Noname manuscript No.
(will be inserted by the editor)

Modeling and Reasoning about Uncertainty in Goal Models: A Decision-Theoretic Approach

Sotirios Liaskos · Shakil M. Khan · John Mylopoulos

Received: date / Accepted: date

Abstract Goal models have been a popular subject of study by researchers in Requirements Engineering, due to their ability to capture and analyze alternative solutions through which a software system can achieve business objectives. A plethora of analysis methods for automated identification of optimal alternatives has been proposed. However, such methods often assume an idealized reality where all tasks are successfully performed when attempted and all goals are eventually satisfied with certainty when pursued according to a solution. In reality, some tasks run the risk of failure while others produce chance outcomes. In this paper, we extend the standard goal modeling language to allow representation and reasoning about both uncertainty and preferential utility in goals. Tasks are extended to allow for probabilistic effects and preferential statements of stakeholders are captured and translated into utilities over possible effects. Moreover, solutions are not mere specifications (functions, quality constraints, and assumptions), but rather policies, that is sequences of situational action decisions, through which stakeholder goals can be fulfilled. An AI reasoning tool is adapted and used for identifying optimal policies with

Sotirios Liaskos
School of Information Technology
York University
4700 Keele Street, Toronto, ON
E-mail: liaskos@yorku.ca

Shakil M. Khan
Department of Computer Science
University of Regina
3737 Wascana Parkway, Regina, SK
E-mail: Shakil.Khan@uregina.ca

John Mylopoulos
Department of Computer Science
University of Toronto
214 College St, Toronto, ON
E-mail: jm@cs.toronto.edu

respect to the value they offer to stakeholders measured against their probability of failure. Evaluation of the approach includes a simulation study and scalability experiments to assess the applicability of automated reasoning for larger problems.

Keywords Goal Modelling, Markov Decision Processes (MDP), DT-Golog, Golog

1 Introduction

Goal models have long been studied as tools for capturing, representing, and organizing stakeholder intentions within a variety of application contexts such as software engineering and business analysis [3, 10, 52]. One of their most appealing features is their ability to show how top-level strategic objectives of stakeholders can be analyzed into low-level actions that can be performed for the fulfillment of these objectives [40]. Moreover, they allow the concise representation of high degrees of variability in the ways by which the low-level actions fulfill top-level objectives, capturing large numbers of interrelated decisions. Several automated reasoning techniques have been proposed for computer-aided analysis of such variability of solutions [2, 19, 31, 34, 43] (see also [24] for a survey). Such tools search for groups of actions that both fulfill high-level functional goals and at the same time bring about optimal value concerning criteria of interest.

Typically in goal modeling and related techniques for automated reasoning, actions are deterministic, that is, they are assumed to bring about one and only one desired effect with certainty every time they are performed. However, in reality, actions are rarely deterministic. Instead, they bring about different outcomes with different probabilities. Some actions, such as sending a message, or saving a document, have a unique intended outcome but, with some probability, they may fail to deliver it due to human or system errors. Other actions, such as making a monetary investment or participating in a competition such as an election, embody non-determinism by design in that they can bring about different outcomes, each with different probability, desirability, and qualities. Given such a non-deterministic interpretation of actions, the value of performing the action concerning specific criteria depends on the outcome of the action rather than the mere fact that it has been attempted.

In this paper, we propose an extension of the standard iStar 2.0 [10] modeling notation to model and automatically reason about optimal solutions to a requirements problem in the presence of non-deterministic actions. To reason about optimality, we need solutions that are not mere specifications, i.e., functions, quality constraints, and assumptions for a system-to-be, but rather *policies*, that is, choices of actions to be taken under given circumstances in order to fulfill stakeholder goals. Accordingly, in this work, a solution to a requirements problem is a policy rather than a specification. The iStar 2.0 concepts for modeling agent action, tasks, are accompanied by effects, which are special constructs that describe alternative outcomes that emerge from

their performance. Higher-level goal and quality criteria satisfaction is defined through the use of AND/OR decompositions and contribution links based on such effects. Diagrammatic and tabular representations are proposed for representing probabilities and values of effects and combinations thereof. These extensions allow the translation of the resulting model into DT-Golog, a language for describing decision-theoretic action theories [48,49]. Through DT-Golog's automated reasoning tool, we can identify policies using tasks from the goal model that maximize expected value, taking into account both the probabilities of task outcomes and the values of these outcomes with respect to multiple criteria and the relative importance of such criteria. The reasoning exercise is most useful for design-time exploration and analysis of domains featuring complex actor operations and is aimed at improved understanding of the domain and increased accuracy of the model. We demonstrate the extension and the reasoning practice using an example inspired by the iStar 2.0 guide. To evaluate and further explore the technique we perform simulation and sensitivity analyses, as well as a scalability study.

The paper extends our earlier publication on the matter [33] in several ways including (a) complete treatment and presentation of the translation from Goal Models to DT-Golog, (b) a proposal to diagrammatically present the proposed extensions, (c) an exploration of probability-maximizing (in addition to just expected utility-maximizing) reasoning, (d) a new case and its analysis and validation through simulation, (e) an approach to performing sensitivity analysis for utility elicitation, (f) new, extended tool scalability analysis. The translation details specifically allow further independent validation, tool development, and extension of the framework by the research community.

The paper is organized as follows. In Section 2 we review the goal modeling language we adopt and motivate the proposed extensions and techniques. In Section 3 we present the proposed extension, while in Section 4 we discuss the practice of automatically reasoning with it. Section 5 provides details of the translation from the extended goal model to DT-Golog, and in Section 6 we describe the steps we took to further evaluate and understand the proposed technique, including simulation, sensitivity, and scalability analysis. We present related work in Section 7 and conclude in Section 8.

2 Background and Running Example

2.1 Goal Models

The goal models we consider in this work are based on the i^* family [3,52] and specifically the latest iStar 2.0 standard [10]. An example of such a model can be seen in Figure 1. The model in the Figure is inspired by the guiding example of the iStar 2.0 document [10]. In that model, the intentions and tasks related to booking and securing reimbursement for a trip by a research student are modeled. Here we consider a more elaborate case of reimbursement. Specifically, the researcher in question is assumed to have a last-minute opportunity

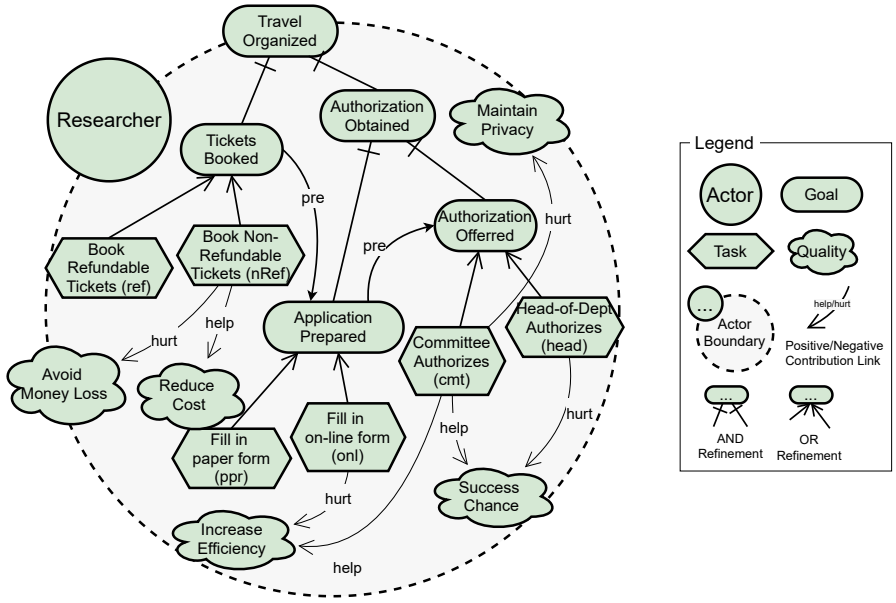


Fig. 1 Goal Model

to attend a research meeting. The researcher contemplates whether attending that meeting is worth the time and effort to travel there and decides that she would go only if she secures a reimbursement for that. However, given that this is last-minute, she should first book the tickets and then apply for the authorization needed for the reimbursement. Such authorization is not certain. She applies for it by filling a form and also adding a short research proposal and then submits it to either a committee or the head of the department, who operate on different budgets. Applying through the committee is, in some cases, more likely to lead to an approval but takes more time. Specifically, if the application is filled using a paper form, it is reviewed by a different, more lenient subcommittee, compared to when it is filled on-line. That option is better than letting the head of the department approve it, as the latter manages a smaller budget but still receives many requests. On the other hand, having the entire committee know the actor's research endeavors may be uncomfortable to the researcher.

In the model, *goals* (e.g. *'Tickets Booked'*), that is, states of affairs desired by *actors* (represented using circular elements – e.g. *'Researcher'*), are modeled through oval-shaped elements and recursively refined into subgoals and eventually *tasks* [10]. The latter, depicted through hexagon-shaped elements represent actions that an actor wants to be executed to achieve the goals as per the refinement structure, such as *'Book Refundable Tickets'* in the figure. The cloud-shaped elements are *qualities* (also: *quality goals*, we use the two terms interchangeably), i.e. attributes for which an actor desires some level of

achievement. Qualities can be precise, i.e., allow for readily measurable and clear-cut criteria of achievement, such as *'Errors reduced by 10%'* or imprecise as in *'Reduce Errors'* – both types are relevant in our work. Modeling goals in the absence of a precise definition for their satisfaction reflects the need to represent and reason about qualities of options in the early stages of analyzing a problem and/or when getting objective evidence of satisfaction is not possible.

Goal refinement is represented through *AND-refinement* and *OR-refinement* links. The AND-refinement implies that every sub-goal of the parent goal must be fulfilled for the parent goal to also be considered fulfilled. The OR-refinement signifies that fulfillment of one of the sub-goals suffices for assuming that the parent goal is fulfilled as well. Furthermore, quality goals can be the targets of *contribution links*, labelled with *help* or *hurt*. A *help* (respectively *hurt*) link signifies that if the origin of the link is satisfied or performed, this constitutes evidence of satisfaction (resp. denial) of the destination. Contribution links offer a rough indication of how a choice of an option within an OR-decomposition affects our belief about the satisfaction of a quality goal. For example, an alternative that includes {`nRef`,`onl`,`head`} (referring to the abbreviations in the diagram of Figure 1) is preferable if we are interested in the goal *'Reduce Cost'*, due to the presence of `nRef` but not good for *'Success Chance'* due to the presence of `head`. Thus, depending on our priority between those two high-level goals, the above or some other alternative is suitable for the top goal.

Note finally that, to remind us that all the above intentional elements are desired by an actor (*'Researcher'*), the elements are included within the scope of the *actor boundary* associated with the specific actor.

2.2 Probabilistic Effects of Tasks

Traditionally, goal models do not consider uncertainty concerning task executions and their impact on goal fulfillment. In reality, however, tasks do not always have the same outcome. Some tasks which have a unique success outcome may fail, and other tasks have multiple possible outcomes each with a different probability. Recognizing that tasks are of this nature, requires us to adopt a different view of how we model and reason with them within goal models.

Let us go back to our example of Figure 1 and explore how probabilities become relevant, by also adding detail to the example. Any task in the model such as *'Book Refundable Tickets'* always carries a probability of not being performed properly for a variety of reasons. For example, the researcher may have thought they booked it but they forgot to finish the booking, they may have booked the wrong flight or time, or the company promising the booking may not deliver. In consequence, *'Tickets Booked'* and, by extension, *'Travel Organized'* are not guaranteed to succeed.

Alternative tasks *'Fill in paper form'* and *'Fill in online form'* also face the risk of failure for the application to reach its destination. In addition, while the actor can generally consider online forms to be more efficient, that will not turn out to be the case in the presence of errors and problems while completing it, including an unavailable server, lost passwords, etc. So the convenience of filling in an online form is only enjoyed with a certain probability; frustration and delays are experienced with a different probability. In that case, filling in a paper form may be more convenient, even when problems emerge that require help from the appropriate administrative officer. In other words, contributions to qualities can depend on the possible outcomes of the task from which the contribution originates, in a way that, in turn, affects the preferability of the task against other tasks.

While the previous tasks may unintentionally fail due to, e.g., error and mishaps, tasks *'Committee Authorizes'* and *'Head-of-Dept. Authorizes'* are expected to succeed or fail with a probability, by the nature or design of the underlying process or system. For example, both authorization tasks may assume a fixed success rate. Importantly, the probabilities of success may depend on probabilistic events captured elsewhere in the model. In our case, recall that paper applications are more likely to be accepted than online applications when sent to a committee. Online applications may have a better fate with the head of the department. In light of this information, the relationship between the choice of authorization body and *'Success Chance'* becomes more complex.

Note, finally, the choice between buying refundable and non-refundable tickets. The task *'Book Non-refundable Tickets'* helps *'Reduce Cost'* based on the idea that non-refundable tickets are less expensive. In practice, however, the dilemma between refundable and non-refundable could be a comparison between two more nuanced refund schedules (e.g., 50% refund for a 20% lower purchasing price versus 80% refund at original price), in a way that decision of which choice affects *'Reduce Cost'* the most depends on what is the likelihood that a refund will be needed, even when the actor is willing to take a risk.

To be able to express and reason with probabilistic effects of tasks and the impact of various outcomes on the satisfaction of goals and qualities, we introduce a series of decision-theoretic extensions, presented in the next section.

3 Goals, Probabilities and Utilities

3.1 Overview

We now turn our focus to the extensions we introduce to the standard goal modeling language. The core of an extended model contains a set of tasks $\mathcal{T}$, a set of goals $\mathcal{G}$ and a set of quality goals $\mathcal{O}$ as per the iStar 2.0 specification [10]. The following concepts are then introduced:

- A set D of *domain predicates.*
- A set $\mathcal{E}_t$ of *effects* for each task $t \in \mathcal{T}$; the effects are individuals or sets of elements drawn from D.

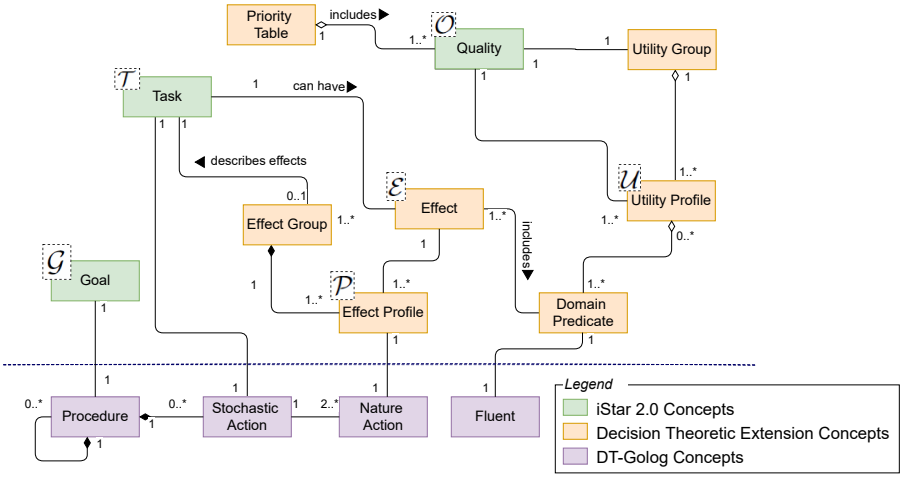


Fig. 2 Metamodel showing the extensions. Key concepts are annotated with the symbol used in the translation Algorithms 1, 2, 3 and 4 to denote the set of the concept instances in a specific model. In the algorithms, subscripts are used to denote specific subsets, e.g. $\mathcal{E}_t$ is the set of effects connected to task t. The complete iStar 2.0 metamodel can be found in the iStar 2.0 Language Guide [10].

- A set $\mathcal{P}_t$ of *effect profiles* for each task $t \in \mathcal{T}$, called the *effect group* of t. An effect profile is an effect augmented with a probability value and a condition under which the value holds.
- A set $\mathcal{U}_o$ of *utility profiles* for each quality $o \in \mathcal{O}$, called the *utility group* of o. A utility profile contains a utility value with respect to o and the condition, a formula over D, under which the utility value holds.
- *Effect groups* and *utility groups* can be represented diagrammatically or tabularly. In the latter format, they take the form of *effect tables* and *utility tables*, respectively.

A meta-model demonstrating the new concepts and how they relate is shown in Figure 2. We next describe the above in more detail.

3.2 Domain Predicates and Probabilistic Effects

Our first concern is to introduce constructs to represent what is true in the environment before, after, or independent of the performance of tasks. The set of *domain predicates* D is introduced for that purpose. Domain predicates typically represent the fact that, in the process of attempting to fulfill the root goal, a task has been performed and an effect has occurred as the result of such performance. For example, we could use domain predicates *'Non-Refundable Tickets Booked'*, *'Paper Submitted'*, or *'Committee Denied Authorization'* to denote outcomes of tasks *'Book Non-Refundable Tickets'*, *'Fill in paper Form'* and *'Committee Authorizes'*. However, any kind of contextual information can also be captured and used for reasoning irrespective of tasks, such as *'Travel*

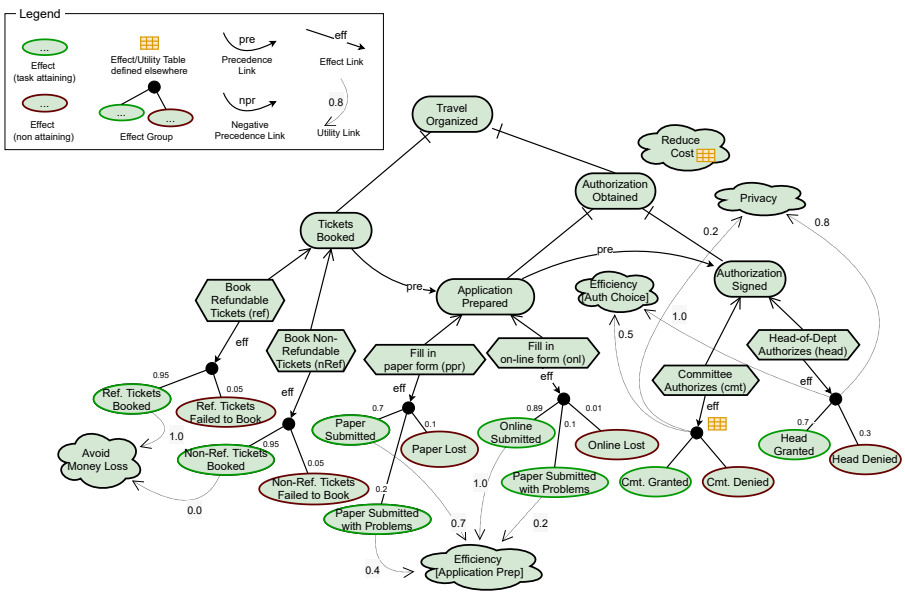


Fig. 3 Effect Model for Tasks

Season High', *'Number of Applicants High'*, *'Actor Dislikes Computers'*, *'University Closed'*. The space $S = 2^D$ of all possible combinations of truth values of the set of all domain predicates D, represents the space of all possible stages of a process towards fulfilling the root goal – though not all such stages are necessarily reachable given temporal constraints.

Tasks have the capability of causing changes to domain predicates effectively resulting in transitions within S. As we argued above, however, this rarely happens in a deterministic way. Rather, each task t has the potential of affecting the truth value of a number of domain predicates $Q_t \subseteq D$ and each with a different probability. We call these domain predicates *effect predicates* of t. The *probabilistic effects* (or simply *effects*) of tasks are tuples $e_t = (t, \vec{q_t})$ forming a set $\mathcal{E}_t$, where t is the task in question, $\vec{q_t}$ a set of elements from Q_t that the effect makes true leaving all other elements of Q_t unaffected.

For example, task $t =$ *'Book Refundable Tickets'* is associated with two effect predicates $Q_t = \{$*'Ref. Tickets Booked'*,*'Ref. Tickets Failed to Book'*$\}$, representing success and failure respectively. Upon performance of the task, effects $e_t^1 = (t, \{$*'Ref. Tickets Booked'*$\})$ and $e_t^2 = (t, \{$*'Ref. Tickets Failed to Book'*$\})$ may happen with different probabilities and turn the domain predicates mentioned in the effect true, leaving the rest of the domain predicates unaffected. Note that, for clarity and uniformity, we represent negative effects with designated predicates, (e.g. 'Task Failed') rather than negations of success predicates (such as ¬'Task Succeeded'). Our formalization ensures that only one of these effects is realized after the occurrence of the associated task.

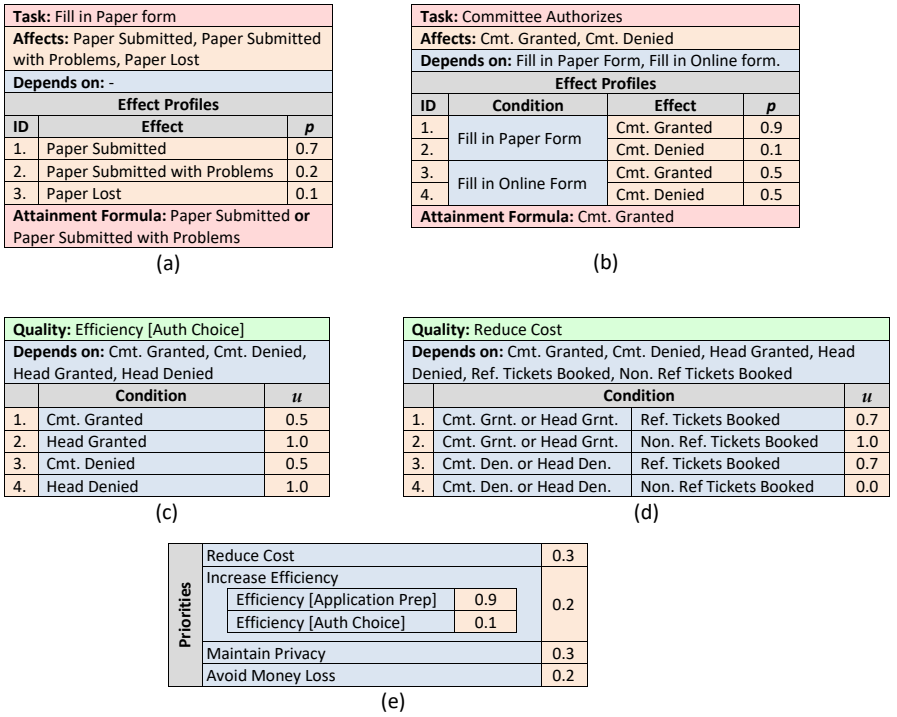
To associate probabilities with effects we use *effect profiles*. An effect profile is a triple (e_t, ϕ_{c_t}, p), signifying that, upon performance of t, effect e_t will happen with probability p if ϕ_{c_t} is true, where ϕ_{c_t} is a *condition formula* built from a subset of predicates $C_t \subset D$ that are relevant to the probability definition of effects for t; we call these the *condition predicates* of t. Thus, in the above example $t =$ *'Book Refundable Tickets'* is associated with two effect profiles, $(e_t^1, \mathbf{true}, 0.95)$, $(e_t^2, \mathbf{true}, 0.05)$. The effect profiles $\mathcal{P}_t$ associated with a task t are grouped together to signify this association, forming an *effect group*. The probabilities of the effect profiles of an effect group that share the same condition formula should add up to 1; as is the case in the above example.

An additional component of effect groups associated with task t is the *attainment formula* ϕ_t^{att}, constructed using elements from Q_t. The attainment formula represents the conditions under which the tasks can be considered to be successfully performed, in a way that can inform the satisfaction status of the parent and other high-level goals. Thus the attainment formula of task $t =$ *'Fill in paper form'* is $\phi_t^{att} =$ *'Paper Submitted'* **or** *'Paper Submitted With Problems'*.

Let us now see how we can diagrammatically visualize effects, effect profiles and effect groups and map them with tasks. For effects we use *beliefs*, a construct borrowed from the original *GRL* language [51]. These are represented as elliptical shapes in the *effect model* of Figure 3. Each belief element represents an effect and the elliptical shape contains the set $\vec{q_t} \in 2^{Q_t}$ the effect concerns; in the example, they all contain one predicate. Individual effects are grouped together as effect profiles into an effect group through a connection link to a common solid dot and an annotation on that link signifying the corresponding probability. Such representation of effect groups is preferable when C_t is empty, hence $\phi_{c_t} = \mathbf{true}$. The association of the task with its effect group is signified through *effect links* $\xrightarrow{eff}$ drawn from the task to such an effect group. Attainment formulae can be represented diagrammatically as well. In the example of Figure 3, color coding of the outline of the belief shapes signify which effects of the exclusive disjunction are considered to be *task attaining*, i.e. their satisfaction implies satisfaction of the attainment formula.

Whenever C_t is non-empty, meaning that the probabilities of the effects of a task depend on the truth status of other domain predicates, effect groups are difficult to represent diagrammatically. In that case, we use *effect tables* to represent effect groups. In Figure 4, such tables are shown. Each table mentions the task t it concerns, the set of affected domain predicates Q_t ("Affects"), and the a set of condition predicates C_t ("Depends on"). The table is populated with an exhaustive list of effect profiles, i.e., combinations of ϕ_{c_t} formulae, predicate sets $\vec{q_t}$ constituting effects, and the probability p of each. The tables also contain attainment formulae when diagrammatic representation thereof is difficult.

To see a complete example of an effect table, consider task *'Committee Authorizes'*. The task introduces a complex set of effects with a non-empty C_t in which the probability that the committee grants the authorization depends



Task: Fill in Paper form
Affects: Paper Submitted, Paper Submitted with Problems, Paper Lost
Depends on: -

Effect Profiles

ID	Effect	*p*
1.	Paper Submitted	0.7
2.	Paper Submitted with Problems	0.2
3.	Paper Lost	0.1

Attainment Formula: Paper Submitted **or** Paper Submitted with Problems

(a)

Task: Committee Authorizes
Affects: Cmt. Granted, Cmt. Denied
Depends on: Fill in Paper Form, Fill in Online form.

Effect Profiles

ID	Condition	Effect	*p*
1.	Fill in Paper Form	Cmt. Granted	0.9
2.		Cmt. Denied	0.1
3.	Fill in Online Form	Cmt. Granted	0.5
4.		Cmt. Denied	0.5

Attainment Formula: Cmt. Granted

(b)

Quality: Efficiency [Auth Choice]
Depends on: Cmt. Granted, Cmt. Denied, Head Granted, Head Denied

	Condition	*u*
1.	Cmt. Granted	0.5
2.	Head Granted	1.0
3.	Cmt. Denied	0.5
4.	Head Denied	1.0

(c)

Quality: Reduce Cost
Depends on: Cmt. Granted, Cmt. Denied, Head Granted, Head Denied, Ref. Tickets Booked, Non. Ref Tickets Booked

	Condition		*u*
1.	Cmt. Grnt. or Head Grnt.	Ref. Tickets Booked	0.7
2.	Cmt. Grnt. or Head Grnt.	Non. Ref. Tickets Booked	1.0
3.	Cmt. Den. or Head Den.	Ref. Tickets Booked	0.7
4.	Cmt. Den. or Head Den.	Non. Ref Tickets Booked	0.0

(d)

Priorities

Reduce Cost	0.3
Increase Efficiency	0.2
Efficiency [Application Prep] 0.9	
Efficiency [Auth Choice] 0.1	
Maintain Privacy	0.3
Avoid Money Loss	0.2

(e)

Fig. 4 Examples of effect tables [(a) and (b)], utility tables [(c) and (d)] and priority table (e)

on whether it was an online or paper form. Due to its complexity, the resulting effect group is difficult to be represented diagrammatically and, hence, a table-looking annotation is added in Figure 3, to inform of the existence of a detailed effect table outside the diagram. That effect table is shown in Figure 4(b). The table shows how, for example, the effect {*'Cmt. Granted'*} is different when *'Fill in paper form'* is performed (0.9) from when *'Fill in online form'* has been performed (0.5). Note that, for brevity, instead of condition predicates, the names of tasks are used in the table, implying reference to the attainment formula of the task. Thus, in our case condition *'Fill in paper form'* stands for its attainment formula *'Paper Submitted'* ***or*** *'Paper Submitted with Problems'*, as per Figure 4(a).

Finally, two additional kinds of links are introduced to the goal diagram, the $\xrightarrow{pre}$ link and the $\xrightarrow{npr}$ link – the former seen in both Figures 1 and 3. The links can be drawn from any goal or task to any other goal or task, provided that origin and destination are not part of the same path to the root. The former link, which we call *precedence* link, indicates that the destination task (resp. goal) cannot be performed (resp. attempted) unless the origin has been performed (task) or satisfied (goal) – as per the attainment formula. A goal is considered *attempted*, if any of the leaf level tasks that have that goal as an

ancestor is executed. Thus, a precedence link towards a goal is a shorthand for multiple precedence links towards each of the leaf level tasks under that goal. Likewise, the *negative precedence* link $\xrightarrow{npr}$ indicates that the destination task (resp. goal) cannot be performed (resp. attempted) if the origin has been performed (task) or satisfied (goal).

3.3 Probabilistic Goal Satisfaction, Utilities and Preferences

Interpreting tasks as probabilistic implies that overlaying goals, which tasks are meant to satisfy, are also achieved by a certain probability. Satisfaction of *hard goals* is defined by way of constructing conjunctions and disjunctions of leaf-level task attainment formulae, in a way that mirrors the AND/OR refinement structure. For example each of *'Application Prepared'* and *'Authorization Signed'* has a probability of success equal to the probability of the attainment formulae of the tasks chosen to perform for fulfilling each of the goals. Satisfaction of goal *'Authorization Obtained'*, in turn, is the product of the probabilities calculated for the aforementioned goals. Traversing the AND/OR tree upwards, we see that the root goal, e.g. *'Travel Organized'* in our case, has a probability of success based on the choices made at the OR-decompositions, which, in turn, imply the choice of different tasks at the leaf level and, as such, different probabilities of success at that level.

Satisfaction of *quality goals* is also assumed to depend on the outcome of the tasks rather than the mere fact that they were attempted. For example, it is relevant to say that booking non-refundable tickets hurts *'Avoid Money Loss'*, if we assume that the purchasing of such tickets was successful. If ticket purchase actually fails (e.g. the travel agent lost the order), money loss does not occur, and the contribution is not relevant. As another example, not shown in the figure, it may be known to the agent that flying with company A is a gamble comfort-wise: sometimes it is an extremely comfortable experience, while other times it is not. Hence, task *'Book with Company A'* makes a different contribution to quality goal *'Travel Comfort'* depending on the effect (e.g. *'Flight Comfortable'* vs. *'Flight Uncomfortable'*) in a way that it is difficult to draw the contribution without explicating these effects.

In the effect model of Figure 3, the original quality goals of Figure 1 have been refined, replaced, and/or abstracted and a different kind of contribution links now connect effects or effect groups with quality goals. To represent the impact of task effects to a quality goal o we define the *utility function* u_o of the goal to be a mapping from a combination of truth values of domain predicates to the real interval [0,1]: $u_o : S \mapsto [0, 1]$. Naturally, a subset $C_o \subseteq D$ of the domain predicates affects the value of u_o. Thus, each combination of truth values for the elements from C_o implies a different satisfaction value for the quality goal in question, 1.0 associated with full satisfaction and 0.0 associated with no satisfaction. We represent u_o as a collection $\mathcal{U}_o$ of *utility profiles*, i.e., triples $(o, \phi^i_{c_o}, u^i_o)$ of the quality goal o in question, the truth value combination

of domain predicates from C_o in the form of a formula $\phi^i_{c_o}$, and a utility value u^i_o. We call such a collection of utility profiles, *utility group*.

In simple cases in which C_o consists of a small set of mutually exclusive predicates, the corresponding utility profiles can be represented diagrammatically as in Figure 3. A special kind of links, *utility links* are utilized to represent a utility profile, decorated with the utility value of quality goal o when the origin effect or effect group is true. For example, for quality goal *'Privacy'*, the utility link from the effect group of $o =$ *'Committee Authorizes'* is decorated with 0.2, meaning that $u_o = 0.2$ if *'Committee Authorizes'* is performed irrespective of the actual effect. Note that although utility links are akin to contribution links, we prefer here to treat them as a separate construct due to their specific semantics and quantitative labeling.

When larger numbers of predicates affect u_o, diagrammatic representation is difficult. Thus, in a way similar to that of effect tables, we define *utility tables* in which the utility profiles of u_o are exhaustively listed. In Figures 4(c) and 4(d) we see examples of such utility tables for goals *'Efficiency [Auth Choice]'*, which is also represented diagrammatically in Figure 3, and *'Reduce Cost'* which is too complex to be represented diagrammatically.

Through utility tables we are able to map each state $s \in S$ to a utility value $u_o(s)$ with respect to each quality goal o. To obtain a total utility value U for each state in S, the individual u_o's need to be combined to one global quality value. We follow the preference specification approach introduced in our earlier work [30,31,35] and combine individual utilities through the formation of (nested, if needed) linear combinations.

Given a set of quality goals $o_1, o_2, \ldots, o_i$ of interest and $w_1, w_2, \ldots, w_i$ weights reflecting their relative importance, the total quality U of a state s is simply $U(s) = \sum_i w_i \times u_{o_i}(s)$. One way to represent such combinations is through a third kind of table, *priority tables*, as seen in Figure 4(e). For example, given the table, In a state s in which:

$$u_{ReduceCost}(s) = 0.7$$
$$u_{Efficiency[ApplicationPrep]}(s) = 0.2$$
$$u_{Efficiency[AuthChoice]}(s) = 0.1$$
$$u_{AvoidMoneyLosses}(s) = 1.0$$
$$u_{Privacy}(s) = 0.8$$

... the total utility is:

$$\begin{aligned} U(s) &= 0.3 \cdot u_{ReduceCost}(s) \\ &\quad + 0.2 \cdot [0.9 \cdot u_{Efficiency[App.Prep]}(s) + 0.1 \cdot u_{Efficiency[AuthChoice]}(s)] \\ &\quad + 0.3 \cdot u_{Privacy}(s) + 0.2 \cdot u_{AvoidMoneyLosses}(s) \\ &= 0.3 \cdot 0.7 + 0.2 \cdot [0.9 \cdot 0.2 + 0.1 \cdot 0.1] + 0.3 \cdot 0.8 + 0.2 \cdot 1.0 \\ &= \mathbf{0.688} \end{aligned}$$

As we discuss below, hierarchies of such priority distributions can be elicited following, e.g., AHP-like pairwise comparisons, as we have also shown in [30], or other techniques [45].

4 Reasoning with the Extended Model

With the goal model appropriately extended we are now in the position to automatically reason about optimal solutions of the goal tree on the basis of maximizing global utility, as specified in the utility and priority tables, taking into account the probability by which the utility will be observed as per the probability tables. For this purpose, we adopt *DT-Golog* [9,49], a tool that combines Golog-style [47] action theory-based high-level program execution and reasoning with *Markov Decision Processes* (MDPs) [50] to allow for the generation of policies that maximize cumulative expected utility.

At the level of the extended goal model we can think of a *policy* as a conditional task sequence, dictating what choice of action an agent should make at each stage of a process of fulfilling the root goal. Specifically a policy π would have the following form:

$$\begin{aligned}\pi = t;\ &\textbf{if } (t\text{'s } \mathit{outcome} = e_t^1) \textbf{ then } \pi_1\\ &\textbf{else if } (t\text{'s } \mathit{outcome} = e_t^2) \textbf{ then } \pi_2\\ &\ldots\\ &\textbf{else if } (t\text{'s } \mathit{outcome} = e_t^k) \textbf{ then } \pi_k\end{aligned}$$

where $\pi_1, \ldots, \pi_k$ are policies or elementary tasks t, the latter being either a task from the goal model, or one of two reserved instructions `stop`, denoting execution failure, or `nil` meaning the executing agent should just do nothing. The above is simply a concise representation of a DT-Golog policy that the DT-Golog reasoner returns when given the appropriate translation of the corresponding extended goal model. The policy can then be given to an executing agent to allow such agent decide what action to perform in each stage of a process to fulfil the root goal. The policy maximizes *cumulative expected utility* in a sense that repeated policy-compliant executions bring about, on average, higher utility than repeated executions of any other policy.

Let us consider the model of our example as extended in Figure 3 and the tables in Figure 4. For these specific numbers (Scenario 1) DT-Golog will return the top left policy of Figure 5, where the conditional task sequence constituting the policy is represented graphically. The policy implies that, in repeated occurrences of the need to fulfill the root goal, whenever there is a choice for booking the tickets, the researcher should book refundable ones, whenever there is a choice of mode of submission, she will submit the application online and whenever there is a choice to select authorization body, she will choose the head of the department. With this strategy, her probability of succeeding (i.e. having a ticket, a successfully submitted application, and the authorization, as per the root goal decomposition) would be 0.66 and the cumulative expected utility is 0.55. The $1.0-0.66 = 0.34$ probability refers to any combination of task performances that can go wrong, including not being able

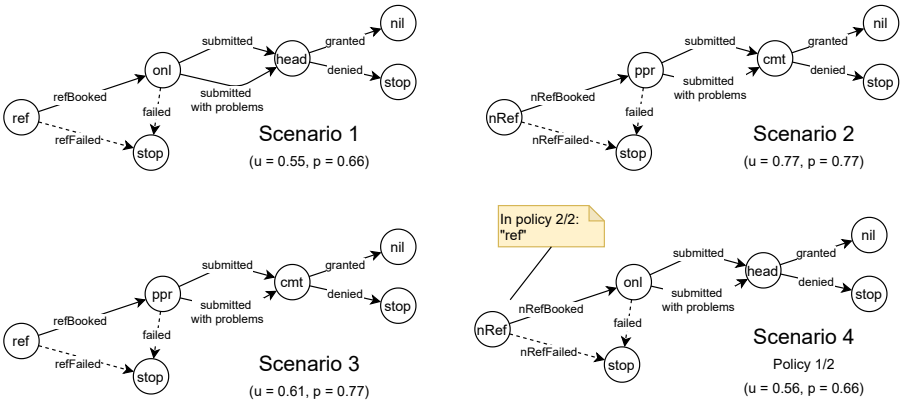


Fig. 5 Simplified DT-Golog policies for Scenarios (1)-(4). Paths that lead to `stop` states have been abbreviated for simplicity.

to book the ticket, having the application lost, and having the authorization request denied.

If her goal was exclusively minimizing costs, we would update the priority table so that *Reduce Costs* has weight 1.0 making all other weights 0.0 (Scenario 2). In that case the output is the policy on the top right of Figure 5: the actor would buy a non-refundable ticket and then apply through paper form and have the authorization granted by a committee. The probability of success is now higher at 0.77, thanks to seeking authorization from a committee using a paper form. This explains the reasoner's confidence to recommend a non-refundable ticket. The expected utility is the same number, 0.77, as the value of the specific outcome – if we do not normalize for its probability of occurrence – is 1.0. However, the actor may still want to minimize the loss from having the reimbursement request denied. By reducing *Reduce Costs*'s weight to 0.7 and adding weight 0.3 to *Avoid Loss Possibility* (Scenario 3), the optimal becomes the one seen on the bottom left of Figure 5, which is equally likely to succeed but with lower expected utility, due to the risk-averse choice. In other words, in repeated executions of the strategy, the actor would, on average, pay more than what they would if they remained to the non-refundable strategy. However, the money loss event will always be the minimum possible.

As a final scenario (Scenario 4), assume that the actor is not interested in *Reduce Cost* (so, now, weight is set to 0.0) but is interested in *Privacy* (0.6), and *Efficiency* (0.4). There are now two optimal policies one of which can be viewed at the bottom right of Figure 5 and the other one – not depicted here for simplicity – is identical except that instead of `rRef`, `ref` is considered. In order to safeguard her privacy and efficiency, the actor appears to need to follow a policy of a slightly lower success probability. Note that choice of `rRef` or `ref` is irrelevant to expected utility in that *'Reduce Cost'* does not weigh in the priority specification.

Interestingly, DT-Golog can be tweaked in terms of the criteria it uses to base its optimality calculations [49]. Specifically, by default and in accordance with MDP theory, policies are calculated on the basis of maximization of cumulative expected utility. However, we can change this objective function to e.g. be more reliant on the probability of success. For example, if we base the decisions exclusively on maximizing success probability, for Scenario 1 priorities, DT-Golog returns policies that involve actions `[nRef,ppr,cmt] (U = 0.37, p = 0.77)` and `[ref,ppr,cmt] (U = 0.46, p = 0.77)`, which have poorer utility but maximize probability. Note that here and in subsequent sections we use the notation $[\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3, \ldots]$ to refer to a more complex policy using a branch that contains task-attaining tasks – omitting the detailed policy for simplicity.

Through the exploration process we described in this section, analysts are able to investigate solutions that best match elicited priorities and qualities of interest. Probability is accounted both in the calculation of the optimal, as possible total utility values are factored by the probability that these values occur, and in assessing how likely the optimal policy is to succeed. In what follows, we describe how the extended goal model is translated to DT-Golog to allow for such reasoning.

5 Translating to DT-Golog

5.1 DT-Golog Basics

Overview. DT-Golog [9, 49] is a decision-theoretic extension of the high-level agent programming language Golog [29], which is, in turn, based on the situation calculus [47], a language for modeling and reasoning about dynamic domains. More specifically, DT-Golog incorporates Markov Decision Processes (MDPs) [50] in Golog's reasoning infrastructure, enabling the integration of programming and decision-theoretic planning. On one hand, DT-Golog allows the specification of a high-level program that cuts down the search space by prescribing a partial policy: the agent can only adopt policies that are consistent with the execution of the program. On the other hand, decision-theoretic planning in DT-Golog allows the programmer to specify uncertain worlds and probabilistic actions subject to the optimization of expected utility.

In the following, we briefly describe the aspects of DT-Golog that are essential for understanding the subsequent translation procedures, referring the reader to the respective literature for more details [49]. The following are a summary of the concepts we introduce below:

- Elementary Concepts: *fluents*, *actions* and *situations*.
- Axioms: *action precondition axioms* and *successor-state axioms*.
- Program Constructs: including *procedures* and *sequences*.
- Decision Theoretic Features: *stochastic actions*, *nature actions*, *reward predicates* and *probability predicates*.
- The outputs of DT-Golog: *policies*.

Elementary Concepts. The core of DT-Golog consists of constructs prescribed by the situation calculus: *fluents*, *actions* and *situations*. Fluents, playing the role of state features and represented through n-ary predicates with a situation term as their last argument, are understood as properties whose value can vary from situation to situation due to the performance of actions. For example, fluent *ticketBooked(traveler, location, s)* holds in situation s as a result of an action of submitting a booking order. Actions are first-order terms signifying specific activity performed by agents, e.g. *bookTicket(traveler, location)*. A situation is also a first-order term that denotes a sequence of actions, those that have been performed in the history of this situation. In particular the function symbol $do(a, s)$ denotes the situation which results from performing action a in situation s. A special constant S_0 denotes the initial situation, one where no action has been performed. Finally, there is a special predicate $Poss(a, s)$ used to state that action a is executable in situation s.

Axioms. A set of axioms $\mathcal{D}$ over the above basic constructs are then specified in order to describe the domain. From these, the most important are *action precondition axioms* that tell us when actions are possible and *successor-state axioms* that describe how fluent values change due to the performance of actions. The former are defined for each action α, and are of the form

$$\forall s.\ Poss(\alpha, s) \leftrightarrow \Pi_\alpha(s) \qquad \text{(Action Precondition)}$$

signifying that performance of the action α in some situation s is possible if and only if some formula Π_α is true in situation s. Successor-state axioms, on the other hand, are defined for each fluent and are of the form

$$\forall a, s.\ f(\vec{x}, do(a, s)) \leftrightarrow \Phi_f(\vec{x}, a, s) \qquad \text{(Successor-State Axiom)}$$

where f is an n-ary fluent symbol, $\vec{x}$ represents its n arguments, and Φ_f is a formula that intuitively says that the fluent f will be true after the performance of action a, if and only if, either a is an action that enables f and the conditions under which a brings about f hold in situation s, or f was already true in s and a did not turn it false.

Program Constructs. Having the background action theory $\mathcal{D}$ defined using the above set of axioms, languages in the Golog family further allow the development of high-level programs that describe behavior to be followed by agents while abiding by the action theory. Constructs found in most procedural languages are used for constructing Golog programs. Relevant to our purposes are *primitive actions*, defined as above, *procedures* δ, *sequences* (of actions or other procedures) denoted as $(\delta_1; \delta_2)$, *non-deterministic choices of actions* (denoted as $(\delta_1 \mid \delta_2)$), *tests/wait for conditions* (denoted by $\phi?$) and *if-then-else* conditionals.

In executing programs written using the above constructs, the Golog interpreter finds an execution of the specified high-level program relative to the action theory $\mathcal{D}$. The presence of non-deterministic choices within such programs, allows Golog to behave in part as an AI planner, searching for a

legal sequence of actions that amount to a legal execution of the high-level non-deterministic program such that the action theory is satisfied.

Decision Theoretic Features. The above features are found in all members of the Golog family. DT-Golog augments these with an additional component, an optimization theory, in which non-deterministic choices are made with respect to maximization of a decision-theoretic objective function – by default, the *cumulative expected utility*. To achieve this, DT-Golog extends the standard Golog features as follows.

Firstly, to the (deterministic) agent actions of core Golog, which are called *nature actions* in the context of DT-Golog, *stochastic actions* are added to denote exogenous events. Each stochastic action α is associated with a finite set of deterministic nature actions α^i. We use a set of nature actions to represent the actions that might have actually happened due to the influence of nature when α was attempted; hence the use of the term "nature". Successor-state axioms are provided for deterministic nature actions directly, but not for stochastic actions. The *probability predicates* $prob(\alpha^i, p_i, s)$ are used to assign probabilities p_i to each such nature action α^i in situation s. Note that the probabilities p_i can be simple numerical values or complex numerical functions of fluent values holding in the particular situation s. Further, *reward predicates* $reward(r, do(\alpha^i, s))$ assign a *reward* value to situations, actions or both.

Policies. In the face of non-deterministic choices, DT-Golog's reasoning engine searches for an optimal policy that maximizes the total accumulated expected utility defined as a sum of the products of the reward and the probability that this reward occurs when following a certain action trajectory. The *policy* π returned by the interpreter is a conditional Golog program[1] of (roughly) the form [49]:

$$
\begin{aligned}
a; \mathit{senseEffect}(a); \textbf{if } \phi_1 \textbf{ then } \pi_1 & \\
& \ldots \\
& \ldots \\
& \textbf{else if } \phi_{k-1} \textbf{ then } \pi_{k-1} \\
& \textbf{else } (\phi_k)?;\ \pi_k
\end{aligned}
$$

... where π_i are policies and ϕ's are outcomes of the $\mathit{senseEffect}(a)$ actions. The latter, sense-effect actions, are actions that the policy executing agent performs in order to identify the nature action a^i that was evoked by the stochastic action. The outcome of the sense-effect action is registered through a *sense-condition axiom* of the form $\mathit{senseCond}(a^i, \phi)$, that holds if ϕ is a logical condition that identifies the occurrence of nature's action a^i. This way, each branch of the policy is conditioned on a test formula that identifies the

[1] DT-Golog's definition of a policy is slightly different from the usual concept of a non-stationary Markov policy [50], which is a function mapping each state and a decision epoch to an action. In particular, DT-Golog policies prescribe an action only in those states that are reachable from the initial state (that corresponds to the initial situation S_0).

nature's outcome that was implemented. Accordingly, the choice is dictated by the nature and not by the agent.

To acquire a high-level view of how the DT-Golog policy is calculated and appreciate the meaning of cumulative expected utility maximization we mention here the Bellman optimality equation [49,50]:

$$V_n(s_i) = R(s_i) + \max_a\{\sum_{s_j} Pr(s_i, a, s_j)V_{n-1}(s_j)\}$$

In the above, $R(s)$ is the reward gained by reaching state s, $Pr(s_i, a, s_j)$ is the probability of transitioning to state s_j after executing action a at state s_i while $V_n(s)$ is the calculated cumulative expected utility of the process after n steps, where $V_0(s) = R(s)$. This basic MDP formulation shows how the value of a given state depends on the respective value of subsequent states, calculated recursively up to a horizon, multiplied by the probability of reaching each such subsequent state, while also making action choices that maximize such value.

DT-Golog combines such MDP optimization approach with Golog program execution, where Golog situations play the role of states, the MDP reward function is represented through the predicate $reward(r, s)$ and the transition probabilities are captured through the nature action probability clauses $prob(\alpha^i, p_i, s)$. The details and nuances of this synergy are not of direct relevance here and we refer the reader to the DT-Golog sources for a detailed presentation of DT-Golog semantics and the exact relationship to the standard MDP formulation [49].

Note that, for our purposes, the *off-line* version of the DT-Golog interpreter is used, in which the optimal policy is identified prior to execution. However, on-line versions of DT-Golog exist [15,49] in which sensing or other exogenous actions or events can be interleaved with action execution and trigger policy recalculation.

We now turn our focus to the translation of the probabilistic goal models into a DT-Golog specification – readers interested in more details on the Situation Calculus, Golog, and DT-Golog are referred to the corresponding literature [29,47,48,49].

5.2 From Goal Models to DT-Golog

To allow reasoning about goal alternatives in light of probabilistic effects we translate it into a DT-Golog specification. The translation is such that it can be automated allowing analysts to perform the subsequent reasoning activities without having any knowledge of the DT-Golog formalisms. For illustration purpose, we sketch how the translation is possible based on the example of Figure 6 translated into DT-Golog specifications as described in Figures 7 and 8. The general translation procedures can be found in Algorithms 1, 2, 3 and 4. The DT-Golog translation of the running example can be found in the accompanying technical report [32].

Algorithm 1: Translating Tasks and Effects

Let $\mathcal{G}$ be the set of hard-goals, $\mathcal{T}$ the set of leaf-level tasks, $\mathcal{E}_t$ the set of probabilistic effects of task t and Q_t the effect predicates of a task t.

1 $\mathcal{ST} \leftarrow \emptyset$; /* initialize the set of stochastic actions */
2 $\mathcal{F} \leftarrow \emptyset$; /* initialize the set of satisfaction fluents */
3 $\mathcal{N} \leftarrow \emptyset$; /* initialize the set of nature actions */
4 $\mathcal{SC} \leftarrow \emptyset$; /* initialize the set of sense conditions */
/* add a stochastic action a_t for each task in $\mathcal{T}$; add a fluent ϕ_{e_t} and a nature action a_t^i for each effect e_t of t; set up t's attainment formula */
5 **for** $t \in \mathcal{T}$ **do**
6 $\quad \mathcal{ST} \leftarrow \mathcal{ST} \cup \{a_t\}$;
$\quad$ /* for each domain predicate */
7 $\quad$ **for** $q_t \in Q_t$ **do**
8 $\quad\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{\phi_{q_t}\}$; /* add satisfaction fluent ϕ_{q_t} */
9 $\quad\quad \mathcal{N}_{\phi_{q_t}} \leftarrow \emptyset$; /* a set of actions that affect it */
10 $\quad$ **end**
11 $\quad$ **for** $e_t \in \mathcal{E}_t$ **do**
12 $\quad\quad \mathcal{N} \leftarrow \mathcal{N} \cup \{a_t^i\}$; /* add nature action a_t^i for e_t */
13 $\quad\quad \mathcal{SC} \leftarrow \mathcal{SC} \cup \{senseCond(a_t^i, sc_{a_t^i})\}$; /* $sc_{a_t^i}$ signifies performance of a_t^i in the resulting policy */
14 $\quad\quad$ **foreach** $q_t \in \mathcal{Q}_t$ *mentioned in* e_t **do**
15 $\quad\quad\quad \mathcal{N}_{\phi_{q_t}} \leftarrow \mathcal{N}_{\phi_{q_t}} \cup \{a_t^i\}$;/* keep track of actions a_t^i affecting ϕ_{q_t} */
16 $\quad\quad$ **end**
17 $\quad$ **end**
$\quad$ /* attainment formula below constructed based on effect table; constituent atoms are replaced with corresponding satisfaction fluents and may involve other elements from D */
18 $\quad \phi_t^{att} \leftarrow$ *attainment formula for* t;
19 **end**
20 $\mathcal{ST} \leftarrow \mathcal{ST} \cup \{a_f\}$; /* add the final stochastic action */
21 $\mathcal{N} \leftarrow \mathcal{N} \cup \{a_{s\text{-}e_f}, a_{f\text{-}e_f}\}$; /* add the nature actions for a_f */
22
/* set up hard-goal g's attainment formula */
23 **for** $g \in \mathcal{G}$ **do**
$\quad$ /* attainment formula below reflects the AND/OR structure of the decomposition and is grounded on attainment formulae of leaf level tasks; construction through recursion omitted here for simplicity */
24 $\quad \phi_g^{att} \leftarrow$ *attainment formula for* g;
25 **end**
26

Translating the elementary constructs. The translation of the elementary goal modeling constructs into DT-Golog ones is performed as follows (Lines 1-21 of Alg. 1). First, each leaf level task t is translated into a stochastic agent action a_t (Line 6). An additional stochastic agent action a_f is also produced (Line 20), playing the role of the *final action*. The final action is introduced to allow consideration of rewards only at the final stage of the action sequence, allowing for more accurate calculation of the expected utility – this will become clearer below. For each such stochastic action, we also introduce

a set of nature actions a_t^i each distinctly representing each of the possible effects $e_t \in \mathcal{E}_t$ due to the execution of a_t (Line 12). For each such action we also include a sense-condition clause, that associates the nature action with a fluent (Line 13). As we saw, these sense conditions are used for describing the resulting policy to a run-time policy execution environment.

Each task t is also associated with as many *satisfaction fluents* ϕ_{q_t} as the effect predicates (Line 8), each of which is true if the nature action that was actually performed made it so – we see below how we represent this through a successor-state axiom. For each task t, an attainment formula ϕ_t is also defined to denote what combinations of domain predicates make the task satisfied; this is simply a translation of the task's attainment formula grounded on the corresponding fluents (Line 18). Similarly, a DT-Golog attainment formula ϕ_g is introduced to represent the goal-level attainment formula of higher-level hard-goal g, in turn grounded on satisfaction formulae ϕ_t of leaf-level tasks (Lines 22-26).

Thus in Table 1 of Figure 7, task t_2 of Figure 6 has been translated into four nature actions denoting the four different effects of t_2 constructed, as per the table, through combinations of the predicates $Q_{t_2} = \{s\text{-}e_2, s\text{-}e_2', f\text{-}e_2, f\text{-}e_2'\}$. Actions $a_{s\text{-}e_2, s\text{-}e_2'}$ and $a_{s\text{-}e_2, f\text{-}e_2'}$ are examples of how such nature actions are denoted based on the effects they bring about. The final action a_f is associated with two nature actions $a_{s\text{-}e_f}$ and $a_{f\text{-}e_f}$ (Line 21). Note that in these examples the s- and f- prefixes denote success and failure of that action, respectively, but meanings alternative to success and failure can be utilized depending on the specific problem. Further, Table 2 of Figure 7 shows how the attainment formulae from Figure 6 are represented using simple DT-Golog axioms.

Precondition Axioms. For each nature action, we specify an *action-precondition axiom*. In particular, if there is an incoming $\xrightarrow{\text{pre}}$ link to a task node t from a goal or task h, then the attainment formula of h is added as a conjunct to the preconditions of all the associated nature actions a_t^i (Line 6 of Algorithm 2). Moreover, if there is an incoming $\xrightarrow{\text{npr}}$ link to the task node t, then the negation of the attainment formula for the source node of this link is added as a conjunct to the preconditions of a_t^i (Line 10). In the absence of any such links, these actions are specified to be always executable. The resulting formulae are assigned to the special predicate $Poss(a, s)$, which, as we saw, denotes that action a is executable in situation s (Line 12).

In Table 3 of Figure 7, we specify the preconditions of the tasks of Figure 6. We see below how we treat precedence links targeting higher-level hard goals.

Successor-State Axioms. For each satisfaction fluent ϕ_{q_t} associated with effect predicates of task t, we need a *successor-state axiom* that succinctly encodes both direct effects and non-effects and specifies exactly when the fluent changes. Such axioms are generated according to Lines 16-24 of Alg. 2. In short, a satisfaction fluent will retain its truth value unless one of the associated nature actions is performed, which will necessarily make it true. The nature actions a_t^i associated with the fluent are known by examining if q is a domain predicate that is part of the effect associated with a_t^i – see again Lines 14-16 of Alg. 1.

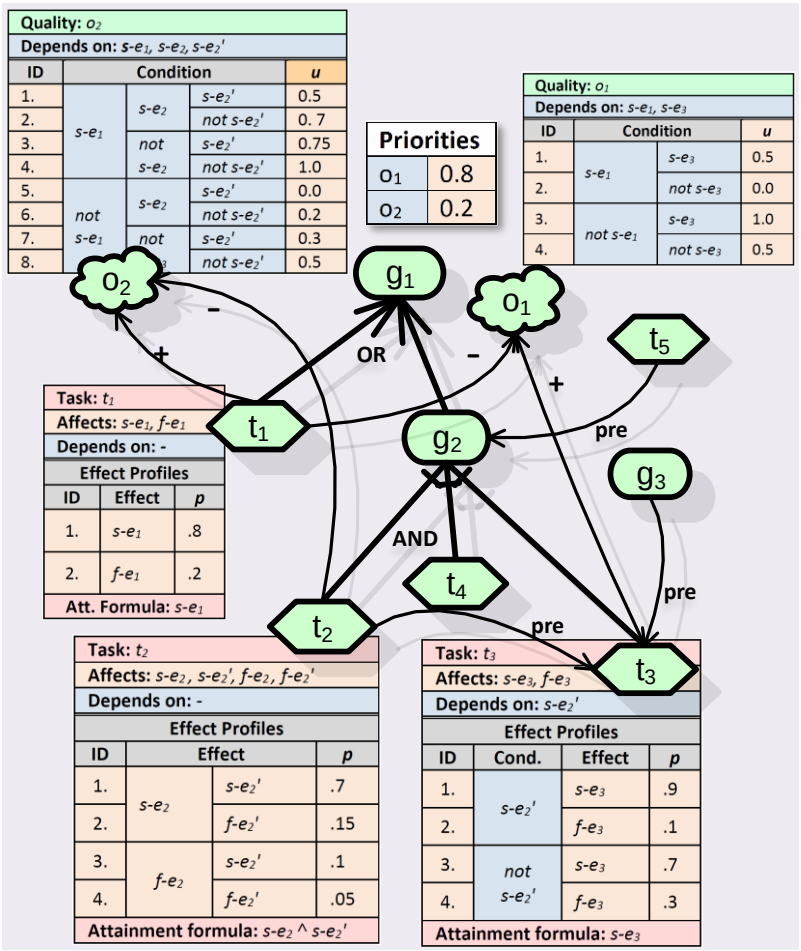


Fig. 6 Translation by Example

An example is shown in Table 4 of Figure 7. Thus, for t_2 there are four axioms, each describing how each satisfaction fluent constructed from Q_{t_2} becomes or remains true.

Procedures. For each goal g, we also introduce a DT-Golog procedure $proc_g$, which comprises of a test action $\phi?$ that waits for the preconditions of the procedure to hold, followed by some program δ. The precondition requires that conjunction of all incoming $\overset{\text{pre}}{\rightarrow}$ links must be satisfied and the disjunction of all incoming $\overset{\text{npr}}{\rightarrow}$ links must not be satisfied. If g is AND-decomposed, δ consists of the interleaving of its subtasks and subgoal procedures. On the other hand, if g is OR-decomposed, the program δ consists of the non-deterministic choice between its various subgoals and subtasks. The generation of procedures is described in Algorithm 3.

In Table 5 of Figure 7, the translation of the AND/OR structure of Figure 6 through applying these ideas can be seen as an example. Note that $(a\|b\|c)$ de-

Table 1: Actions and Fluents

Task	Stochastic Action	Nature Action	Fluent
$\mathbf{t}_1$	a_1	$a_{s\text{-}e_1}, a_{f\text{-}e_1}$	$\phi_{s\text{-}e_1}, \phi_{f\text{-}e_1}$
$\mathbf{t}_2$	a_2	$a_{s\text{-}e_2,s\text{-}e'_2}, a_{s\text{-}e_2,f\text{-}e'_2}$ $a_{f\text{-}e_2,s\text{-}e'_2}, a_{f\text{-}e_2,f\text{-}e'_2}$	$\phi_{s\text{-}e_2}, \phi_{f\text{-}e_2}$ $\phi_{s\text{-}e'_2}, \phi_{f\text{-}e'_2}$
$\mathbf{t}_3$	a_3	$a_{s\text{-}e_3}, a_{f\text{-}e_3}$	$\phi_{s\text{-}e_3}, \phi_{f\text{-}e_3}$

Table 2: Attainment Formulae:

Task/Goal	Axiom
$\mathbf{t}_1$	$\phi_{t_1} \leftrightarrow \phi_{s\text{-}e_1}$
$\mathbf{t}_2$	$\phi_{t_2} \leftrightarrow \phi_{s\text{-}e_2} \wedge \phi_{s\text{-}e'_2}$
$\mathbf{t}_3$	$\phi_{t_3} \leftrightarrow \phi_{s\text{-}e_3}$
$\mathbf{g}_1$	$\phi_{g_1} \leftrightarrow \phi_{t_1} \vee \phi_{g_2}$
$\mathbf{g}_2$	$\phi_{g_2} \leftrightarrow \phi_{t_2} \wedge \phi_{t_3} \wedge \phi_{t_4}$

Table 3: Action Precondition Axioms:

Task	Precondition Axioms
$\mathbf{t}_1$	$Poss(a_{s\text{-}e_1}, s) \leftrightarrow true$ $Poss(a_{f\text{-}e_1}, s) \leftrightarrow true$
$\mathbf{t}_2$	$Poss(a_{s\text{-}e_2,s\text{-}e'_2}, s) \leftrightarrow true$ $Poss(a_{s\text{-}e_2,f\text{-}e'_2}, s) \leftrightarrow true$ $Poss(a_{f\text{-}e_2,s\text{-}e'_2}, s) \leftrightarrow true$ $Poss(a_{f\text{-}e_2,f\text{-}e'_2}, s) \leftrightarrow true$
$\mathbf{t}_3$	$Poss(a_{s\text{-}e_3}, s) \leftrightarrow \phi_{t_2}(s) \wedge \phi_{g_3}(s)$ $Poss(a_{f\text{-}e_3}, s) \leftrightarrow \phi_{t_2}(s) \wedge \phi_{g_3}(s)$
-	$Poss(a_{s\text{-}e_f}, s) \leftrightarrow \phi_{g_1}$
-	$Poss(a_{f\text{-}e_f}, s) \leftrightarrow \phi_{g_1}$

Table 4: Successor State Axioms:

Task	Successor State Axioms
$\mathbf{t}_1$	$\phi_{s\text{-}e_1}(do(a,s)) \leftrightarrow \phi_{s\text{-}e_1}(s) \vee a = a_{s\text{-}e_1}$ $\phi_{f\text{-}e_1}(do(a,s)) \leftrightarrow \phi_{f\text{-}e_1}(s) \vee a = a_{f\text{-}e_1}$
$\mathbf{t}_2$	$\phi_{s\text{-}e_2}(do(a,s)) \leftrightarrow \phi_{s\text{-}e_2}(s) \vee a = a_{s\text{-}e_2,s\text{-}e'_2} \vee a = a_{s\text{-}e_2,f\text{-}e'_2}$ $\phi_{f\text{-}e_2}(do(a,s)) \leftrightarrow \phi_{f\text{-}e_2}(s) \vee a = a_{f\text{-}e_2,s\text{-}e'_2} \vee a = a_{f\text{-}e_2,f\text{-}e'_2}$ $\phi_{s\text{-}e'_2}(do(a,s)) \leftrightarrow \phi_{s\text{-}e'_2}(s) \vee a = a_{s\text{-}e_2,s\text{-}e'_2} \vee a = a_{f\text{-}e_2,s\text{-}e'_2}$ $\phi_{f\text{-}e'_2}(do(a,s)) \leftrightarrow \phi_{f\text{-}e'_2}(s) \vee a = a_{s\text{-}e_2,f\text{-}e'_2} \vee a = a_{f\text{-}e_2,f\text{-}e'_2}$
$\mathbf{t}_3$	$\phi_{s\text{-}e_3}(do(a,s)) \leftrightarrow \phi_{s\text{-}e_3}(s) \vee a = a_{s\text{-}e_3}$ $\phi_{f\text{-}e_3}(do(a,s)) \leftrightarrow \phi_{f\text{-}e_3}(s) \vee a = a_{f\text{-}e_3}$
$\_$	$\phi_{s\text{-}e_f}(do(a,s)) \leftrightarrow \phi_{s\text{-}e_f(s)} \vee a = a_{s\text{-}e_f}$ $\phi_{f\text{-}e_f}(do(a,s)) \leftrightarrow \phi_{f\text{-}e_f(s)} \vee a = a_{f\text{-}e_f}$

Table 5: Procedures:

Goal	Procedures
$\mathbf{g}_1$	$proc_{g_1} \stackrel{\text{def}}{=} [(a_1 \vert proc_{g_2}); a_f]$
$\mathbf{g}_2$	$proc_{g_2} \stackrel{\text{def}}{=} [\neg\phi_{t_5}?; (a_2 \vert\vert a_3 \vert\vert a_4)]$

Fig. 7 Examples of DT-Golog Specifications for Figure 6

notes the non-deterministic choice between all possible interleaving of actions a, b, and c – though some may not be feasible due to lower-level precedence constraints. Note also that the top-level procedure is augmented with the final action a_f at the end, in a way that no policy is successful without concluding with that action.

Once the core Golog aspect has been developed as above, the decision-theoretic component is added by defining the probability distributions and the reward functions as follows.

Algorithm 2: Generating Axioms

Let $\mathcal{G}$ be the set of hard-goals, $\mathcal{T}$ the set of leaf-level tasks, $\mathcal{E}_t$ the set of probabilistic effects of task t and Q_t the effect predicates of a task t.

```
   /* set up precondition axioms for nature actions                    */
 1 APA ← ∅;      /* initialize the set of action precondition axioms   */
 2 for n ∈ N do
 3 |   ψ_n ← true;
 4 |   Let InPre be the set of nodes from which there is an incoming -pre-> to t;
 5 |   for i ∈ InPre do
 6 |   |   ψ_n ← ψ_n ∧ φ_i^att;        /* φ_i^att is attainment formula for i */
 7 |   end
 8 |   Let InNpr be the set of nodes from which there is an incoming -npr-> to t;
 9 |   for i ∈ InNpr do
10 |   |   ψ_n ← ψ_n ∧ ¬φ_i^att;       /* φ_i^att is attainment formula for i */
11 |   end
12 |   APA ← APA ∪ {Poss(n, s) ↔ ψ_n};
13 end
   /* φ_root below is the attainment formula of the root goal          */
14 APA ← APA ∪ {Poss(a_{s-e_f}, s) ↔ φ_root, Poss(a_{f-e_f}, s) ↔ φ_root};
15
    /* set up successor-state axioms for satisfaction fluents          */
16 SSA ← ∅;       /* initialize the set of successor-state axioms      */
17 for φ_e ∈ F do
18 |   ψ_e ← φ_e(do(a, s)) ↔ φ_e(s);
19 |   Let N_{φ_e} be the set of nature actions that make φ_e true;   /* a task is
   |    associated with various nature actions as described above     */
20 |   for n ∈ N_{φ_e} do
21 |   |   ψ_e ← ψ_e ∨ a = n;
22 |   end
23 |   SSA ← SSA ∪ {ψ_e};
24 end
```

Probabilities. Recall that effects are used for the construction of effect profiles, which are triples of the form $(e_t^i, \phi_{c_t}^i, p_i)$ representing the probability of effect e_t occurring with probability p_i, if $\phi_{c_t}^i$ is true, once task t is performed. For each such effect profile we introduce predicates of the form $prob(a_t^i, p_i, s)$, where a_t^i is the nature action associated with effect e_t^i (see Line 12 of Algorithm 1) whose probability we define to be p_i for each situation s (Lines 1-4 of Algorithm 4). The formula is written in the form $prob(a_t^i, p_i, s)$ *if* $\phi_{c_t}^i(s)$, i.e. the p_i probability is assigned in situation s if $\phi_{c_t}^i$ is true in situation s.

Finally we assign the final action to be certainly successful. Table 6 of Figure 8 shows the probability definitions for nature actions related to tasks t_1, t_2 and t_3 of Figure 6.

Rewards. The reward function is calculated in a very similar way, with the difference that, since a reward is a unique value that characterizes an entire solution, values from individual reward tables are merged together based on the given quality goal preference profile. Recall that a priority specification describes the relative importance of each of the top-level goals. At the same time, for each truth assignment for the effects, the satisfaction function of each

Algorithm 3: Translating the Goal Structure

Let $\mathcal{G}$ be the set of hard-goals

```
   /* set up procedures for each goal g ∈ G                                  */
 1 for g ∈ G do
 2 |
   |  /* first set up the preconditions of g                                 */
 3 |  Pre_g ← true;          /* initialize the preconditions of g           */
 4 |  Let InPre be the set of nodes from which there is an incoming -pre-> to g;
 5 |  for i ∈ InPre do
 6 |  |  Pre_g ← Pre_g ∧ φ_i^att;      /* φ_i^att is attainment formula for i */
 7 |  end
 8 |  Let InNpr be the set of nodes from which there is an incoming -npr-> to g;
 9 |  for i ∈ InNpr do
10 |  |  Pre_g ← Pre_g ∧ ¬φ_i^att;     /* φ_i^att is attainment formula for i */
11 |  end
12 |
   |  /* now deal with the structure for g                                   */
13 |  Let c be the first child of g;
14 |  Let C_g be the rest of the children of g;
15 |  δ_g ← c;
16 |  if g is AND-decomposed then
17 |  |  for c' ∈ C_g do
18 |  |  |  δ_g ← δ_g || c';
19 |  |  end
20 |  else
21 |  |  for c' ∈ C_g do
22 |  |  |  δ_g ← δ_g | c';
23 |  |  end
24 |  end
25 |
26 |  δ_g ← Pre_g?; δ_g;          /* add preconditions of g to δ_g          */
27 |
28 |  if g is the top level goal then
29 |  |  δ_g ← δ_g; a_f;
30 |  end
31 end
```

quality has a specific value. Gathering all those values, multiplying them by the weight of their corresponding quality in the preference specification, and adding them up gives us the overall reward value for the situation. Lines 6-15 of Algorithm 4 describe the logic of the translation.

In the reward formulation, the rationale for the inclusion of the final action a_f finally becomes apparent. Specifically, we want DT-Golog to assign values only to complete policies, i.e. policies in which a_f has been included, which allows for accurate calculation of expected utility. Omission of a_f will add to the final utility value the values of intermediate states, which may be counter-intuitive in most applications of our framework, in which successful performance of a task brings about a reward only once and for the entire policy. This is indeed the case in our running example. However, for problems in which effects can be undone by subsequent tasks, it may be more pertinent

Algorithm 4: Translating Effect Tables/Groups and Utility Tables

Let $\mathcal{P}_t$ be the set of effects profiles of task t
Let $\mathcal{O}$ be the set of quality goals o_i, each with global priority w_i
Let $\mathcal{U}_o$ be the set of utility profiles of quality goal o

$\mathcal{RA} \leftarrow \emptyset$; /* initialize the set of probability clauses */
for $p \in \mathcal{P}_t$ **do**
  /* p is of the form $(e_t, \phi^i_{c_t}, p_i)$ where each e_t maps 1-1 with nature action a^i_t */
  $\mathcal{RA} \leftarrow \mathcal{RA} \cup \{prob(a^i_t, p_i, s) \leftrightarrow \phi^i_{c_t}(s)\}$;
end

for $o \in \mathcal{O}$ **do**
  $\psi_o(r, s) \leftarrow false$;
  $\psi_{\bar{o}}(s) \leftarrow true$;
  /* each v below is of the form $(q, \phi^i_{c_o}, u_i)$, where q the quality, u_i a utility value, $\phi^i_{c_o}$ a condition under which u_i is obtained */
  for $v \in \mathcal{U}_o$ **do**
    /* if $\phi^i_{c_o}(s)$ then unify r with the corresponding u_i */
    $\psi_o(r, s) \leftarrow \psi_o(r, s) \vee (\phi^i_{c_o}(s) \wedge (r = u_i))$;
    /* collect also the negations of the conditions */
    $\psi_{\bar{o}}(s) \leftarrow \psi_{\bar{o}}(s) \wedge \neg\phi^i_{c_o}(s)$;
  end
  /* if none of the conditions apply unify r to zero (0) */
  $\psi_{\hat{o}}(r, s) \leftarrow \psi_o(r, s) \vee (\psi_{\bar{o}}(s) \wedge (r = 0))$;
end
/* unify all the r's from each quality, then weight-average them based on the corresponding priorities; note the role of $\phi_{s\text{-}e_f}(s)$ */
$reward(r_T, s) \leftrightarrow \phi_{s\text{-}e_f}(s) \wedge \psi_{\hat{o}1}(r_1, s) \wedge \psi_{\hat{o}2}(r_2, s) \wedge \ldots \wedge (r_T = w_1 \cdot r_1 + w_2 \cdot r_2 + \ldots) \vee \neg\phi_{s\text{-}e_f}(s) \wedge (r_T = 0)$

to keep track of the values of the intermediate states by simply omitting any mention of the final action a_f in the translation.

For the example of Figure 6 the translation can be seen in Table 7 of Figure 8; noting that, for comprehensibility, disjunctions are written in separate clauses as in a Prolog-style program. In a situation in which $s\text{-}e_1, s\text{-}e'_2, s\text{-}e_3$ are true and $s\text{-}e_2$ is false, by looking at the tables, o_1 is satisfied by 0.5 and o_2 by 0.75, and given their relative importance 0.8 and 0.2, the total reward is $0.5 \cdot 0.8 + 0.2 \cdot 0.75 = 0.55$.

Policies. As we saw, given the procedures and domain theory, DT-Golog's reasoner will return a policy that includes: stochastic actions a, *senseEffect*(a) actions to identify the nature action a^i that was evoked by the stochastic action, test conditions (ϕ)? and *if-then-else* conditionals that lead to different choices of subsequent actions based on the nature actions that result from a. The abstract policies introduced in Section 4 are simplified representations of the same policy construct with reference to tasks and effects that correspond to the mentioned action and fluent. Specifically, to produce the high-level representations, we: (a) remove the *senseEffect*(a) actions, (b) abbreviate branches that inevitably lead to failure/`stop`, and (c) remove references to a_f.

Table 6: Probabilities:

Task	Clause
$\mathbf{t}_1$	$prob(a_{s\text{-}e_1}, 0.8, s)$ $prob(a_{f\text{-}e_1}, 0.2, s)$
$\mathbf{t}_2$	$prob(a_{s\text{-}e_2, s\text{-}e'_2}, 0.7, s)$ $prob(a_{s\text{-}e_2, f\text{-}e'_2}, 0.15, s)$ ...
$\mathbf{t}_3$	$prob(a_{s\text{-}e_3}, 0.9, s)$ if $\phi_{s\text{-}e'_2}(s)$ $prob(a_{f\text{-}e_3}, 0.1, s)$ if $\phi_{s\text{-}e'_2}(s)$ $prob(a_{s\text{-}e_3}, 0.7, s)$ if $\neg\phi_{s\text{-}e'_2}(s)$ $prob(a_{f\text{-}e_3}, 0.3, s)$ if $\neg\phi_{s\text{-}e'_2}(s)$
...	...
-	$prob(a_{s\text{-}e_f}, 1.0, s)$ $prob(a_{f\text{-}e_f}, 0.0, s)$

Table 7: Rewards:

Reward Clause
$reward(0.5 \cdot 0.8 + 0.5 \cdot 0.2, s)$ if $\phi_{s\text{-}e_1}(s) \wedge \phi_{s\text{-}e_2}(s) \wedge \phi_{s\text{-}e'_2}(s) \wedge \phi_{s\text{-}e_3}(s) \wedge \phi_{e_f}(s)$
$reward(0.0 \cdot 0.8 + 0.5 \cdot 0.2, s)$ if $\phi_{s\text{-}e_1}(s) \wedge \phi_{s\text{-}e_2}(s) \wedge \phi_{s\text{-}e'_2}(s) \wedge \neg\phi_{s\text{-}e_3}(s) \wedge \phi_{s\text{-}e_f}(s)$
$reward(0.5 \cdot 0.8 + 0.7 \cdot 0.2, s)$ if $\phi_{s\text{-}e_1}(s) \wedge \phi_{s\text{-}e_2}(s) \wedge \neg\phi_{s\text{-}e'_2}(s) \wedge \phi_{s\text{-}e_3} \wedge \phi_{s\text{-}e_f}(s)$
$reward(0.0 \cdot 0.8 + 0.7 \cdot 0.2, s)$ if $\phi_{s\text{-}e_1}(s) \wedge \phi_{s\text{-}e_2}(s) \wedge \neg\phi_{s\text{-}e'_2}(s) \wedge \neg\phi_{s\text{-}e_3} \wedge \phi_{s\text{-}e_f}(s)$
...
$reward(0.0, s)$ otherwise.

Fig. 8 Examples of Probabilities and Rewards for Figure 6

6 Analysis and Evaluation

We now turn our focus to steps we have taken to evaluate and explore the capabilities and limitations of the proposed modeling and reasoning approach. We specifically report on (a) a simulation analysis, (b) a sensitivity analysis approach, and a (c) scalability analysis. The simulation code, the models used for performance evaluation described in this section, as well as DT-Golog installation and execution instructions, can be found in the accompanying technical report [32].

6.1 Simulation Analysis

As a first evaluative step for the meaningfulness of the translation procedure and the tool output, we develop a simulation of our running problem of Figure 1. The program simulates a large number of instances in which the main actor, the researcher, is confronted with the problem of quickly scheduling a trip and making the corresponding decisions. Monte-Carlo sampling of outcomes of stochastic actions is used in each run based on probabilities defined in the model. Likewise, the reward structure follows the one defined in the extended goal model of Figures 3 and 4. The simulation calculates success probabilities by counting the proportion of runs in which the policy led to the achievement

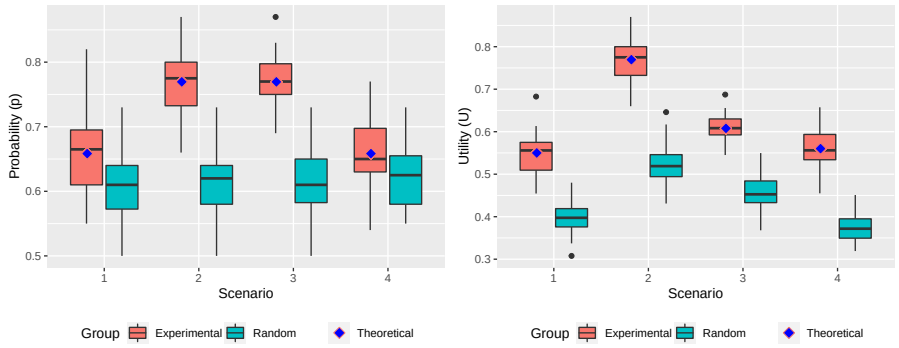


Fig. 9 Calculated (Experimental) vs. Random Policies

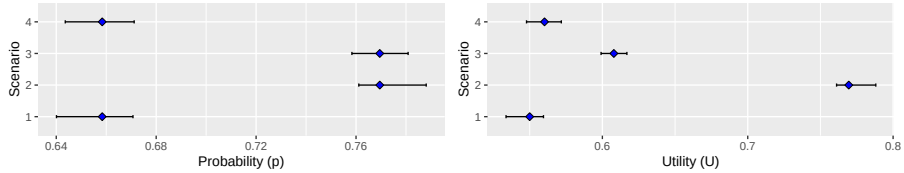


Fig. 10 Observed vs. Theoretical Probabilities and Utilities

of the root goal. The expected utility is, likewise, calculated by averaging the values acquired in each run, including the zero utilities acquired in policies that led to failure. This calculation is suitable when the model assumes reward upon the success of the final task in a policy, maintaining a zero reward up to that point, as is the case in our example model.

The two different scenarios for actor decisions include following the policy generated by DT-Golog under the same parameters, versus making a random choice every time. We expect that, if DT-Golog's proposed policy is optimal, then, following that policy offers, on average in the long term, a higher total value than simply making random actions.

For a total of 50 simulations, each including 100 runs, the results acquired can be seen in Figure 9. As expected, following the suggested policy offers a higher probability and utility compared to choosing randomly each time. Further, the probabilities and utilities that DT-Golog calculates fall at all times within the 95% confidence interval of the observed values, as can be seen in Figure 10.

6.2 Number Acquisition and Sensitivity Analysis

The presented extension and its use heavily rely on the identification of numbers of three kinds: probabilities, utility values, and preference weights.

Within the decision theory literature methods for eliciting probabilities and utility measures, as well as priorities have long been studied (e.g., [7,

25]). Probabilities can come in the form of measurements in the domain. In our running example, the majority of success rates of various tasks (e.g. success rate for authorization granting) can be assumed to be available in past data. Whenever this is not possible, probabilities can be subjective evaluations. Utility values, on the other hand, can be more challenging to identify [46]. The problem is common in most quantitative goal modeling frameworks, decision-theoretic or otherwise [3,19]. As has been shown [30], quantitative contributions can be the results of a sequence of simple AHP-style pairwise comparisons [26], when certain structural assumptions can be made about the models. In our diagram, this would imply treating each OR-decomposition as a separate decision problem and (a) performing pairwise comparisons between each OR-decomposition alternative with respect to each of the relevant quality goals or conditions, effectively constructing utility tables such as those of Figure 4(c-d), (b) performing pairwise comparisons between soft-goals with respect to the overall quality in meeting the root hard-goal, producing thereby the preference table of Figure 4(e).

Sensitivity analysis can also assist utility and preference value identification. To perform such analysis in our context we start from an initial set of numbers that yields a specific policy. Then, focusing on a specific utility or preference value, we identify the policies that DT-Golog would generate if the value were to be replaced by values in the entire [0.0,1.0] interval, leaving all other probability, utility, and preference values unchanged. To acquire those results, a linear traversal of the interval is performed, following a precision step (e.g. 0.05, i.e., 21 evaluations). The goal is to identify threshold values within the interval, above and below which different optimal policies are produced.

The results of this analysis for utility values taken from utility tables or utility links can be seen in the sensitivity chart of Figure 11. The figure shows an analysis based on Scenario 1 of Section 4, and numbers as in Figures 3 and 4. Each horizontal bar represents a different variable. For example, the top bar represents the utility value taken from the utility link from *'Head Authorizes'* to *'Privacy'*. The red vertical bar represents the initial value of the variable in question (0.8, in our case). The lightly shaded part of the bar represents the values of the contribution in which the policy is the same as the initial one, `[ref,onl,head]` in our case using the notation we introduced in Section 4 for abbreviating policies. Darker shades show values of the variable in question in which DT-Golog would produce a different policy, even if all else stayed the same. The policy to which it would switch is annotated in the corresponding area. Thus, to switch from `[ref,onl,head]` to `[ref,ppr,cmt]` the appreciation of how well it serves *'Privacy'* to have the head authorize the application (first bar from the top) must lower from 0.8 to 0.32, assuming all else stays equal. Alternatively, the perception of the contribution to privacy when having a committee to authorize the application must increase from 0.2 to 0.6 (second bar from the top) for DT-Golog to decide that `[ref,ppr,cmt]` is now a better policy, all else being equal. Notice in the diagram that certain contributions do not seem to have any effect in the determination of the optimal policy, as, regardless of their value, the optimal policy remains the same.

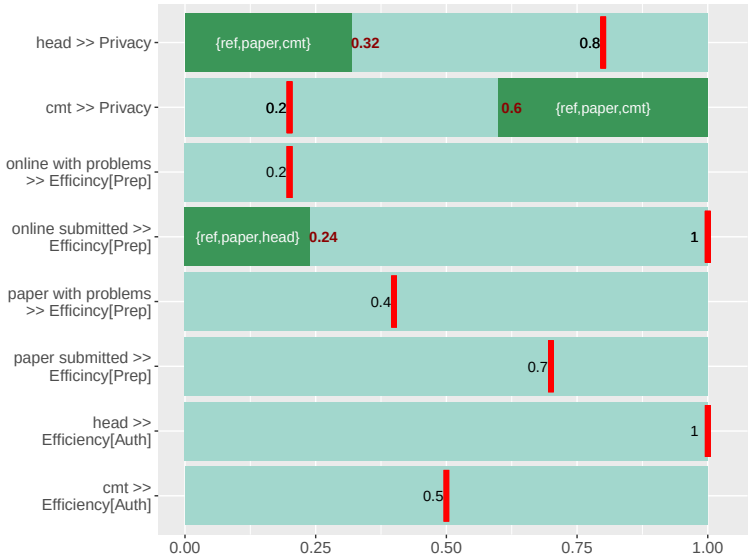


Fig. 11 Sensitivity Analysis for Individual Utility Values

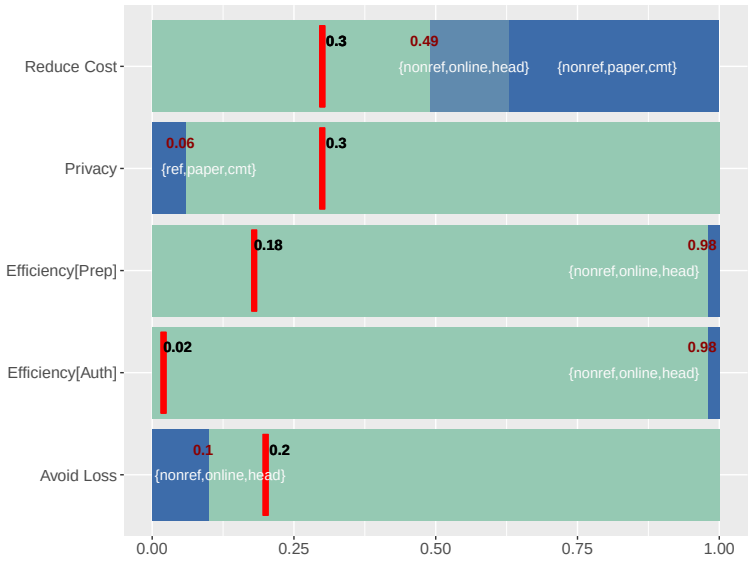


Fig. 12 Sensitivity Analysis for Preferences

To perform such analysis with preferences or contributions that are results of comparisons (as per [30]), we relax the holding-all-else-constant condition as follows. Considering a preference table such as that of Figure 4(e), to maintain a total weight equal to 1.0, when considering an increased (resp. decreased) weight for one component of the preference by an amount e the other components must share a decrease (resp. increase) of the same amount e. More specifically, when testing sensitivity with respect to the component i of the preference, by updating its initial value w_i into $w'_i = w_i + e$, $e \in [-w_i, 1 - w_i]$, then for all other components $j \neq i$ we adjust $w'_j = w_j - e\frac{w_j}{\sum_{k \neq i} w_k}$. Thus, the share of each of the other components to the overall amount of adjustment is based on its weight compared to the weight of the other affected components.

Following this approach, the sensitivity chart of the analysis for priority table of Figure 4(e) can be seen on Figure 12. For example, if we were to increase *'Reduce Cost'* from 0.3 to 0.49 the optimal policy switches from `[ref,onl,head]` to `[nRef,onl,head]`. The switch is due to the effect of both the increase of *'Reduce Cost'* and the decrease of *'Avoid Loss'* as per the above preference-wide adjustment. The same effect would be observed if we reduced the *'Avoid Loss'* weight from 0.2 to 0.1. Increasing *'Reduce Cost'* further to 0.63, discounting the other weights accordingly, the policy switches to `[nRef,ppr,cmt]`.

It may be of value to make identification of the most sensitive parameter the focal point of this exercise. In our case, it is *'Avoid Loss'*, for which the value in which we obtain a different policy (0.1) is the closest to its current one (0.2). In addition, looking at what changes within the policy offers information with regards to what the weight updates actually affect. In the last example, it is the choice between a refundable and non-refundable ticket that appears to be sensitive to parameter fluctuation.

6.3 Scalability

DT-Golog is able to offer us solutions for smaller problems like the one we discussed in previous sections in fractions of a second. However, MDP problems are known to be computationally intractable. DT-Golog, thanks to its constraining the search/calculation space using pertinent domain information, is expected to perform well in larger problems. To evaluate this proposition we developed a number of goal models of different sizes and structures and measured the amount of time it takes for DT-Golog to produce the optimal policy. A first set of models is random and is constructed manually from scratch. A second set of models was based on re-purposing real goal models from a variety of domains that we have developed in the past, including for a meeting scheduler, an online bookstore, an automatic teller machine as well as a geriatric clinic. Working manually we combined these models in various ways to produce larger models. That we used real goal models allows the resulting structures to preserve some naturalness. For all probabilities and utilities, automatically generated random numbers are used.

	Total Nodes	Goals	Tasks	Precedences	Time (sec)
1	20	9	11	4	0.08
2	30	13	17	7	0.00
3	40	17	23	8	0.02
4	50	21	29	9	1.36
5	55	23	32	10	95.20
6	60	25	35	11	354.10
7	65	27	38	12	8031.80
8	70	29	41	13	16819.80
9	80	33	47	16	*

Table 1 Performance Results for Randomly Generated Models ('*' means no result within 5h)

	Total Nodes	Goals	Tasks	Precedences	Time (sec)
1	35	20	15	10	0.57
2	40	19	21	12	1.60
3	45	21	24	15	10.73
4	55	25	30	17	285.24
5	65	29	36	21	*

Table 2 Performance Results for Combined Models ('*' means no result within 5h)

Experiments are run on an Intel ® Core™ i7-6700 CPU @ 3GHz x8 with 16GB of RAM running Linux 4.16.0 (Debian 10). The results for the random and combined realistic models can be seen in Tables 1 and 2, respectively. We observe that the "knee" in the running time emerges in model sizes between 50 and 60 goals. Up to about 40 goals, reasoning seems to be possible in a small number of seconds allowing usable exploratory and sensitivity analysis. We note that the presence of x number of tasks in the model entails an MDP of x actions and at least 2^{2x} states, assuming that each task has a success and a fail effect, each modeled by the corresponding binary variable. DT-Golog's ability to encode detailed domain information through a control program – in our case the procedure resulting from translating the decomposition tree – allows concise representation of the problem, ruling out transitions that do not satisfy precondition axioms, while also enabling quicker identifications of solutions.

For large models, a strategy to address increased computational time is through breaking the problem into sub-problems and efficiently solving each. This is possible when the sub-problems do not depend on each other in any way. For example, when the top goal is AND-decomposed into two or more sub-trees with top-level precondition dependencies between each other, the individual optimal policies can be used to construct a combined optimal one.

6.4 Validity Threats

We now discuss some of the validity threats that emerge from our evaluation effort, focusing on external and internal validity.

External validity is concerned with the generalizability of our analysis. In our case, this concerns the representativeness of the models we have applied the framework to and the tests we run. Aspects of interest include computational performance and the pragmatic quality of the language. We have addressed performance by firstly, randomly generating goal structures, and, secondly, generating structures by combining existing goal models, so that the resulting model preserves some of the naturalness of the original models. In case there is a class of models that, unbeknownst to the current analysis, possesses structural characteristics that are drastically detrimental to performance, the scalability tactics we described above may offer a first line of defense. Pragmatic quality of the language itself [42], i.e. the ability to develop models such as the grants adjudication described here or the meeting scheduling ones described in [33] that are useful for communication and decision making in real-world settings is a matter that requires extensive empirical work which we are hoping to conduct in the future.

Let us turn to internal validity, which is concerned with whether evaluation claims follow from our procedures and the results. The performed simulations are chosen as evaluative instruments in that they constitute a sanity check of the translation theory, the tool and its utilization, as well as the way we interpret its results. While the fact that both the DT-Golog specification and the simulation (written in R) are produced by the same team and on the basis of the same probabilistic and reward model can be construed as a validity threat, the observed consistency between calculated and simulated scenarios diminishes the chance of important issues with the translation or our interpretation of the results such as misunderstanding of DT-Golog's constructs or its definition of optimality. For example, if the optimal policies resulting from our decision-theoretic analysis did not turn out optimal in the simulation, this would raise questions with regards to how the DT-Golog specification is produced and how it is used. While this was not observed in the model we tried, to rule out coincidental positive results, more and preferably independently conducted simulation analyses could be conducted. In fact, given that simulations may also help identify problems with the models themselves, it may be a good practice to generally accompany DT-Golog analysis with an independently developed simulation. We are working on developing toolsets for supporting such activities. Finally, the development and correctness of the DT-Golog reasoner itself are discussed at length in the DT-Golog literature [49].

7 Related Work

The idea to view the performance of tasks and achievement of goals within goal models as stochastic events has been investigated in the literature. Letier and van Lamsweerde [28] use the goal model structure to construct probability functions for probabilistically measuring the achievement of non-functional objectives. A top-down approach for selecting solutions that optimize such prob-

abilistically constructed objectives was later introduced by Heaven and Letier [20]. These techniques are most suitable when there is a need for analyzing how various solutions of goal models affect the satisfaction probability of high-level system goals – we review more work with that goal below. Our work takes additional aspects into consideration for when the problem at hand requires more expressive modeling and analysis – and the resources are available to pursue such expressiveness. This includes dynamic aspects of agent actions (including effects and preconditions), combining probability with measures of utility and preference to allow for a decision-theoretic formulation of the optimality criterion and the generation of agent policies, i.e., ordered action sequences rather than simple alternative selection within the goal model.

A view of goal models as tools for multi-criteria decision representation is one that has attracted substantial interest as well. Ma and de Kinderen, for example [37], introduce a reference model and a process for performing MCDA (Multi-criteria Decision Making) using goal models; though not through a decision-theoretic perspective (e.g., considering probabilities and utilities). Elsewhere, Nguyen et al. [43] offer an extended goal modeling language for efficient reasoning with preferences among multiple objectives backed by efficient optimization modulo theory solvers. While the specific work also does not consider probabilities and rewards, it is possible that SMT/OMT solvers promoted in it can be adapted to perform some form of decision-theoretic reasoning. This would assume that objective functions and temporal constraints are formulated in a way that reflects, on the one hand, the probabilistic and reward components of the decision-theoretic aspect and, on the other hand, the components of the action theory aspect (preconditions, effects, domain facts, etc.). While such investigation may be warranted by SMT/OMT's reported computational performance, DT-Golog is readily suited to satisfy the modeling requirements adding also the important ability to refine specifications with procedures using common imperative constructs.

Earlier, we have utilized various forms of preference-enabled planning for both reasoning about requirements alternatives and for designing adaptive software [31,34,35]. In these efforts, priority is given to modeling the action-theoretic aspect and the efficient generation of agent plans that fulfill prioritized – through preference specifications – high-level objectives. However, they are generally not concerned with the quantitative representation of uncertainty, although they feature some notion of utility. Nevertheless, when modeling uncertainty is not a priority, these approaches offer the advantage of scalable reasoning through state-of-the-art planners such as HTNs [41]. A relatively different notion of uncertainty within goal models is proposed by Horkoff et al. [23]. Rather than modeling uncertainty in the domain, the authors propose a way to model uncertainty in the modeling process, by annotating modeling elements with tags such as "may" or "var" indicating uncertainty about respectively the existence or constancy of any element within a model.

The application of MDPs is common in the area of self-adaptive systems. Solano et al. [14] augment contextual goal models [1] with parametric symbolic formulae for reliability and cost to effectively allow for MDP style reasoning us-

ing PRISM [22]. The work has several features that serve self-adaptiveness including future parameter values, run-time goal substitution (incompleteness), contexts that are evaluated at run time, and sensing noise, making it suitable for later-stage analysis of the adaptive system. Our proposal complements this work in that it is geared towards design-time exploration and may fit better analyses earlier in the lifecycle when, for example, analysis of a quality goal hierarchy is more pertinent. More directly connected to goal modeling, Dell' Anna et al. [11,12] introduced a framework that utilizes a translation of goal models into Bayesian Networks (BNs) to allow checking of run-time compliance to requirements models and support thereby the evolution of a Socio-Technical System (STS) that is modeled by the goal models. The BNs, developed based on available system data, encode how various factors, including design-time assumptions of the analysts, affect the satisfaction of high-level requirements, in a vein similar to the one followed by Letier et al. [20,28], albeit here with a stronger focus on adaptation. With such a model in hand, the authors go on to propose a way to allow run-time automatic revision of STS requirements. The work is complementary to ours in that it focuses on the development and structure of the probability distributions of the effects of specific design decisions and optimize accordingly – comparatively, our probability representations are restricted to the level of low-level action effects – and incorporate a run-time automation component. In comparison, our approach is based on decision-theoretic optimization that includes expressions of the relative value of the top-level objectives, for when the need for dealing with multiple conflicting goals is pronounced. In addition, our proposal appears to be more expressive at the level of complex interactions of agents with their environment, modeled through preconditions, effects, (potentially) procedures etc., which is useful when the problem at hand involves complex business processes and agent operations.

In the same context of self-adaptive systems, expressing goal models in terms of model predictive control (MPC) formulations has been suggested [4] as a way to calculate optimal adaptations with respect to AHP-elicited preference aggregations of performance indicators and environmental parameters. The benefit of MPCs is that they optimize on the basis of future predicted states of the system. As with work we discussed above [11,12,20,28], the focus is on macroscopic evaluation of high-level requirements (e.g. that "80% of participants show up in a meeting", in the meeting scheduling domain) to inform low-level run-time adaptations, through what can be seen as a learning process. These features are complementary to our approach which puts forth and tackles the problem of reasoning in the presence of complex agent actions and ordering thereof, leaving run-time adaptation and learning outside its current scope. Work by Moreno et al. [38], investigates the use of MDPs within the context of adaptive systems while taking adaptation latency into account. On the matter of accuracy of probability values, the use of parametric MDPs (PMDPs) has been proposed for the analysis of the effects of perturbations in MDP model probabilities [36]. While the focus of that work is model-checking of reachability properties rather than enriching requirements

goal models with decision-theoretic elements and reasoning capabilities, the toolset utilized could be suitable for the kind of investigation we here perform through sensitivity analysis.

Another run-time-focused effort is offered by Bencomo et al. [5,6]. They apply Dynamic Decision Networks (DDNs), through modeling soft-goals as chance nodes and contribution links as probabilities that are conditional to alternative strategies employed as realizations of hard-goals (decision nodes). Furthermore, Paucar and Bencomo employ partially observable MDPs (POMDPs) for modeling MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) adaptive architectures, where, while goal models do not occupy the center stage in the model, NFRs (Non-Functional Requirements) are modeled as the main state-describing variables [18]. Interestingly, the same group has proposed an approach to combine to elicit NFR weighting through P-CNP (Primitive Cognitive Network Process) [45], which may be applicable to priority and reward elicitation in our proposal. These proposals are, again, geared towards run-time adaptation offering several features to facilitate such use. Through the use of *claims*, for example, a goal modeling construct [5], assumptions about the operational context are modeled and checked at run-time through associated monitorable variables. This way, effects to quality goals are predicated on environmental sensing. Our approach can complement such methods by focusing on design-time solution exploration with an emphasis on expressive power including, e.g., nuanced preconditions, successor state axioms, and probability and reward schemes, with also a strong focus on expressive policies. Further comparing the merits and drawbacks of this approach with that of DDNs in the context of run-time adaptation, specifically on the front of computational performance, is an interesting topic to pursue. Moreover, the adoption of POMDPs [18] adds expressiveness in the model, in that the sensing of action outcomes and of the state of the environment is not assumed to be deterministic, as we do in our plain MDP-based work. It follows, however, that the choice to work with POMDPs may imply additional modeling and computational burden.

While in many of the efforts we discussed above the focus is run-time adaptation – versus design-time exploration and analysis on which we focus here – DT-Golog has several features that can allow extension of our work to support run-time policy calculation and execution. Possibilities include systematic use of sensing actions and utilization of the full policy, as well as utilization of the on-line DT-Golog interpreter.

As a final remark, probabilistic model checkers such as PRISM [22] based on MDPs or Discrete-Time Markov Chains (DTMCs) [16] have been used in a variety of contexts, particularly for verification of system requirements and adaptive systems engineering as discussed above. Given this trend, our choice of DT-Golog over those model checkers, such as PRISM, is worth some justification. The fundamental difference of DT-Golog from such model checkers is its Golog component rather than its MDP component, that is, its ability to allow specification of complex executable *programs* that go beyond state-transition specifications. Thus, while we would use PRISM to explore various

properties of the underlying MDP formulation, translation to DT-Golog gives us a basis for developing executable modules. The result of our automated translation to DT-Golog remains on the surface of the expressive power of Golog and constitutes a skeleton for formalizing and developing the domain theory in much more detail, including, for example, domain objects (as action parameters), while loops, or complex conditionals. Given Golog's expressive power and its potential use as a programming language, for e.g., simulations or actual controllers, we found it to be particularly appealing for modeling requirements. Similar translations of goal models to non-decision-theoretic Golog have been successfully attempted in the past, e.g. by Lapouchnian et al. to ConGolog [27]. Nevertheless, the merits of model checkers such as PRISM for a variety of queries and analyses that are not the focus of DT-Golog, make a study of the translation of our extended goal models into such languages a worthwhile future project.

8 Concluding Remarks

We presented an extension to the iStar modeling language that allows modeling probabilistic tasks and reasoning about goal satisfaction alternatives on the basis of optimal expected value, for the purpose of design-time solution exploration and model analysis. Tasks are augmented with effects describing their possible outcomes and goal satisfaction is redefined based on such effects. The extended model is translated into a DT-Golog specification, which allows identification of optimal policies in terms of both expected utility and probability of successful execution. We further introduce a sensitivity analysis procedure and visualization approach that can be used to assist elicitation of relevant numbers. In our scalability analysis, the reasoning tool is found to perform reasonably well for models of practical size.

The toolset is particularly useful for design-time analysis of operationally complex requirements problems, where it is important to understand the risks associated with specific policy options and incorporate that risk in the definition of optimality. A very fitting use case is, for example, the early design of a business process, in which the various actor tasks are stochastic or run a probability of failure which is important to analyze, while, at the same time, process analysts have specific conflicting process quality objectives in mind. We briefly discuss below the possibility of extending this framework to process languages. Another application area example is that of assisting the design of self-adaptive systems, especially when there are complex human and/or software agent actions involved in the domain of interest.

One of the major advantages of our approach is the utilization of DT-Golog, which allows for writing imperative style programs of varying degrees of determinism. Hence, analysts can further refine the result of the translation to develop accurate models of the domain. Furthermore, compared to related frameworks, our approach is strongly focused on the generation of policies, that is sequences of agent actions, and, as such, it allows reasoning about

how decisions and outcomes early in the policy affect probabilities, values, and decisions that take place later, making even small models, like the running example we considered in this paper, difficult to analyze manually and unsystematically.

Nevertheless, more complex problems imply more complex effect and utility tables, making their elicitation and comprehension harder. We perceive this as the main focus for our future work, which is coping with the elicitation and representation of complex, conditional effect probabilities and utilities. Thankfully, a variety of approaches for dealing with this have long existed in the literature and can serve as a source of inspiration. For example, expected utility networks (EUNs) [39] and UCP-Networks [8] could allow for modular representation of probability and utility dependencies in a more compact way compared to exhaustively specifying the effect and utility relations. Further, elicitation techniques such as regret-based [46], in which reward is elicited based on minimization of maximum deviation from optimality seem promising directions for studying the number acquisition problem.

An additional direction for future investigation is, as we saw, exploring how the framework can best serve run-time adaptation. One opportunity for investigation is tapping into DT-Golog's expressiveness [49] as a way to develop programs utilizable for making run-time decisions. For example, modulo a fixed horizon, DT-Golog allows the specification of continuous problems – compared to episodic ones which goal models describe – and, as mentioned, can also feature richer representations of the domain. Further, proposed online versions of the interpreter [15, 49] allow efficient weaving of planning and execution which can be useful in run-time adaptive context.

Moreover, our framework is founded on pre-calculated probabilities. In the absence of initial probabilities, the question that emerges, which is also particularly relevant for adaptive systems, is how goal models can guide the learning process for accurate and efficient acquisition of those probabilities, as well as utilities. It appears that meta-constructs such as, for example, goal fulfillment episodes might need to be introduced for formulating such analyses.

Finally, it is easy to envision possible adaptation of this framework to modeling languages beyond goal models, such as BPMN [44], UML Activity Diagrams [17], or other languages for modeling processes and behavior. Taking BPMN as an example, the question would be the identification of optimal decisions within gateways (ie., decision nodes in BPMN) given probabilities and rewards of success of individual process steps. For such reasoning to be possible in DT-Golog, the appropriate probabilistic and reward extensions would need to be introduced in the language, and a translation procedure would need to be designed, whereby e.g., processes translate to DT-Golog actions and flow links into precondition axioms. Noting work that has already been conducted in the field – e.g. using stochastic simulations [13] or PRISM [21] – an attempt to utilize DT-Golog for decision-theoretic analysis of BPMN models appears to be very promising.

References

1. Ali, R., Dalpiaz, F., Giorgini, P.: A goal-based framework for contextual requirements modeling and analysis. Requirements Engineering **15**(4), 439–458 (2010). DOI 10.1007/s00766-010-0110-z. URL https://doi.org/10.1007/s00766-010-0110-z
2. Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., Yu, E.S.K.: Evaluating goal models within the goal-oriented requirement language. International Journal of Intelligent Systems **25**(8), 841–877 (2010)
3. Amyot, D., Mussbacher, G.: User Requirements Notation: The First Ten Years, The Next Ten Years (Invited Paper). Journal of Software (JSW) **6**(5), 747–768 (2011)
4. Angelopoulos, K., Papadopoulos, A.V., Silva Souza, V.E., Mylopoulos, J.: Model Predictive Control for Software Systems with CobRA. In: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'16), pp. 35–46. Madrid, Spain (2016). DOI 10.1145/2897053.2897054. URL https://doi.org/10.1145/2897053.2897054
5. Bencomo, N., Belaggoun, A.: Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks. In: Proceedings of the 19th International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 221–236. Essen, Germany (2013)
6. Bencomo, N., Belaggoun, A., Issarny, V.: Dynamic decision networks for decision-making in self-adaptive systems: A case study. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 113–122. San Francisco, CA (2013). DOI 10.1109/SEAMS.2013.6595498
7. Boland, P.J.: Statistical and Probabilistic Methods in Actuarial Science. Chapman and Hall – CRC Interdisciplinary Statistics (2007)
8. Boutilier, C., Bacchus, F., Brafman, R.I.: UCP-Networks: A Directed Graphical Representation of Conditional Utilities. In: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI'01), pp. 56–64 (2001)
9. Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In: Proceedings of the 17th Conference on Artificial Intelligence (AAAI-00), pp. 355–362. AAAI Press, Austin, TX (2000). URL http://www.cs.toronto.edu/cogrobo/Papers/dtgologaaai00.ps.Z
10. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 Language Guide. The Computing Research Repository (CoRR) **abs/1605.0** (2016). URL http://arxiv.org/abs/1605.07767
11. Dell'Anna, D., Dalpiaz, F., Dastani, M.: Validating Goal Models via Bayesian Networks. In: Proceedings of the 5th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), pp. 39–46. Banff, AB, Canada (2018). DOI 10.1109/AIRE.2018.00012
12. Dell'Anna, D., Dalpiaz, F., Dastani, M.: Requirements-driven evolution of sociotechnical systems via probabilistic reasoning and hill climbing. Automated Software Engineering **26**(3), 513–557 (2019). DOI 10.1007/s10515-019-00255-5. URL https://doi.org/10.1007/s10515-019-00255-5
13. Durán, F., Rocha, C., Salaün, G.: Stochastic analysis of BPMN with time in rewriting logic. Science of Computer Programming **168**, 1–17 (2018). DOI https://doi.org/10.1016/j.scico.2018.08.007. URL https://www.sciencedirect.com/science/article/pii/S0167642318303307
14. Félix Solano, G., Diniz Caldas, R., Nunes Rodrigues, G., Vogel, T., Pelliccione, P.: Taming Uncertainty in the Assurance Process of Self-Adaptive Systems: a Goal-Oriented Approach. In: Proceedings of the 14th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 89–99. Montreal, Canada (2019). DOI 10.1109/SEAMS.2019.00020
15. Ferrein, A., Fritz, C., Lakemeyer, G.: On-line decision-theoretic Golog for unpredictable domains. In: Proceedings of the 27th Annual German Conference on AI (KI 2004), pp. 322–336. Ulm, Germany (2004)
16. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-Time Efficient Probabilistic Model Checking. In: Proceedings of the 33rd ACM International Conference on Software Engineering, ICSE '11, pp. 341–350. Waikiki, Honolulu, HI (2011)
17. Fowler, M., Scott, K.: UML Distilled. Addison Wesley (1997)

18. Garcia Paucar, L.H., Bencomo, N.: Knowledge Base K Models to Support Trade-Offs for Self-Adaptation using Markov Processes. In: Proceedings of the 13th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pp. 11–16. Umeå, Sweden (2019). DOI 10.1109/SASO.2019.00011
19. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. In: S. Spaccapietra, S. March, K. Aberer (eds.) Journal on Data Semantics I, pp. 1–20. Springer, Berlin, Heidelberg (2003)
20. Heaven, W., Letier, E.: Simulating and Optimising Design Decisions in Goal Models. In: Proceedings of 19th IEEE International Requirements Engineering Conference (RE 2011). Trento, Italy (2011)
21. Herbert, L.T., Hansen, Z.N.L., Jacobsen, P.: SBOAT: A Stochastic BPMN Analysis and Optimisation Tool. In: M.G. Karlaftis, N.D. Lagaros, M. Papadrakakis (eds.) Proceedings of the 1st International Conference on Engineering and Applied Sciences Optimization (OPT-i), pp. 1136–1152. Kos Island, Greece (2014). URL `http://www.opti2014.org/`
22. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems. In: Proceedings of the 12 International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006), *Lecture Notes in Computer Science (LNCS)*, vol. 3920, pp. 441–444. Vienna, Austria (2006)
23. Horkoff, J., Salay, R., Chechik, M., Di Sandro, A.: Supporting early decision-making in the presence of uncertainty. In: Proceedings of the 22nd International Requirements Engineering Conference (RE'14), pp. 33–42. Karlskrona, Sweden (2014). DOI 10.1109/RE.2014.6912245
24. Horkoff, J., Yu, E.: Comparison and evaluation of goal-oriented satisfaction analysis techniques. Requirements Engineering (REJ) **18**(3), 199–222 (2011)
25. Ishizaka, A., Nemery, P.: Multi-criteria Decision Analysis: Methods and Software. Wiley (2013)
26. Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software **14**(5), 67–74 (1997)
27. Lapouchnian, A., Lespérance, Y.: Using the ConGolog and CASL Formal Agent Specification Languages for the Analysis, Verification, and Simulation of i* Models. In: A.T. Borgida, V.K. Chaudhri, P. Giorgini, E.S. Yu (eds.) Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos, pp. 483–503. Springer, Berlin, Heidelberg (2009)
28. Letier, E., van Lamsweerde, A.: Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering. In: Proceedings of the 12th International Symposium on the Foundation of Software Engineering (FSE-04), pp. 53–62. ACM Press, Newport Beach, CA (2004). URL `http://www2.info.ucl.ac.be/people/eletier/`
29. 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). DOI 10.1016/S0743-1066(96)00121-5
30. Liaskos, S., Jalman, R., Aranda, J.: On Eliciting Preference and Contribution Measures in Goal Models. In: Proceedings of the 20th International Requirements Engineering Conference (RE'12), pp. 221–230. Chicago, IL (2012)
31. Liaskos, S., Khan, S.M., Litoiu, M., Jungblut, M.D., Rogozhkin, V., Mylopoulos, J.: Behavioral adaptation of information systems through goal models. Information Systems (IS) **37**(8), 767–783 (2012)
32. Liaskos, S., Khan, S.M., Mylopoulos, J.: Replication Data for: Modeling and Reasoning about Uncertainty in Goal Models: A Decision-Theoretic Approach (2021). DOI 10.5683/SP3/R8PGP8. URL `https://doi.org/10.5683/SP3/R8PGP8`
33. Liaskos, S., Khan, S.M., Soutchanski, M., Mylopoulos, J.: Modeling and Reasoning with Decision-Theoretic Goals. In: Proceedings of the 32th International Conference on Conceptual Modeling, (ER'13), pp. 19–32. Hong-Kong, China (2013)
34. Liaskos, S., McIlraith, S.a., Mylopoulos, J.: Towards Augmenting Requirements Models with Preferences. In: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE'09), pp. 565–569. Auckland, New Zealand (2009). DOI 10.1109/ASE.2009.91. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5431732`

35. Liaskos, S., McIlraith, S.A., Sohrabi, S., Mylopoulos, J.: Integrating Preferences into Goal Models for Requirements Engineering. In: Proceedings of the 10th IEEE International Requirements Engineering Conference (RE'10). Sydney, Australia (2010)
36. Llerena, Y.R.S., Su, G., Rosenblum, D.S.: Probabilistic Model Checking of Perturbed MDPs with Applications to Cloud Computing. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), pp. 454–464. Paderborn, Germany (2017)
37. Ma, Q., de Kinderen, S.: Goal-Based Decision Making. In: M. Daneva, O. Pastor (eds.) Proceedings of the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016), pp. 19–35. Göteborg, Sweden (2016)
38. Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B.: Proactive Self-Adaptation under Uncertainty: A Probabilistic Model Checking Approach. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015), pp. 1–12. Association for Computing Machinery, Bergamo, Italy (2015). DOI 10.1145/2786805.2786853. URL https://doi.org/10.1145/2786805.2786853
39. Mura, P.L., Shoham, Y.: Expected utility networks. In: K.B. Laskey, H. Prade (eds.) In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI '99), pp. 366–373. Morgan Kaufmann, Stockholm, Sweden (1999)
40. Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Exploring Alternatives During Requirements Analysis. IEEE Software **18**(1), 92–96 (2001)
41. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research (JAIR) **20**, 379—-404 (2003)
42. Nelson, H.J., Poels, G., Genero, M., Piattini, M.: A conceptual modeling quality framework. Software Quality Journal (20), 201–228 (2012)
43. Nguyen, C.M., Sebastiani, R., Giorgini, P., Mylopoulos, J.: Multi-objective reasoning with constrained goal models. Requirements Engineering **23**(2), 189–225 (2018). DOI 10.1007/s00766-016-0263-5. URL https://doi.org/10.1007/s00766-016-0263-5
44. Object Management Group: Business Process Model And Notation (v2.0). Tech. rep. (2011)
45. Paucar, L.H.G., Bencomo, N.: ARRoW: Tool Support for Automatic Runtime Reappraisal of Weights. In: Proceedings of the 25th IEEE International Requirements Engineering Conference (RE), pp. 458–461. Lisbon, Portugal (2017). DOI 10.1109/RE.2017.58
46. Regan, K., Boutilier, C.: Regret-based reward elicitation for Markov decision processes. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI'09), pp. 444–451. Montreal, QC, Canada (2009)
47. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
48. Soutchanski, M.: An On-line Decision-Theoretic Golog Interpreter. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001), pp. 19–24. Seattle, Washington (2001). URL http://www.cs.toronto.edu/cogrobo/Papers/onlinedtgi.ps
49. Soutchanski, M.: High-Level Robot Programming in Dynamic and Incompletely Known Environments. Ph.D. thesis, Department of Computer Science, University of Toronto (2003)
50. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press (2018)
51. Yu, E.S.: GRL - Goal-oriented Requirement Language. URL http://www.cs.toronto.edu/km/GRL/
52. Yu, E.S.K.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), pp. 226–235. Annapolis, MD (1997)