyisPP(amr), e1).
  – A3.1: Because amr needs to report on agenda items 3 and 4.
```

Fig. 4. Simulated history and ad-hoc explanations.

sought (see Axiom 4.2) can be limited to avoid explanations that are unlikely to be of interest or even comprehensible. This limit is exceeded in `Q1` hence the additional canonical explanation does not show up. The final answer for `Q2`, `A2.4.`, shows that while for collecting Amr's constraints both options are available, the one that aligns with the promotion of the *Quick Scheduling* quality appears to have been chosen. Note that this is not a definite but a plausible explanation, as how exactly the variability-binding decision is made is, as we saw, subject to optimization procedures outside of the model. Finally, the third question `Q3` is not ad-hoc but based on a pre-designed template from Figure 3.

As part of the explanation requirements exploration process, analysts may add or remove constraints and initialization clauses from/to the model, and further steer the interpreter to generate situations which satisfy certain properties, e.g., omit or necessarily include a specific action/task. The exercise allows testing

the efficacy of the explanation axioms to the given domain and, when the latter fall short, triggers development of custom templates. To facilitate exploration, in our prototypical implementation [29] the possibility of interactive ad-hoc explanation exploration is also available. Following the initial explanation question, the possible answer literals are listed prompting the explainee to choose one of them for adoption as the explanandum of the next round.

Implementing the Explanation Service. Let us further assume that the systems supporting the socio-technical structure captured in i^+ are implemented (a) in full compliance with the i^+ model, (b) such that performance of every i^+ task is captured and logged, (c) instances of top-level goal fulfillment have concrete identifiers, such that different instances of top-level goal fulfillment can be extracted from logs. In that case, the formalization of the axioms and templates can be used as-is for the generation of explanations. Thus, in the meeting scheduling example of Figure 1, the name and parameter instances of leaf-level tasks are appended to the log upon their performance, and, hence, in the order in which they are performed. By requiring the value of parameter `m` to identify a specific meeting of interest, a pre-processor can then extract the tasks that relate to that meeting and construct a history such as the one in Figure 4, top frame. The history is then used according to the process sketched above.

The assumption of proper logging becomes more and more plausible as participating actors and activities are increasingly automated, as can be the case with the Meeting Scheduler and the Secretary in our example which can nowadays both be envisioned as AI-powered systems. In the history of Figure 4, actions pertaining to constraint collection (`04`, `05`, `07`) or announcements (`09 - ...`) can easily be envisioned as present in an audit log of an system that supports scheduling.

7 Related Work

The topic of explanations spans multiple disciplines and a huge literature that stretches back to Aristotle and his seminal work on causality in Nature that laid the foundations for Science and scientific explanations [16]. An explanation is about a set of phenomena (the *explananda*) that needs to be generated by an explainer and help an explainee understand the phenomena. For physical phenomena, such as a ball dropping to the ground when it is let go, the explainer is a scientist who comes up with laws, the canons of Nature, that explain the phenomena. For computational phenomena, such as an action of an AV, the explainer is the AV itself while the explainee is a stakeholder (passenger, owner, auditor, etc.). We focus our discussion here only to works where the explainer is a software-intensive system and the explananda are the system's actions.

Our proposal treats explanation requirements for such a system as *"why"* questions with associated answer templates. These requirements are operationalized by endowing the system with functions that possess requirements and design information through conceptual models and acquire runtime information so that they can generate answers to each *"why"* question in accordance with the cor-

responding answer templates. This approach vis-à-vis the form of explanation requirements was inspired by *competency questions* used for scoping and evaluating formal ontologies (e.g., [17]).

There has been considerable recent interest in explainability within the Requirements Engineering (RE) community. Chazette, Kohl, Schneider et al. have studied explainability in a series of papers [3,4,5,6,14,24] as a non-functional requirement that measures how well explanations generated by a system contribute to a better understanding of the system by its stakeholders. Their work has a much broader scope than what is presented herein, as it includes other types of explanations well beyond the teleological and canonical explanations treated here. Moreover, they acknowledge that explanations have to be contextual, taking into account an explainee's background and the circumstances under which the system action, the explanandum, was carried out. In addition, they investigate how to evaluate explainability for a given system through empirical studies [39]. Finally, they study how explainability influences and is influenced by other non-functional requirements, such as transparency [10,11], understandability, and usability using Softgoal Interdependency Graphs (SIGs) [7]; Sterz et al. use the umbrella term *perspicuity* to refer to this class of qualities [42]. Efforts such as that of Berani et al. [2] propose a framework for organizing concepts surrounding explainability at different granularity levels. At the same time, the motivation behind seeking explanations has been studied by Sadeghi et al. [36], and includes training, validation, debugging and interaction; all such situations can be supported by our proposal. Consistent also with our i^*-based approach an explainee-oriented elicitation practice has been proposed through the use of personas [15]. Elsewhere, Mann et al. [32] offers various categories of opacity that explainability aims at addressing, of which complexity (in our case: variability of system outputs/behaviors) and epistemic dependency (actors possessing different pieces of the explanation puzzle) can be seen as suitable targets of our approach. Explanation in road navigation systems seems to be a popular exemplar (e.g., [5]) and Alsheeb and Brandão actually offer an interesting case study of an explainable road navigation system that can answer *"why is path A fastest, rather than path B?"*, and *"why does the fastest path not go through point W?"* by leveraging a combination of inverse optimization and diverse shortest path methods. In general, rather than studying explainability as a quality or offering technical solutions to specific problems (e.g., road navigation), our effort is oriented towards developing tools and techniques for systematic elicitation, modeling, analysis, and implementation of explanation requirements of specific kinds – namely, here, observed system actions – for a broader range of socio-technical systems.

The pursuit of explainable AI (XAI) has been the topic of investigation in the AI community as well. The roots of XAI lie in expert systems and work done in the 70's by W.J. Clancey [8]. The topic has been explored from multiple angles ever since. For instance, in his early work, Shanahan [37] proposed a deductive and an abductive approach to explanation in the situation calculus, both of which are based on default reasoning. In [13], Dennis and Oren

used dialogue between the user and a Belief-Desire-Intention (BDI) agent system to explain why the agent has chosen a particular action. Their approach aims to identify any divergence of views that exist between the user and the BDI agent relative to the latter's behaviour and allows for an interactive and user-friendly explanation process. In his Structural Equations Models-based formalization, Beckers [1] presented formal definitions of various causal notions of explanation and proposed to use actual causation for the purpose of explainable AI. The connections between these notions and the consequences of ignoring the causal structure are explored. Miller [34] proposed causal models-based *contrastive explanation* that asks *"why P rather than Q?"* in his attempt to enhance understanding and trust in AI decision-making. In [31], SEM-based causal models are utilized to generate explanations of the behaviour exhibited by model-free reinforcement learning agents. Sridharan et al. [40,41] proposed an explainable robotic architecture by integrating stepwise refinement, non-monotonic reasoning, probabilistic planning, and interactive learning. Khan and Rostamigiv [23] appealed to theory of mind and showed how causal analysis of agents' knowledge and their motivation along with a goal recognition module can be employed to explain agent behaviour in communicative multiagent contexts. Comparatively, we aim at supporting analysis of explanation requirements using an established modeling framework, and basing explanations on information within the models.

8 Conclusions

We presented a goal-oriented framework for modeling, analyzing, simulating, and prototyping explanation requirements for actions of software-intensive systems. We utilize i^+, an extension of i^* that allows temporal and causal relationships among intentional elements. Subsequently, explanation requirements are captured through explanation goals, are assigned to various explainee roles, and are operationalized into actions that capture run-time explanation information or render explanation interactions, the format of which is described through templates. Further, a set of axioms exploits information embedded in the i^+ models to offer possible answers to ad-hoc explanation questions. By appropriately formalizing the model, both simulations and, under certain assumptions, prototyping of the envisioned explanation interactions are possible.

Future work aims at empirically evaluating both the proposed analysis approach and the quality of explanations produced by our axioms through case studies and experiments. We, further, plan to broaden the kinds of explanations our system can offer, including explanations for exceptional system actions, action failures, and, generally, expected actions that did not occur, such as counterfactual explanations [18] – our current axioms are restricted to explaining actions that have occurred. In addition, we are interested in refining our axiomatization to include ad hoc explanations that are context- and explainee-sensitive [35]. We, finally, wish to study how systems can be engineered in full compliance with the explanation requirements model, especially in the context of learning-based systems, whose decision-making rules are known to be opaque.

References

1. Beckers, S.: Causal Explanations and XAI. In: Schölkopf, B., Uhler, C., Zhang, K. (eds.) Proceedings of the 1st Conference on Causal Learning and Reasoning, (CLeaR 2022). vol. 177, pp. 90–109. PMLR (2022), https://openreview.net/forum?id=pJuO-5QEKa
2. Bersani, M.M., Camilli, M., Lestingi, L., Mirandola, R., Rossi, M., Scandurra, P.: A Conceptual Framework for Explainability Requirements in Software-Intensive Systems. In: Proceedings of the 31st IEEE International Requirements Engineering Conference Workshops, REW 2023. pp. 309–315 (2023). https://doi.org/10.1109/REW57809.2023.00059
3. Chazette, L., Karras, O., Schneider, K.: Do End-Users Want Explanations? Analyzing the Role of Explainability as an Emerging Aspect of Non-Functional Requirements. In: Proceedings of the 27th IEEE International Requirements Engineering Conference (RE'19). pp. 223–233 (2019). https://doi.org/10.1109/RE.2019.00032
4. Chazette, L., Brunotte, W., Speith, T.: Exploring Explainability: A Definition, a Model, and a Knowledge Catalogue. In: Proceedings of the 29th IEEE International Requirements Engineering Conference (RE'21). pp. 197–208 (2021). https://doi.org/10.1109/RE51729.2021.00025
5. Chazette, L., Brunotte, W., Speith, T.: Explainable software systems: from requirements analysis to system evaluation. Requirements Engineering **27**(4), 457–487 (2022). https://doi.org/10.1007/s00766-022-00393-5
6. Chazette, L., Schneider, K.: Explainability as a non-functional requirement: challenges and recommendations. Requirements Engineering **25**(4), 493–514 (2020). https://doi.org/10.1007/s00766-020-00333-1
7. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Springer (1999). https://doi.org/doi.org/10.1007/978-1-4615-5269-7
8. Clancey, W.J.: An Antibiotic Therapy Selector which Provides for Explanations. In: Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI'77). p. 858 (1977)
9. Cohen, J.: Teleological Explanation. Proceedings of the Aristotelian Society **51**, 255–292 (may 1950), http://www.jstor.org/stable/4544486
10. Cysneiros, L.M.: Using i* to elicit and model transparency in the presence of other non-functional requirements: A position paper. In: Proceedings of the 6th International i* Workshop (iStar 2013), CEUR Vol-978. vol. 978, pp. 19–24 (2013), https://ceur-ws.org/Vol-978/paper_4.pdf
11. Cysneiros, L.M., Raffi, M., Leite, J.C.S.D.P.: Software transparency as a key requirement for self-driving cars. In: Proceedings of the 26th IEEE International Requirements Engineering Conference (RE'18). pp. 382–387. IEEE (2018). https://doi.org/10.1109/RE.2018.00-21
12. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 Language Guide. The Computing Research Repository (CoRR) **abs/1605.0** (2016), http://arxiv.org/abs/1605.07767
13. Dennis, L.A., Oren, N.: Explaining BDI agent behaviour through dialogue. Autonomous Agents & Multi Agent Systems **36**(1), 29 (2022). https://doi.org/10.1007/s10458-022-09556-8
14. Deters, H., Droste, J., Obaidi, M., Schneider, K.: How Explainable Is Your System? Towards a Quality Model for Explainability. In: Mendez, D., Moreira, A.

(eds.) Proceedings of the 30th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2024). pp. 3–19. Springer Nature Switzerland, Cham (2024). https://doi.org/doi.org/10.1007/978-3-031-57327-9_1

15. Droste, J., Deters, H., Puglisi, J., Klunder, J.: Designing End-User Personas for Explainability Requirements Using Mixed Methods Research. pp. 129–135 (2023). https://doi.org/10.1109/REW57809.2023.00028
16. Falcon, A.: Aristotle on Causality. In: Zalta, E.N., Nodelman, U. (eds.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, Spring 2 edn. (2023), https://plato.stanford.edu/archives/spr2023/entries/aristotle-causality/
17. Grüninger, M., Fox, M.S.: The Role of Competency Questions in Enterprise Engineering. In: Rolstadås, A. (ed.) Benchmarking — Theory and Practice, pp. 22–31. Springer US, Boston, MA (1995). https://doi.org/10.1007/978-0-387-34847-6_3, https://doi.org/10.1007/978-0-387-34847-6_3
18. Guidotti, R.: Counterfactual explanations and how to find them: literature review and benchmarking. Data Mining and Knowledge Discovery (2022). https://doi.org/10.1007/s10618-022-00831-6, https://doi.org/10.1007/s10618-022-00831-6
19. Guizzardi, G., Guarino, N.: Explanation, semantics, and ontology. Data & Knowledge Engineering **153**, 102325 (2024). https://doi.org/https://doi.org/10.1016/j.datak.2024.102325, https://www.sciencedirect.com/science/article/pii/S0169023X24000491
20. Honderich, T.E.: Explanation (2005), https://www.oxfordreference.com/display/10.1093/acref/9780199264797.001.0001/acref-9780199264797-e-838
21. Josephson, J.R., Josephson, S.G. (eds.): Abductive inference: Computation, philosophy, technology. Cambridge University Press, New York, NY, US (1996)
22. Kastner, L., Langer, M., Lazar, V., Schomacker, A., Speith, T., Sterz, S.: On the Relation of Trust and Explainability: Why to Engineer for Trustworthiness. In: Proceedings of the 29th IEEE International Conference on Requirements Engineering Workshops (REW'21). pp. 169–175 (2021). https://doi.org/10.1109/REW53955.2021.00031
23. Khan, S.M., Rostamigiv, M.: On Explaining Agent Behaviour via Root Cause Analysis: A Formal Account Grounded in Theory of Mind. In: Gal, K., Nowé, A., Nalepa, G.J., Fairstein, R., Radulescu, R. (eds.) Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023). Frontiers in Artificial Intelligence and Applications, vol. 372, pp. 1239–1247. IOS Press (2023). https://doi.org/10.3233/FAIA230401
24. Köhl, M.A., Baum, K., Langer, M., Oster, D., Speith, T., Bohlender, D.: Explainability as a Non-Functional Requirement. In: Proceedings of the 27th IEEE International Requirements Engineering Conference (RE'19). pp. 363–368 (2019). https://doi.org/10.1109/RE.2019.00046
25. Levesque, H., Pirri, F., Reiter, R.: Foundations for a Calculus of Situations. Electronic Transactions of AI (ETAI) **2**(3–4), 159–178 (1998), https://ep.liu.se/ej/etai/1998/005/
26. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. The Journal of Logic Programming **31**(1-3), 59–83 (1997). https://doi.org/10.1016/S0743-1066(96)00121-5

27. Liaskos, S., Khan, S.M., Mylopoulos, J.: Modeling and reasoning about uncertainty in goal models: a decision-theoretic approach. Software & Systems Modeling **21**, 1–24 (2022). https://doi.org/https://doi.org/10.1007/s10270-021-00968-w
28. Liaskos, S., McIlraith, S., Sohrabi, S., Mylopoulos, J.: Representing and reasoning about preferences in requirements engineering. Requirements Engineering Journal (REJ) **16**, 227–249 (2011). https://doi.org/10.1007/s00766-011-0129-9
29. Liaskos, S., Mylopoulos, J., Borgida, A., Khan, S.M.: Model-driven analysis and design of explanation services (long technical report). Tech. rep., York University (2024), https://hdl.handle.net/10315/42239
30. Liaskos, S., Mylopoulos, J., Borgida, A., Khan, S.M.: Xp-i: extracting explanations from models (2024), https://github.com/cmg-york/xp-i
31. Madumal, P., Miller, T., Sonenberg, L., Vetere, F.: Explainable Reinforcement Learning through a Causal Lens. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI. pp. 2493–2500. AAAI Press (2020)
32. Mann, S., Crook, B., Kastner, L., Schomacker, A., Speith, T.: Sources of Opacity in Computer Systems: Towards a Comprehensive Taxonomy. In: Proceedings of the 31st IEEE International Requirements Engineering Conference Workshops (REW 2023). pp. 337–342 (2023). https://doi.org/10.1109/REW57809.2023.00063
33. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence **267**, 1–38 (2019). https://doi.org/https://doi.org/10.1016/j.artint.2018.07.007
34. Miller, T.: Contrastive explanation: a structural-model approach. The Knowledge Engineering Review **36**, e14 (2021). https://doi.org/10.1017/S0269888921000102
35. Sadeghi, M., Herbold, L., Unterbusch, M., Vogelsang, A.: SmartEx: A Framework for Generating User-Centric Explanations in Smart Environments. In: 2024 IEEE International Conference on Pervasive Computing and Communications (PerCom). pp. 106–113. IEEE Computer Society, Los Alamitos, CA, USA (mar 2024). https://doi.org/10.1109/PerCom59722.2024.10494449
36. Sadeghi, M., Klos, V., Vogelsang, A.: Cases for Explainable Software Systems: Characteristics and Examples. In: Proceedings of the IEEE International Conference on Requirements Engineering Workshops (REW'21). pp. 181–187 (2021). https://doi.org/10.1109/REW53955.2021.00033
37. Shanahan, M.: Explanation in the Situation Calculus. In: Bajcsy, R. (ed.) Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93). pp. 160–165. Morgan Kaufmann (1993), https://dl.acm.org/doi/abs/10.5555/1624025.1624048
38. Shvo, M., Klassen, T.Q., McIlraith, S.A.: Towards the Role of Theory of Mind in Explanation. In: Calvaresi, D., Najjar, A., Winikoff, M., Främling, K. (eds.) Proceedings of the Second International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems (EXTRAAMAS 2020). Lecture Notes in Computer Science, vol. 12175, pp. 75–93. Springer (2020)
39. Speith, T.: How to Evaluate Explainability? - A Case for Three Criteria. Proceedings of the IEEE International Conference on Requirements Engineering pp. 92–97 (2022). https://doi.org/10.1109/REW56159.2022.00024
40. Sridharan, M.: REBA-KRL: Refinement-Based Architecture for Knowledge Representation, Explainable Reasoning and Interactive Learning in Robotics. In: Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarin, A., Lang,

J. (eds.) Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 2935–2936. IOS Press (2020), https://doi.org/10.3233/FAIA200461

41. Sridharan, M., Meadows, B.: Towards a Theory of Explanations for Human-Robot Collaboration. Künstliche Intelligenz **33**(4), 331–342 (2019), https://doi.org/10.1007/s13218-019-00616-y
42. Sterz, S., Baum, K., Lauber-Ronsberg, A., Hermanns, H.: Towards Perspicuity Requirements. In: Proceedings of the 29th IEEE International Conference on Requirements Engineering Workshops (REW'21). pp. 159–163 (2021). https://doi.org/10.1109/REW53955.2021.00029
43. The Editors of Encyclopedia: Explanation (2017), https://www.britannica.com/topic/explanation
44. Yu, E.S.K., Mylopoulos, J.: Understanding "Why" in software process modelling, analysis, and design. In: Proceedings of the 16th International Conference on Software Engineering (ICSE'94). pp. 159–168. Sorrento, Italy (1994), https://dl.acm.org/doi/10.5555/257734.257757