Model-Driven Design and Generation of Training Simulators for Reinforcement Learning

Sotirios Liaskos[1] ✉, Shakil M. Khan[2], John Mylopoulos[3], and Reza Golipour[1]

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

Abstract. Reinforcement learning (RL) is an important class of machine learning techniques, in which intelligent agents optimize their behavior by observing and evaluating the outcomes of their repeated interactions with their environment. A key to successfully engineering such agents is to provide them with the opportunity to engage in a large number of such interactions safely and at a low cost. This is often achieved through developing simulators of such interactions, in which the agents can be trained while also different training strategies and parameters are explored. However, specifying and implementing such simulators can be a complex endeavor requiring a systematic process for capturing and analyzing both the goals and actions of the agents and the characteristics of the target environment. We propose a framework for model-driven goal-oriented development of RL simulation environments. The framework utilizes a set of extensions to a standard goal modeling notation that allows concise modeling of a large number of ways by which an intelligent agent can interact with its environment. Though subsequent formalization, the model is used by a specially constructed simulation engine to simulate agent behavior, such that off-the-shelf RL algorithms can use it as a training environment. We present the extension of the goal modeling language, sketch its semantics, and show how models built with it can be made executable.

Keywords: Goal Modeling · Reinforcement Learning · DT-Golog

1 Introduction

Over the past years, the demand for Artificial Intelligence (AI) systems has been on the rise. Such systems perform tasks requiring autonomy and complex decision making, such as driving vehicles, controlling devices, or making trading decisions. Some of these AI systems are based on Reinforcement Learning (RL), whereby intelligent software agents learn to optimize their behavior by continuously interacting with their environment [62]. RL has been studied in a variety of application domains including energy [5], traffic control [64], finance [58], and healthcare [26]. Software *RL agents* engage in goal-oriented activity, whereby they perform actions to fulfill functional goals (e.g., administer a therapy, trade

securities, control a heating device), while maximizing the satisfaction of higher level quality objectives (respectively, health outcome, profit, occupant comfort) based on experience.

Key to successfully engineering an RL agent is the ability to subject it to a large number of training interactions with the target environment. Using a *simulator* based on a model of the latter allows a large number of such interactions to take place safely and at a low cost. When such a model is accurate, the trained agent is readily deployable to the target environment. When the model is provisional, approximate, or imprecise, simulator-based training is useful for exploring the performance of different learning algorithms under alternative problem formulations and parameter settings.

We propose a framework for model-driven goal-oriented development of training simulators for RL. The framework is based on developing goal models that describe the required intentional structure of RL agents through representing how high-level agent goals are refined into low level actions, how the latter, upon their performance, give raise to stochastic outcomes, and how such outcomes, in turn, affect a variety of quality variables of interest, an aggregate of which is used to represent outcome rewards. The requirements model is then translated into an action- and decision-theoretic formal specification, which, through a set of proposed querying and simulation components, can be directly used by RL algorithms as a training workbench. In this way, the training simulator is the result of a principled requirements-based approach that fully embraces modeling both for facilitating analysis and communication of the RL agent design and for generating its training simulator.

Our main contributions include the extensions to an existing decision-theoretic goal-modeling language to allow RL agent modeling, the corresponding formalization rules, and the architecture and implementation of components that make the models executable and usable by RL algorithms. To demonstrate feasibility, a set of experiments with a selection of popular RL algorithms is performed.

The paper is organized as follows. We describe the modeling language and its semantics in Sections 2 and 3 and the generation of simulators in Section 4. In Section 5 we discuss related work and we conclude in Section 6.

2 Modeling RL Domains

2.1 Motivating Example

Consider a large-scale woodwork manufacturer who builds custom furniture and cabinetry in a make-to-order fashion. For every order they receive, they need to source the material and manufacture the requested product – among many other activities omitted here for simplicity. They have options as to how they perform these two steps. The material can be sourced from domestic or foreign sources. In the first case, the cost is higher, but in the latter case there are delay risks. Once the material is acquired, they can engage their in-house craftspersons to build the product or subcontract to a more specialized group, who use precise

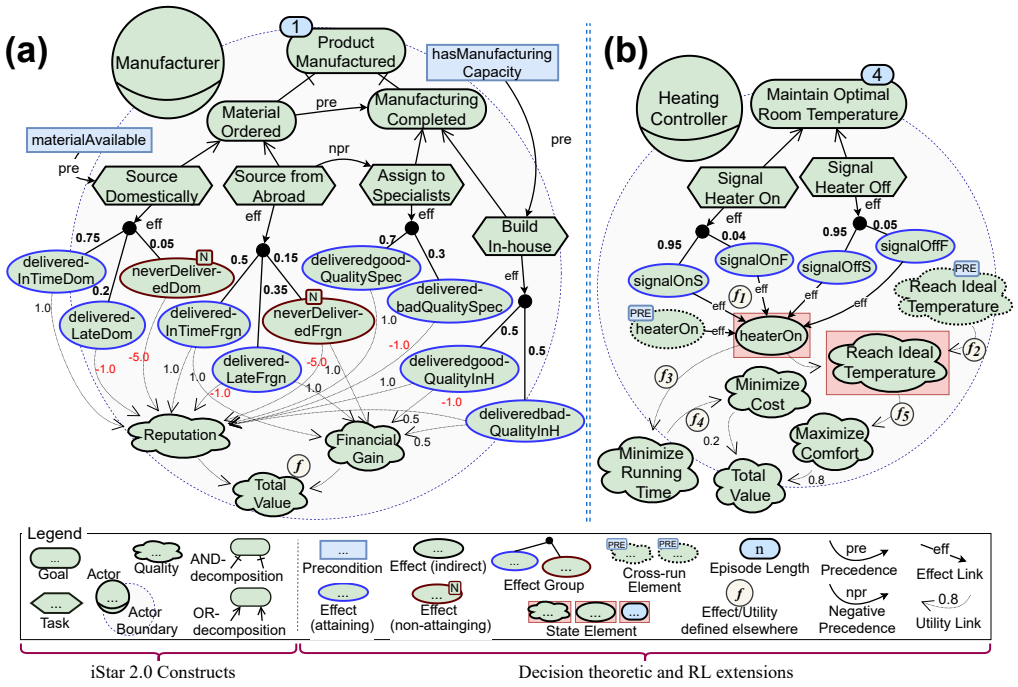

Fig. 1: An extended goal model – a (partial) meta-model can be found in [40].

manufacturing techniques but cost more and only work with domestic material. Importantly, a decision outcome of one step of the process, may affect what decision is best in a subsequent step. For example, a delay in sourcing may necessitate expedience in manufacturing to meet deadlines. In addition, choices have non-deterministic outcomes. For example, sourcing material from abroad may or may not result in a delay, and the in-house craftspersons may or may not produce a lower quality product.

The question for the manufacturer as they engage in this sequential decision making process for every incoming order is what decision they should be making at each step so that their goals are, on average, maximally satisfied in the end. Reinforcement learning (RL) refers to a set of techniques that has been proposed for addressing this problem via learning through experience [62]. Specifically, an RL agent actively engages with the environment by repeatedly performing *actions* which have the effect of (a) changing the *state* of the environment, typically in a non-deterministic way, and (b) offering to the agent a (positive or negative) *reward* reflecting the desirability of the action outcome or state change. This is repeated until a goal state is reached. The agent then restarts with a new effort to achieve the same goal. In our example, an RL agent would, for each order, chose and take a sourcing action, observe the outcome (delayed or not), proceed with a manufacturing action, observe that result (good quality or not), and move on to the next order. Various RL algorithms exist for turning these

repeated decision making sequences into a *policy*, which maps each state to an action to be taken at that state so that the total expected reward is maximized.

As RL agents need to first process a potentially large number of cases before they can start making safe and good quality decisions, it is preferable that their training is not taking place in the actual environment but an executable model thereof, i.e., a simulator. Such a model should capture the decision points that are available to the RL agent, the corresponding alternative actions, the possible effects that these actions bring about, as well as the reward structure reflecting the preferability of these effects. The latter can be challenging in RL applications within socio-technical systems, such as the one of our example, where reward structures are abstract and multi-dimensional. We next describe an extension to a standard goal modeling language that aims at developing such models.

2.2 Extending iStar for Reinforcement Learning: iStar-RL

The proposed language, which we will refer to as *iStar-RL*, extends iStar 2.0 [17], a state-of-the-art goal-oriented requirements modeling language for socio-technical systems. Two example models can be viewed in Figure 1. The one on the left (a) shows the goals of our hypothetical woodwork manufacturer.

As per the iStar 2.0 ontology, *actors*, such as *Manufacturer*, have *goals*, such as *(Have) Material Ordered*, which represent states of affairs that actors want to achieve and/or maintain. Through *AND-* and *OR-decompositions*, high-level goals are refined into lower-level ones, whereby satisfaction of all or, respectively, one subgoal(s) is necessary (resp., sufficient) for fulfilling the parent goal. At the leaf level, *tasks* signify concrete actions to be taken for fulfilling goals – e.g. *Source Domestically*. Quality goals (*qualities*) describe attributes for which actors desire some level of achievement, without the requirement that such level is precisely defined. *Contribution links* are used to signify that achievement of a goal/task affects a quality in a way that is described using an annotation on the link.

Models such as that of Figure 1(a) can encode a great number of subsets of leaf-level tasks and temporal orderings thereof that can fulfill top-level goals. Various ways for formalizing goal models for purposes similar to identifying such task sequences have been introduced – e.g. [23, 30, 36, 42, 53]. We adopt here a decision-theoretic extension to iStar for reasoning in the presence of probabilistic task effects [40, 41] and further extend it with constructs that facilitate RL.

The baseline extensions are showcased in Figure 1. A set of *domain propositions* are, firstly, introduced for describing the state of the environment at different points in time – for example, *hasManufacturingCapacity* and *materialAvailable*. Domain propositions are used in *precondition* and *effect* elements. A precondition contains a Boolean formula of domain propositions, whereas an effect contains one such predicate. Preconditions are connected to tasks through *precedence links* $\xrightarrow{pre}$ and, respectively, *negative precedence links* $\xrightarrow{npr}$ which denote that performance of the task is not possible unless (resp., if) the formula in the precondition is satisfied. Precedence (resp., negative precedence) links can also originate from goals meaning that the task cannot be performed unless

(resp., if) the origin goal has been satisfied (resp., has been attempted, i.e. at least one of its descendant tasks has been successfully performed).

Further, tasks are connected with effects through the use of *effect links* $\xrightarrow{eff}$, denoting that performance of the task from where the link originates can cause the effect which the link points at to occur, i.e., make the domain proposition contained in it true. As tasks may have several possible effects, they may be connected to *effect groups* which represent a collection of effects, each carrying a distinct probability to occur once the task is performed. The probability is added as a label on the corresponding link on the diagram. Effects are marked as *task satisfying* if their occurrence implies successful performance of the task, and *non-satisfying* otherwise. For example, an attempt to submit an order may be deemed successful (task satisfying) if it is finally delivered, despite delays, errors, etc., and non-satisfying if it is never delivered. Moreover, *indirect effects*, such as *heaterOn* in Figure 1(b), are effects whose truth status depends on other effects in a way that is defined outside the diagram – noted through an ⓕ annotation. Effect links are used to connect regular effects with indirect effects.

Further, effects, including indirect ones, are connected to qualities through *utility links*, a specialization of iStar 2.0 contribution links. These are annotated either with a number representing the amount of satisfaction that the effect, if it occurs, adds to the quality or with the ⓕ icon indicating that a more complex formula describes the relationship. Using utility links, qualities also form an hierarchy whose root o_{top} represents the overall quality – in Figure 1, this is *Total Value*. Each quality in iStar-RL is considered to be a continuous variable whose value represents the level of satisfaction of the quality at a given state.

2.3 Task Histories, Goal Runs, and Episodes

Let us now focus on tasks. Let *task (performance) history* be a sequence $tH^{|I} = [t_1, t_2, \ldots]^{|I}$ of leaf-level tasks that (a) starts from an initial state in which propositions and qualities have a value configuration I, and (b) is feasible with respect to the precondition and effect constraints, i.e., t_1 is feasible under I and each subsequent t_i is possible given the state that t_{i-1} brought about. Attempt of each task results in the occurrence of an effect. Hence, a task history is mapped to a set of possible *effect (occurrence) histories*, $eH^{|I} = [e_1, e_2, \ldots]^{|I}$ – we will henceforth omit the initial state superscript I unless needed. Further, let $\mathcal{H}$ be the set of all goals and $\mathcal{O}$ the set of all qualities in a model. The mappings $satG : \mathcal{H} \times \mathbf{eH} \mapsto \{true, false\}$ and $satQ : \mathcal{O} \times \mathbf{eH} \mapsto \mathbb{R}$, describe the satisfaction or not of a goal, and, respectively, the level of satisfaction of a quality, given an effect history eH from the set $\mathbf{eH}$ of all such. The former mapping reflects the AND/OR decomposition structure and the latter the structure of utility links; precisely how is discussed in a subsequent section.

To make effect histories meaningful for RL, we need constructs additional to the original extension [40]. Let $\mathcal{G}$ be a goal model with root goal $r_{\mathcal{G}}$. A *goal run* for goal $r_{\mathcal{G}}$ is an effect history $eH = [e_1, e_2, \ldots]$ such that (a) $satG(r_{\mathcal{G}}, eH)$, i.e. the root goal is satisfied, (*successful run*) *or* (b) no other task can be performed at eH, due to precondition constraints, a situation we will call a *deadlock*.

Back in Figure 1(a), consider effect histories *[deliveredInTimeDom, delivered-BadQualityInH]* (materials sourced domestically and in-house production was of bad quality) and *[neverDeliveredFrgn]* (materials ordered from foreign sources and were never delivered). Both histories are goal runs: the former satisfies the root goal, whereas the latter cannot be continued due to the precedence link between *Material Ordered* and *Manufacturing Completed*. However, neither *[deliveredLateDom]* nor *[deliveredInTimeFrgn]* are goal runs: in both cases, the root goal is not satisfied and there are actions that are still possible. The level of satisfaction *satQ* of quality *Reputation* for each of the aforementioned four (partial) histories is +1.0 - 1.0 = 0, -5.0, -1.0, 1.0, respectively, calculated by adding up the annotations of utility links originating from effects included in the history.

We further define an *episode* to be a concatenation of n goal runs $eP^{|I} = [eH_1^{|I}, eH_2^{|I}, ..., eH_n^{|I}]$ such that for all $eH_i^{|I}$, $i < n$ the root goal is satisfied. In other words, an episode describes a history of repeated goal runs all of which successful except for the last one, which may be either successful or a deadlock. For $n > 1$ the episode is a *multi-run episode* and for $n = 1$ a *single-run episode*. We can specify the maximum number of runs that comprise an episode (*episode length*) as an annotation next to the root goal - see Figure 1. In multi-run episodes, each run is, by default, assumed to start from the same initial configuration I. We may however wish to designate elements that carry their values across goal runs. We accomplish this through *cross-run elements*.

To appreciate the rationale for multi-run episodes and the role of cross-run elements in such episodes, consider the model of Figure 1(b). It describes the function of a heating device controller, contrived here to showcase additional features of the language. The controller periodically signals wirelessly to the device to turn *on* or *off*. This *on* or *off* signal can be lost with a probability. The overall quality accrued from a sequence of signaling tasks is a function of the distance of the temperature to an ideal one and the amount of time heating is on, which represents cost and environmental impact. To make meaningful optimal decisions we need to look at the quality value accumulated over a sequence of goal runs. In the diagram this is set to four (4). Hence, if the controller makes a decision every, e.g., 5 minutes, which constitutes one goal run, an entire episode spans 20 minutes and overall quality is calculated for all 4 decisions made.

In Figure 1(b), cross-run elements (dashed outline and with a "PRE" annotation) represent the (truth) value of the enclosed proposition or quality in the previous state. The previous state is the configuration of truth values before the latest action was performed within the episode, irrespective of goal run boundaries. Thus, the truth status of the proposition within indirect effect *heaterOn* (solid line effect), depends on its truth status at the end of the previous state (dashed line effect) and the current state of four regular effects in the second run, represented through the remaining four incoming effects. The symbol f_1 denotes that the exact formula that translates the truth value of these five propositions into the new truth value for *heaterOn* is specified outside the diagram. Likewise, the current value of *Reach Ideal Temperature* depends on its previous value and the current value of *heaterOn*. Again, a symbol f_2 denotes that the formula for

combining the two values is specified outside the diagram. Note that whether an element is discrete or continuous is orthogonal to its status as cross-run.

We finally designate the *exported state set* to be the set of propositions and qualities to be used for the calculation of policies. By default, we assume that the problem is modeled as a *discrete-state* one, i.e., the exported state set is the configuration of values of all propositions – the *discrete (exported) state set*. However, in some cases it is useful to use qualities as part of the set. In the heating controller example, whether the heater should turn on or off more obviously depends on the satisfaction level of *Reach Ideal Temperature* than the history of *on*/*off* actions. In Figure 1 we put a shaded rectangle at the background of an indirect effect or a quality, to mark its inclusion in the exported state. When the exported state contains at least one non-propositional element, we model the problem as a *continuous-state* one. In Figure 1(b) we designate variables *heaterOn* and *Reach Ideal Temperature* to exclusively comprise the exported state set – a *continuous exported state set*. Note that such designation does not affect how validity of effect histories is established, which is based solely on the discrete state set; hence the clarifying "exported" qualification. Further, the cross-run status of an element is orthogonal to its inclusion in the exported set.

2.4 iStar-RL and Reinforcement Learning

Let us now sketch how iStar-RL models can be used by an RL agent to allow for optimal decision making. Recall that RL-agents observe the state of the environment, perform an action from a set of available ones, and sense the state that results from the action along with the reward that the action yields. In our case, the state of the environment is, as we saw, the exported state, i.e., a combination of values of designated domain propositions and/or qualities. Upon sensing that state, the RL agent performs a task, which brings about one of the corresponding effects, which, in turn, augments the current episode eP with one more effect, and may also imply updated $satG$ and $satQ$ values. The RL agent will sense the new state and perceive the total value $satQ(o_{top}, eH_i)$ as the reward of the latest action. It will then repeatedly proceed with the next action until the episode is over, i.e., it reaches the maximum number of successful runs or a deadlock. During training, the RL agent will attempt a great number of such episodes, aimed at identifying a policy, i.e., a mapping from state to tasks that, when repeatedly followed, maximizes the average total value.

For the RL agent to accomplish such training using the iStar-RL model, the latter needs to be executable. In the next section, we describe how iStar-RL models can be formalized, aiming at both clarifying their semantics and paving the way for generating executable simulations of such models.

3 Semantics

3.1 DT-Golog

The semantics of iStar-RL are defined by means of its translation to DT-Golog, a high-level agent programming language [12, 61] based on the Situation

Calculus [59]. The basic constructs of DT-Golog are fluents, stochastic (or agent) actions, nature actions, and situations. *Fluents* play the role of state features and have different truth values in different situations. They are represented through predicates such as *materialDelivered(s)*, with the situation s as one of the parameters. *Stochastic actions* a are first-order terms signifying specific activity initiated by agents and may have several alternative outcomes, each occurring with a different probability. These outcomes are modeled through a set of *nature actions* $\hat{A} = \{\hat{a}_1, \hat{a}_2, ...\}$ associated with a through predicate $choice(a, \hat{A})$. The corresponding probabilities are represented using $prob(\hat{a}, v, s)$, where v is the probability of the occurrence of $\hat{a}$. Further, a *situation* s denotes a sequence of actions. The function $do(\hat{a}, s)$ denotes the situation which results from the performance of nature action $\hat{a}$ in situation s. A special constant S_0 denotes the initial situation, where no action has been performed.

A DT-Golog specification contains axioms that prescribe what actions are possible in different situations and how the truth value of fluents is affected by the performance of actions. The former, *precondition axioms*, are of the form:

$$\forall s.poss(a, s) \leftrightarrow \Pi_a(s)$$

where $\Pi_a(s)$ is a fluent formula and special predicate $poss(a, s)$ states that action a is executable in situation s. The latter, *successor state axioms*, are of the form:

$$\forall \hat{a}, \mathbf{x}, s.f(\mathbf{x}, do(\hat{a}, s)) \leftrightarrow \Phi_f(\mathbf{x}, \hat{a}, s)$$

where $\mathbf{x}$ signifies a list of arguments, f a fluent symbol and $\Phi_f(\mathbf{x}, \hat{a}, s)$ a formula whose truth value depends on the parameters, the current situation, and the nature action in question. Finally, by defining:

$$reward(v, s) \leftrightarrow \Psi_r(v, s)$$

where $\Psi_r(v, s)$ is a formula grounded on fluents, it is possible to assign a reward value v to any situation or action.

3.2 From iStar-RL models to DT-Golog specifications

The DT-Golog-based semantics of iStar-RL is based on a treatment offered by Liaskos et al. [40], with the necessary additions for supporting RL. In what follows, let a goal model $\mathcal{G}$ contain a set $\mathcal{H}$ of goals, a set $\mathcal{T}$ of tasks, a set $\mathcal{E}$ of effects, a set $\mathcal{Q}$ of domain predicates, and a set $\mathcal{O}$ of qualities.

Primitives. Each element in the set is translated to a DT-Golog primitive as follows. For each domain proposition $q \in \mathcal{Q}$ introduce a fluent $\phi_q(s)$. For each task $t \in \mathcal{T}$ associated with an effect group $\mathcal{E}_t \subseteq \mathcal{E}$ introduce the following: a stochastic agent action a_t and a set of nature actions N_t each $\hat{a}_t \in N_t$ associated with an effect in $\mathcal{E}_t$. For each task $t \in \mathcal{T}$ and goal $h \in \mathcal{H}$, introduce fluents $\phi_t(s)$ and $\phi_h(s)$. For each quality $o \in \mathcal{O}$ introduce two fluents $\phi_o^{(r)}(v, s)$ and $\phi_o^{(m)}(v, s)$, respectively called *current satisfaction fluent* and *cumulative satisfaction fluent*; for both fluents, parameter $v \in \mathbb{R}$ represents the satisfaction value of o.

Given the 1-1 correspondence between iStar-RL effects and situation calculus nature actions, effect histories $eH = [e_{t_1}, e_{t_2}, \ldots]$ also map 1-1 to situations $s = do(\ldots do(\hat{a}_{t_2}, do(\hat{a}_{t_1}, s_0))\ldots)$, where $\hat{a}_{t_i}$ is the nature action corresponding to effect e_{t_i}. For a goal h and quality o: $satG(h, eH)$ iff $\phi_h(s)$ and $satQ(o, eH) = v$ iff $\phi_o^{(r)}(v, s)$, where s is the situation corresponding to effect history eH.

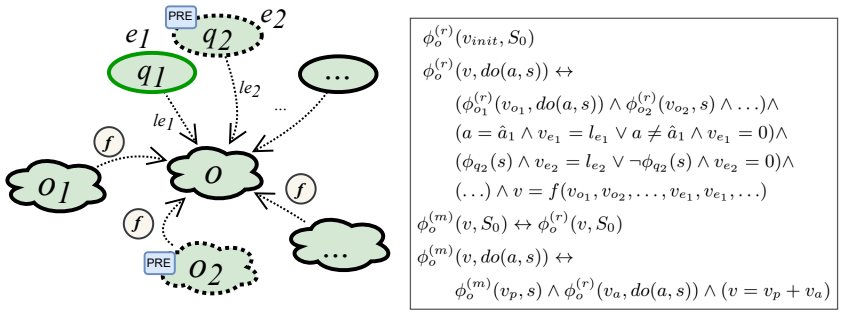

Fig. 2: Constructing quality formulae

Successor State Axioms. For each domain predicate q, collect all effects $e_1, e_2, \ldots$ that mention it and consider the corresponding $\hat{a}_1, \hat{a}_2, \ldots$ nature actions. Introduce the following successor state axiom:

$$\phi_q(do(a,s)) \leftrightarrow (a = \hat{a}_1 \vee a = \hat{a}_2 \vee \ldots) \vee \phi_q(s).$$

Attempt and Attainment Formulae. Consider each task t connected with an effect group $\mathcal{E}_t$. Consider *all* the effects of $\mathcal{E}_t$, and the domain predicates $q_1^{(t)}, q_2^{(t)}, \ldots$ contained in them. Generate the following *task attempt formula* for t:

$$\phi_t^{(att)}(s) \leftrightarrow \phi_{q_1^{(t)}}(s) \vee \phi_{q_2^{(t)}}(s) \vee \ldots$$

A *task attainment formula* $\phi_t(s)$ is defined similarly with the only difference being that it excludes non-satisfying effects. Introduce also a *goal attainment formula* for each goal h as follows:

$$\phi_h(s) \leftrightarrow f_{\text{AND/OR}}(\phi_{t_1}(s), \phi_{t_2}(s), \ldots)$$

where $t_1, t_2, \ldots$ are the tasks that are descendants of h in the goal hierarchy, $\phi_{t_1}(s), \phi_{t_2}(s), \ldots$ their corresponding task attainment formulae, and $f_{\text{AND/OR}}$ an AND/OR formula reflecting the corresponding goal decomposition structure. A *goal attempt formula* for h, $\phi_h^{(att)}(s)$, is similarly defined but grounded on task attempt, rather than attainment, formulae.

Quality Formulae. Consider each quality or domain variable $o \in \mathcal{O}$, all the contribution links toward it coming from effects $e_1, e_2, \ldots$ labeled with values $l_{e_1}, l_{e_2}, \ldots$, and all contribution links coming from other qualities $o_1, o_2, \ldots$ – see Figure 2 left side. Recalling that o is associated with fluents $\phi_o^{(r)}$ and $\phi_o^{(m)}$, the value of each is defined by the formulae at the right side of Figure 2. In the formulae, $\hat{a}_i$ are the nature actions associated with effects e_i. Intuitively, the value v of quality o is a function f of the corresponding current or previous (depending on cross-run status) values of the other qualities and the labels of utility links coming from effects that are(/were) currently(/previously) true.

Action Precondition Axioms. For each task t which receives precedence links from a precondition element, a set of tasks $\{t_1, t_2, \ldots\}$, a set of goals $\{h_1, h_2, \ldots\}$ and a set of effects containing predicates $\{q_1, q_2, \ldots\}$ introduce:

$$poss(a_t, s) \leftrightarrow \phi_{q_1}(s) \wedge \phi_{q_2}(s) \wedge \ldots \wedge \phi_{t_1}(s) \wedge \phi_{t_2}(s) \wedge \ldots$$
$$\wedge \phi_{h_1}(s) \wedge \phi_{h_2}(s) \wedge \ldots \wedge g_{prec}(s) \wedge \neg\phi_t^{(att)}(s)$$

where $g_{prec}(s)$ is the formula inside the precondition element, grounded on fluents of type $\phi_q(s)$, and $\phi_t^{(att)}(s)$ is the attempt formula for the task in question itself. As will become apparent below, the latter addition prevents the RL learning agent from selecting the same task more than once in a given goal run. The above formula can be extended to include $\xrightarrow{npr}$ incoming links, omitted here for brevity; however it must be noted that for these links specifically, we utilize task and goal *attempt* formulae rather than *attainment* ones.

Reward. The total reward calculated for each situation is simply the *current* quality value of what has been designated as o_{top} as per the quality formula:

$$reward(v, s) \leftrightarrow \phi^{(r)}_{o_{top}}(v, s)$$

Probabilities. For each link that connects an effect e with its effect group, introduce $prob(\hat{a}, p, \cdot)$, where $\hat{a}$ is the nature action associated with e, and p is the probability label of the link – independent of situation, hence $\cdot$ instead of s.

OR-decomposition exclusivity. We finally demand that all children of OR-decompositions are connected in pairs with $\xrightarrow{npr}$ links. More formally, let $g_1, g_2, \ldots$ be children (goals or tasks) of an OR-decomposition. Then, assume $g_i \xrightarrow{npr} g_j$ for all $i \neq j$. The additions appear in action precondition axioms for the involved goals, following from the semantics of $\xrightarrow{npr}$.

The above translation process is automatable. A first prototype we have built to demonstrate this is capable of automatically generating the bulk of DT-Golog specification, leaving currently only a minimum of finishes to users [37].

4 Making Models Executable

4.1 The RL Simulation Components and their Function

So far we have discussed the iStar-RL modeling language and how it can be formalized into a DT-Golog specification. We now describe how the latter specification can serve as a domain simulator usable by software agents implementing arbitrary RL algorithms. We adopt a widely used interface specification for RL simulators, Open AI's Gym framework [1], called *gym.Env*, which offers a small collection of functions that satisfy the RL observe-decide-act interaction pattern. Specifically, the most important of *gym.Env*'s functions is:

observation, reward, terminated, info ← *step(action)*

The function requests the simulator to perform *action*, identified as an integer within a range, and return: (a) the state which results from the performance of *action*, which is encoded in variable *observation* as a state-identifying integer for discrete-space problems or as an array of real values for continuous-space problems, (b) the *reward* obtained for performing the action, and (c) whether the current episode is *terminated*, and (d) other miscellaneous *info*. Our goal is, hence, to implement *step* (along with other auxiliary functions) such that it behaves in a way that is compliant to iStar-RL models of our choice.

To achieve this we introduce two software components. The *Query Engine* (*QE*) offers a number of functions that answer queries about the domain relative to a given history of actions, such as what fluents are true and what actions are

possible. These services are used by *GMEnv*, a second component which implements the *gym.Env* standard. *GMEnv* maintains information about the current state of execution of a simulated episode, and executes actions or relays information as per the requests of the client environment. The latter is an arbitrary RL agent, implementing some RL learning algorithm (e.g, A2C [46], Deep Q Network (DQN) [47], etc.) and requiring the *gym.Env* interface for its training. We examine the development of *QE* and *GMEnv* in sequence.

4.2 The Query Engine (QE)

QE offers a set of functions whereby the iStar-RL model can be queried with respect to an effect history *eH*. The supported functions are listed in Table 1 and are implemented as logic programs in Prolog and in accordance to DT-Golog semantics also seen in the table. Below we present these semantics in more detail.

Extracting State Information. Consider the set $\mathcal{Q} = q_1, q_2, \ldots$ of domain predicates in the goal model and an effect history $eH = [e_1, e_2, \ldots]$. Define also list $L_{\mathcal{Q}} = [ql_1, ql_2, \ldots]$ where $ql_i = 1$ if the predicate q_i is satisfied after effect history eH has been observed, and $ql_i = 0$ otherwise. For discrete-state problems, $L_{\mathcal{Q}}$ offers a representation of discrete exported state and it is easily translatable to an integer identifier. For continuous-state problems, list $L_{\mathcal{O}} = [ol_1, ol_2, \ldots]$, where $ol_i = satQ(o_i, eH)$, represents continuous exported state for the goal model after eH for continuous exported state set $\{o_1, o_2, \ldots\} \subseteq \mathcal{O}$.

The semantics of $L_{\mathcal{Q}}$ and $L_{\mathcal{O}}$ in DT-Golog terms are understood as follows. Recall that there is a 1-1 correspondence between domain predicates $\mathcal{Q} = \{q_1, q_2, \ldots\}$ and DT-Golog fluents $\Phi_{\mathcal{Q}} = \{\phi_{q_1}, \phi_{q_2}, \ldots\}$, as well as between an effect history eH and a DT-Golog situation s. The following rule then defines $L_{\mathcal{Q}}$ in terms of situation s:

$$getState(s, L_{\mathcal{Q}}) \leftrightarrow (\phi_{q_1}(s) \wedge (ql_1 = 1) \vee \neg\phi_{q_1}(s) \wedge (ql_1 = 0)) \wedge (\phi_{q_2}(s) \wedge (ql_2 = 1) \vee \neg\phi_{q_2}(s) \wedge (ql_2 = 0)) \wedge ...$$

Thus, $getState(s, L_{\mathcal{Q}})$ holds when binary list $L_{\mathcal{Q}}$ captures the truth value of every fluent in situation s. The predicate is the semantics of *QE* function `getState(eH): bit[]` seen in entry 5 of Table 1. For continuous exported states, recall that qualities $o \in \mathcal{O}$ in the goal model are associated with fluents of the form $\phi_o^{(r)}(v, s)$ in which v is a real value representing o's satisfaction $satQ(o, eH)$. Hence, for continuous exported state set $\{o_1, o_2, \ldots\} \subseteq \mathcal{O}$:

$$getContState(s, L_{\mathcal{O}}) \leftrightarrow \phi_{o_1}^{(r)}(v_1, s) \wedge (ol_1 = v_1) \wedge \phi_{o_2}^{(r)}(v_2, s) \wedge (ol_2 = v_2) \wedge ...$$

The predicate effectively maps a situation s with the value of fluents representing the qualities included in the exported state set, and constitutes the semantics of *QE* function `getContState(eH):float[]` – entry 6 of Table 1.

Episodes and Goal Runs. We now express the semantics of goal runs in terms of DT-Golog constructs. Recall that a task history is a goal run iff the root goal is satisfied or it leads to a deadlock. Recall also that for goal h, $satG(h, eH)$ iff $\phi_h(s)$, where s is the situation corresponding to history eH. Thus, root goal $r_{\mathcal{G}}$ is satisfied iff $\phi_{r_{\mathcal{G}}}(s)$. Secondly, to decide if a situation s is a deadlock we examine if any of the action precondition axioms allow the performance of any task at s. Let $\mathcal{N}_{\mathcal{G}}$ to be the set of all nature actions $\hat{a}$ (each corresponding to an

	QE Function	Semantics		QE Function	Semantics
1	`possibleAt(t,eH): bool`	$poss(a_t, s)$	7	`done(eH):bool`	$done(s) \equiv \phi_{r_\mathcal{G}} \vee deadlock(s)$
2	`getOutcomes(t): integer[]`	$choice(a_t, \{\hat{a}_t^{(1)}, \hat{a}_t^{(2)}, ...\})$	8	`deadlock(eH): bool`	$\forall \hat{a} \in \mathcal{N}_\mathcal{G}. \neg poss(\hat{a}, s)$
3	`getProbs(t,eH): float[]`	$getProbs(a_t, \{p_t^{(1)}, p_t^{(2)}, ...\}, s)$	9	`getCrState(eH): float/bool[]`	$crossState(L_{CR}, s)$
4	`reward(eH): float`	$reward(s, r)$	10	`setCrState(X)` (`X`: *initializations*)	assert(X)
5	`getState(eH):bit[]`	$getState(s, L_\mathcal{Q})$	11	`rootAchieved(eH): bool`	$\phi_{r_\mathcal{G}}(s)$
6	`getContState(eH): float[]`	$getConState(s, L_\mathcal{O})$			

Table 1: Query Engine (QE) functions and their semantics. The functions assume a mapping of nature and stochastic actions to integers, hence `t` and `eH` are respectively an integer and an array thereof.

effect $e \in \mathcal{E}_\mathcal{G}$), a situation s is a deadlock according to the following definition:

$$deadlock(s) \equiv \forall \hat{a} \in \mathcal{N}_\mathcal{G}. \neg poss(\hat{a}, s)$$

where $poss(\hat{a}, s)$ are, as we saw, the left-hand sides of action precondition axioms. Given the above, we can now define predicate $done(s)$ (entry 7 of Table 1) that holds when a situation s is a complete run with respect to root goal $r_\mathcal{G}$:

$$done(s) \equiv \phi_{r_\mathcal{G}}(s) \vee deadlock(s)$$

Cross-run Elements. Recall from Figure 2 that in the initial situation S_0, qualities are assigned an initial value v_{init}, while all other predicates/fluents are assumed to be false. *QE* allows the client environment to both assert these initial values and to read the values at a given situation s. This is useful for implementing cross-run elements within multi-run episodes. Let $L_{CR} = \{ol_1, ol_2, ..., ql_1, ql_2, ...\}$ represent the values of all cross-run qualities $o_1, o_2, \ldots$ and propositions $q_1, q_2, \ldots$, where $ol_i = v$ iff $\phi_{o_i}(v, s)$ and $ql_i = 1$ iff $\phi_q(s)$, 0 otherwise. Predicate $crossState(s, L_{CR})$ (the semantics of *QE*'s `getCrState` – see entry 9 of Table 1) allows extraction of cross-state information given situation s corresponding to effect history eH:

$$crossState(s, L_{CR}) \leftrightarrow \phi_{o_1}^{(m)}(v_1, s) \wedge (ol_1 = v_1) \wedge \phi_{o_2}^{(m)}(v_2, s) \wedge (ol_2 = v_2) \wedge \ldots$$
$$[\phi_{q_1}(s) \wedge (ql_1 = 1) \vee \neg\phi_{q_1}(s) \wedge (ql_1 = 0)] \wedge [\phi_{q_2}(s) \wedge (ql_2 = 1) \vee \neg\phi_{q_2}(s) \wedge (ql_2 = 0)] \wedge \ldots$$

Dynamically defining initial states is a matter of asserting in the specification, using `setCrState`, the list of initializations $\{\phi_{o_1}(v_1', s_0), \phi_{o_2}(v_2', s_0), \ldots, \Phi_q^T(s_0)\}$, where v_i' are the desired initial values, $\Phi_q^T(s_0)$ the subset of fluents representing propositions that are true in s_0 – see entry 10 of Table 1. Thanks to the above two functions, *GMEnv* can implement multi-run episodes by reading the values of cross-run elements $o_1, o_2, \ldots, q_1, q_2, \ldots$ at the end of a run using `getCrState` and setting these as the initial values of the next episode using `setCrState`.

Other predicates. The query engine offers additional functions, which can be viewed in Table 1, along with their DT-Golog semantics. One of them:

$$getProbs(a_t, \{p_1, p_2, \ldots\}, s) \leftrightarrow choice(a_t, \{\hat{a}_t^{(1)}, \hat{a}_t^{(2)}, \ldots\}) \wedge prob(\hat{a}_t^{(1)}, p_1, s) \wedge prob(\hat{a}_t^{(2)}, p_2, s) \wedge \ldots$$

uses $choice(a_t, \hat{A})$ to collect probabilities $\{p_1, p_2, \ldots\}$ for all nature actions $\hat{A} = \{\hat{a}_t^{(1)}, \hat{a}_t^{(2)}, \ldots\}$ associated with task t's stochastic action a_t.

Algorithm 1 Implementation of the Step function of GMEnv.

```
1: GLOBAL: eH,tH = []                          ▷ run-wide effect and task history
2:         eH_Ep,tH_Ep =[]                     ▷ episode-wide effect and task history
3:         curRun = 0                          ▷ the goal run under current consideration
4:         penaltyReward                       ▷ default penalty for infeasible actions
5:         qe                                  ▷ reference to a QE implementing object
6: function STEP(t)
7:    if qe.possibleAt(t,eH) then
8:       E_t = qe.getOutcomes(t)                          ▷ E_t : list of effects
9:       P_t = qe.getProbs(t,eH)                ▷ P_t: list of effect probabilities
10:      e_t = pickRndAction(E_t, P_t)                    ▷ randomly pick effect
11:      eH = append(eH,e_t)                              ▷ append effect to history
12:      tH = append(tH,t)                                ▷ append task to history
13:      reward = qe.reward(eH)                           ▷ retrieve reward
14:      state = qe.getState(eH)                ▷ retrieve new discrete state
15:      c_state = qe.getConState(eH)                     ▷ optional: retrieve
                                                            new continuous state
16:      n_state = bitToInt(state)                        ▷ bit array to int
17:   else
18:      reward = penatlyReward                 ▷ penalize infeasible action
19:   end if
20:   if qe.rootAchieved(eH) then
21:      X = constructInitClauses(qe.getCrState(eH))
22:      qe.setCrState(X)
23:      eH_Ep.append(eH),tH_Ep.append(tH)
24:      eH,tH = []
25:      curRun = curRun + 1
26:   end if
27:   return n_state, reward, DONE, [c_state]
28: end function
29:
30: function DONE
31:   return ((curRun == N) or qe.deadlock(eH))
32: end function
```

4.3 The GMEnv component

GMEnv implements *gym.Env* through utilizing *QE*'s services. While *QE* is stateless, *GMEnv* maintains episode information including the history of tasks that have been attempted from the beginning of an episode, the effect history that has resulted from these attempts, the run count since the episode's beginning, as well as the state after the performance of the latest action.

Of the *gym.Env* functions that *GMEnv* implements, the most important is, as we saw, `step(task): state, reward, terminate, info`. Algorithm 1 sketches the implementation of the function. The function is iteratively called by the RL agent, with parameter a task `t` of its choosing. Upon its call, `step` checks first if the task is feasible given the current history of effect occur-

	Model Characteristics				Learning Tests									
	Model	Size	Run #	State Space	Learning Reward A2C	Learning Reward DQN	Learning Reward PPO	Rnd.	Training Steps A2C	Training Steps DQN	Training Steps PPO	Training Time (s) A2C	Training Time (s) DQN	Training Time (s) PPO
Discrete	Material Ordered	1,12	1	2^6	0.924	0.924	0.924	0.773	10K	10K	10K	6.813	5.000	24.00
			2	2^{12}	1.711	1.712	1.715	1.466	10K	10K	10K	106.2	4.469	113.9
	Product Manuf.	2,20	1	2^{10}	0.477	0.480	0.465	0.058	10K	150K	20K	81.97	848	147.8
			2	2^{20}	0.946	0.705	0.710	0.065	10K	80K	20K	8,123	23.49K	20.32K
	Organize Travel	3,29	1	2^{14}	0.789	0.694	0.788	0.059	20K	150K	10K	400.3	1,957	436.7
	HVAC Control	1,13	4	2^{16}	-1.36	-1.37	-1.36	-5.041	10K	70K	10K	585	702.4	724.2
Conts.	Product Manuf.	2,20	1	$\mathbb{R}^2$	0.456	0.472	0.048	-78.42*	50K	80K	50K	778.4	972.9	181.3
			2	$\mathbb{R}^2$	0.286	0.667	0.325	-90.95*	70K	4.5M	70K	62.59	30,379	343.5
	HVAC Control	1,13	4	$\mathbb{R}^2$	-1.36	-1.36	-1.37	-5.00*	10K	110K	10K	709.5	390.1	769.8

Table 2: Training with off-the-shelf RL agents. Times in CPU seconds. (*) next to an **Rnd.** reward signifies inclusion of deadlock penalty. Model size n, m: n is total # of OR-nodes, m total # of elements.

rences, stored in `eH`, through the use of *QE*'s `possibleAt(t,eH)`. If yes, it uses `getOutcomes(t)` to retrieve a list of possible effects $\mathtt{E_t}$ that `t` may have and through `getProbs(t,eH)` their probabilities $\mathtt{P_t}$. A random choice on the basis of the probabilities is made and both task `t` and the chosen effect $\mathtt{e_t}$ are appended to the corresponding lists, `tH` and `eH` respectively. The state resulting from the performance of `t` is calculated through `getState(eH)`, which translates a history of effect occurrences to an array of bits representing the truth values of domain propositions. An integer representation of the bit array, `n_state`, is, in turn, returned to the calling environment as per the *gym.Env* requirements. Likewise, `getConState(eH)` is called to retrieve any continuous exported state set.

If task `t` is not feasible at `eH`, `step` does not proceed with any changes to history lists and state, but may, based on user configuration, result in a negative reward to bias the learning procedure against performance of the task in the specific state. On the other hand, if the goal is achieved, the current run has concluded at `eH`. Consequently (a) the cross-run elements at `eH` are retrieved and reasserted as initial state for the next run, (b) the action and effect lists are added to the episode-wide record and (c) reset, and (d) the run counter increases by one. As we saw above, the episode is `done` if the root of the Nth goal run has been achieved or a deadlock has been detected.

Finally, the second important *gym.Env* function to be implemented by *GMEnv* `reset()`, simply empties the history lists and resets state to its initial values.

4.4 In Action

The proposed components *QE* and *GMEnv* can be used by off-the-shelf algorithms for learning optimal policies for any appropriately formalized iStar-RL model. We performed tests with three such implementations, namely, Advantage Actor Critic (A2C) [46], Deep Q Network (DQN) [47] and Proximal Policy Optimization (PPO) [60], which are part of the Stable-Baselines3 RL package [57]. For the experiments, we used discrete and continuous versions of models such as those

of Figure 1. As a benchmark, for discrete-space problems, we also calculated optimal policies and their expected rewards using the DT-Golog interpreter. An Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 4 Core(s), 16Gb RAM computer was used. As shown in Table 2, in all cases, using default hyper-parameters, the RL agents converged close to the optimal reward and away from the reward of the random policy. The tool and a reproducibility package can be found at [38].

Note, that, while the formalized iStar-RL models can be used by DT-Golog for exact, search-based identification of optimal policies as presented in our earlier work [40], that option is restricted to discrete-state problems and requires accurate probabilities. In the absence of accurate probabilities, where neither exact [40] nor learning-based methods (enabled in this paper) can calculate the true optimal, generating simulators as described here can still be useful for exploring the behavior of different RL algorithms using provisional probabilities. The utility of the toolset for such exploratory/feasibility analysis work is hinted at by the experimental results of Table 2 in which the three algorithms performed differently both in terms of optimization outcome and in terms of numbers of steps and time required.

5 Related Work

Substantial interest has developed over the past few years in studying the intersection of conceptual modeling (CM) with artificial intelligence (AI), a space referred to as CMAI [10, 11]. The latter describes both the application of AI to support CM tasks (AI4CM), and reversely, the application of CM to systematize and support various qualities of AI-intensive systems (CM4AI). In the CM4AI space, where our work naturally fits, the use of models to organize and support the ML development pipeline from input collection and preparation to training and inference is explored [18]. The use of models has specifically been proposed for quality assuring the AI/ML process [33], detecting bias [66], supporting meta-learning [28], chatbot generation [56], or generation of neural architectures [35].

The problem of systematically devising AI-intensive systems has also been studied from the RE point of view. A major theme in that context is the quality of the end-result [27], with an emphasis on explainability [13–15], how such systems can be specified [9, 63], and how the RE process can be organized [55]. In search for a solution to the RE4AI [3] problem, Nalchigar et al. [51, 52] propose a goal-oriented conceptual framework for expressing machine learning requirements and designs organized around three main modeling views (business, analytics design, and data preparation). Ahmad et al. [3] review, among other things, modeling approaches for conducting RE for AI, to find that Goal Oriented RE languages, along with UML/SysML, are the most popular for modeling in that space.

Goal models have indeed been known to be effective vehicles for allowing formal reasoning about requirements and designs in various ways [19, 20, 23, 30, 36, 43]. This capability of goal models has been utilized also in the area of adaptive systems [6, 8, 22] and multi-agent systems [25]. The modeling approach proposed here, iStar-RL, is heavily based on an iStar dialect proposed for model-

ing decision-theoretic domains and, through formalization, perform search-based reasoning thereof [40, 41]. The strength of the modeling approach we adopt, compared to common approaches for modeling probabilistic transition systems (e.g., [32]) lies in the combined ability of goal models both to concisely encode high-variability processes and behaviors, stemming from overarching stakeholder goals, and to compare variants vis-à-vis intricate, multi-dimensional quality requirements structures [50]. Hence, efforts such as on probabilistic logic shields [65] using ProbLog [34], or on using DT-Golog for Q-learning [7], indicate promising destinations of iStar-RL transformations. Probabilistic reasoning using conceptual models other than goal models, such as BPMN [54], has also been proposed [21, 31], with no clear connection to RL however.

6 Concluding Remarks and Future Work

We presented a framework for goal-oriented modeling and generation of simulators for training RL agents. Through an extension to a standard goal modeling language, designers capture the goals, action space, and reward structure of the desired RL agents, along with the state and effect model of the environment the agents are meant to interact with. The resulting models are subsequently translated to a formal specification, which is used by a set of interpreting components to allow step-wise model execution, which is, in turn, directly usable by a variety of RL algorithms. The framework is aimed at supporting feasibility and performance analyses of training techniques against different action, reward, and probability models, and even at identifying a directly usable optimal policy when reward and probability models are deemed accurate but search-based methods are computationally forbidding or otherwise inapplicable.

Future research can focus on applications, tool support, and exploration of how goal models can support learning quality. Case studies can be helpful in assessing the utility of our approach in various domains, such as behavioral customization [39] and adaptive systems [24, 48] with an emphasis on adaptive business processes [16], whose socio-technical nature fits naturally our i^*-based approach. Enhancing our translation tool [37] to allow full automation will facilitate such studies. Further, while iStar 2.0 is a natural choice for agent modeling, extending the work to other languages, both goal-modeling ones, e.g., URN [4], and others, such as business process modeling ones [54], may be of value for the respective user communities. In addition, given the richness of the proposed extensions, studying approaches for dealing with diagrammatic complexity [44], including text-based modeling [2,45] or various forms of intelligent assistance [49] is high in our future work agenda. Potential may also exist in using our approach to address AI safety and explainability, via recognizing that the behavior of an RL agent that is trained exclusively based on an iStar-RL model is both constrained and explainable by information contained in that model. Finally, adapting our implementation to support multi-objective RL [29] is a logical step toward fully exploiting iStar-RL's ability to represent complex structures of quality criteria.

References

1. Open AI Gym (2022), https://github.com/openai/gym
2. Abdelzad, V., Amyot, D., Alwidian, S., Lethbridge, T.: A Textual Syntax with Tool Support for the Goal-Oriented Requirement Language. In: Proceedings of the 8th International i* Workshop (iStar 2015) (2015), https://ceur-ws.org/Vol-1402/paper6.pdf
3. Ahmad, K., Bano, M., Abdelrazek, M., Arora, C., Grundy, J.: What's up with Requirements Engineering for Artificial Intelligence Systems? In: Proceedigns of the 29th IEEE International Requirements Engineering Conference (RE'21). pp. 1–12 (2021). https://doi.org/10.1109/RE51729.2021.00008
4. 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)
5. Anderson, R.N., Boulanger, A., Powell, W.B., Scott, W.: Adaptive Stochastic Control for the Smart Grid. Proceedings of the IEEE **99**(6), 1098–1115 (2011). https://doi.org/10.1109/JPROC.2011.2109671
6. 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. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2897053.2897054, https://doi.org/10.1145/2897053.2897054
7. Beck, D., Lakemeyer, G.: Reinforcement learning for Golog programs with first-order state-abstraction. Logic Journal of the IGPL **20**(5), 909–942 (2012), https://doi.org/10.1093/jigpal/jzs011
8. Bencomo, N., Belaggoun, A.: Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks. In: Doerr, J., Opdahl, A.L. (eds.) Requirements Engineering: Foundation for Software Quality (REFSQ'13). pp. 221–236. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-37422-7_16
9. Berry, D.M.: Requirements Engineering for Artificial Intelligence: What Is a Requirements Specification for an Artificial Intelligence? In: Gervasi, V., Vogelsang, A. (eds.) Proceedings of the 28th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2022). pp. 19–25. Springer International Publishing, Cham (2022), https://doi.org/10.1007/978-3-030-98464-9_2
10. Bork, D., Ali, S.J., Roelens, B.: Conceptual modeling and artificial intelligence: A systematic mapping study. The Computing Research Repository (CoRR) **abs/2303.0** (2023). https://doi.org/10.48550/arXiv.2303.06758
11. Bork, D., Fettke, P., Maass, W., Reimer, U., Schuetz, C.G., Tropmann-Frick, M., Yu, E.S.: 1st Workshop on Conceptual Modeling Meets Artificial Intelligence and Data-Driven Decision Making (CMAI 2020). In: Grossmann, G., Ram, S. (eds.) Advances in Conceptual Modeling. ER 2020 Workshops CMAI, CMLS, CMOMM4FAIR, CoMoNoS, EmpER. Vienna, Austria (2020), https://doi.org/10.1007/978-3-030-65847-2
12. 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), https://dl.acm.org/doi/10.5555/647288.721273

13. Brunotte, W., Chazette, L., Klös, V., Speith, T.: Quo Vadis, Explainability? – A Research Roadmap for Explainability Engineering. In: Gervasi, V., Vogelsang, A. (eds.) Proceedings of the 28th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'22). pp. 26–32. Springer International Publishing, Cham (2022), https://doi.org/10.1007/978-3-030-98464-9_3
14. Chazette, L., Brunotte, W., Speith, T.: Exploring Explainability: A Definition, a Model, and a Knowledge Catalogue. In: Proceedings fo the 29th IEEE International Requirements Engineering Conference (RE'21). pp. 197–208 (2021). https://doi.org/10.1109/RE51729.2021.00025
15. Chazette, L., Schneider, K.: Explainability as a non-functional requirement: challenges and recommendations. Requirements Engineering **25**(4), 493–514 (2020). https://doi.org/10.1007/s00766-020-00333-1
16. Cognini, R., Corradini, F., Gnesi, S., Polini, A., Re, B.: Business process flexibility - a systematic literature review with a software systems perspective. Information Systems Frontiers **20**(2), 343–371 (2018). https://doi.org/10.1007/s10796-016-9678-2
17. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 Language Guide. The Computing Research Repository (CoRR) **abs/1605.0** (2016), http://arxiv.org/abs/1605.07767
18. Damiani, E., Frati, F.: Towards Conceptual Models for Machine Learning Computations. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) Conceptual Modeling. pp. 3–9. Springer International Publishing, Cham (2018)
19. 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 2018). pp. 39–46 (2018). https://doi.org/10.1109/AIRE.2018.00012
20. 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). https://doi.org/10.1007/s10515-019-00255-5
21. Durán, F., Rocha, C., Salaün, G.: Stochastic analysis of BPMN with time in rewriting logic. Science of Computer Programming **168**, 1–17 (2018). https://doi.org/10.1016/j.scico.2018.08.007
22. 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'19). pp. 89–99 (may 2019). https://doi.org/10.1109/SEAMS.2019.00020
23. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. In: Spaccapietra, S., March, S., Aberer, K. (eds.) Journal on Data Semantics I, pp. 1–20. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39733-5_1
24. Goldsby, H.J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Hughes, D.: Goal-Based Modeling of Dynamically Adaptive System Requirements. In: 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008). pp. 36–45 (2008). https://doi.org/10.1109/ECBS.2008.22

25. Gonçalves, E., Araujo, J., Castro, J.: iStar4RationalAgents: Modeling Requirements of Multi-agent Systems with Rational Agents. In: Laender, A.H.F., Pernici, B., Lim, E.P., de Oliveira, J.P.M. (eds.) Proceedings of the 38th International Conference on Conceptual Modeling (ER 2019). pp. 558–566. Springer International Publishing, Cham (2019), https://doi.org/10.1007/978-3-030-33223-5_46
26. Gottesman, O., Johansson, F., Komorowski, M., Faisal, A., Sontag, D., Doshi-Velez, F., Celi, L.A.: Guidelines for reinforcement learning in healthcare. Nature Medicine **25**(1), 16–18 (2019). https://doi.org/10.1038/s41591-018-0310-5
27. Habibullah, K.M., Horkoff, J.: Non-functional Requirements for Machine Learning: Understanding Current Use and Challenges in Industry. In: Proceedings of the 29th IEEE International Requirements Engineering Conference (RE'21). pp. 13–23 (2021). https://doi.org/10.1109/RE51729.2021.00009
28. Hartmann, T., Moawad, A., Schockaert, C., Fouquet, F., Le Traon, Y.: Meta-Modelling Meta-Learning. Proceedings of the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS 2019) pp. 300–305 (2019). https://doi.org/10.1109/MODELS.2019.00014
29. Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L.M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A.A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P., Roijers, D.M.: A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-Agent Systems **36**(1), 26 (2022). https://doi.org/10.1007/s10458-022-09552-y
30. Heaven, W., Letier, E.: Simulating and optimising design decisions in quantitative goal models. In: Proceedings of the 19th IEEE International Requirements Engineering Conference (RE'11). pp. 79–88. Trento, Italy (2011). https://doi.org/10.1109/RE.2011.6051653
31. Herbert, L.T., Hansen, Z.N.L., Jacobsen, P.: SBOAT: A Stochastic BPMN Analysis and Optimisation Tool. In: Karlaftis, M.G., Lagaros, N.D., Papadrakakis, M. (eds.) Proceedings of the 1st International Conference on Engineering and Applied Sciences Optimization (OPT-i). pp. 1136–1152. National Technical University of Athens (2014), http://www.opti2014.org/
32. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems. In: Hermanns, H., Palsberg, J. (eds.) Proceedings of the 28th 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. Springer Berlin/Heidelberg (2006), https://doi.org/10.1007/11691372_29
33. Ishikawa, F.: Concepts in Quality Assessment for Machine Learning - From Test Data to Arguments. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) Conceptual Modeling. pp. 536–544. Springer International Publishing, Cham (2018), https://doi.org/10.1007/978-3-030-00847-5_39
34. Kimmig, A., Demoen, B., De Raedt, L., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming **11**(2-3), 235–262 (2011). https://doi.org/10.1017/S1471068410000566
35. Kusmenko, E., Nickels, S., Pavlitskaya, S., Rumpe, B., Timmermanns, T.: Modeling and Training of Neural Processing Systems. In: Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS 2019). pp. 283–293 (2019). https://doi.org/10.1109/MODELS.2019.00012

36. 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 (nov 2004), https://doi.org/10.1145/1041685.1029905
37. Liaskos, S.: Tool support for modeling and reasoning with decision theoretic goal models. CEUR Workshop Proceedings **3618** (2023), https://ceur-ws.org/Vol-3618/pd_paper_2.pdf
38. Liaskos, S., Golipour, R.: Tool and reproducibility package for: Model-driven design and generation of training simulators for reinforcement learning (2024), https://github.com/cmg-york/RLGen
39. 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), https://doi.org/10.1016/j.is.2012.05.006
40. Liaskos, S., Khan, S.M., Mylopoulos, J.: Modeling and reasoning about uncertainty in goal models: a decision-theoretic approach. Software & Systems Modeling **21**, 1–24 (2022). https://doi.org/10.1007/s10270-021-00968-w
41. 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). https://doi.org/10.1007/978-3-642-41924-9_3
42. 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 (2009). https://doi.org/10.1109/ASE.2009.91
43. 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). https://doi.org/10.1109/RE.2010.26
44. Lima, P., Vilela, J., Gonçalves, E.J.T., Pimentel, J., Holanda, A., de Castro, J.B., Alencar, F.M.R., Lencastre, M.: Scalability of istar: a systematic mapping study. In: Workshop em Engenharia de Requisitos (WER 2016) (2016), https://api.semanticscholar.org/CorpusID:59248836
45. Liu, W., Wang, Y., Zhou, Q., Li, T.: Graphical modeling vs. textual modeling: An experimental comparison based on istar models. In: Proceedings of the 45th IEEE Annual Computers, Software, and Applications Conference (COMPSAC 2021). pp. 844–853 (2021). https://doi.org/10.1109/COMPSAC51774.2021.00117
46. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: Proceedings of the 33rd International Conference on Machine Learning (ICML'16). pp. 1928–1937. JMLR.org (2016), https://dl.acm.org/doi/10.5555/3045390.3045594
47. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
48. Morandini, M., Penserini, L., Perini, A.: Towards goal-oriented development of self-adaptive systems. In: Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 9–16. SEAMS

'08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1370018.1370021
49. Mussbacher, G., Combemale, B., Kienzle, J., Abrahão, S., Ali, H., Bencomo, N., Búr, M., Burgueño, L., Engels, G., Jeanjean, P., Jézéquel, J.M., Kühn, T., Mosser, S., Sahraoui, H., Syriani, E., Varró, D., Weyssow, M.: Opportunities in intelligent modeling assistance. Software and Systems Modeling **19**(5), 1045–1053 (2020). https://doi.org/10.1007/s10270-020-00814-5
50. Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Exploring Alternatives During Requirements Analysis. IEEE Software **18**(1), 92–96 (2001), http://dx.doi.org/10.1109/52.903174
51. Nalchigar, S., Yu, E.: Business-driven data analytics: A conceptual modeling framework. Data & Knowledge Engineering **117**, 359–372 (2018), https://doi.org/10.1016/j.datak.2018.04.006
52. Nalchigar, S., Yu, E., Keshavjee, K.: Modeling machine learning requirements from three perspectives: a case report from the healthcare domain. Requirements Engineering **26**(2), 237–254 (2021). https://doi.org/10.1007/s00766-020-00343-z
53. Nguyen, C.M., Sebastiani, R., Giorgini, P., Mylopoulos, J.: Multi-objective reasoning with constrained goal models. Requirements Engineering **23**(2), 189–225 (2018). https://doi.org/10.1007/s00766-016-0263-5
54. Object Management Group: Business Process Model And Notation (v2.0). Tech. rep. (2011), https://www.omg.org/spec/BPMN/2.0.2/PDF
55. Pei, Z., Liu, L., Wang, C., Wang, J.: Requirements Engineering for Machine Learning: A Review and Reflection. In: Proceedings of the 30th IEEE International Requirements Engineering Conference Workshops (REW 2022). pp. 166–175 (2022). https://doi.org/10.1109/REW56159.2022.00039
56. Pérez-Soler, S., Guerra, E., de Lara, J.: Model-Driven Chatbot Development. In: Dobbie, G., Frank, U., Kappel, G., Liddle, S.W., Mayr, H.C. (eds.) Proceedigns of the 39th International Conference on Conceptual Modeling (ER 2020). pp. 207–222. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-62522-1_15
57. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021), http://jmlr.org/papers/v22/20-1364.html
58. Rao, A., Jelvis, T.: Foundations of Reinforcement Learning with Applications in Finance. Chapman and Hall/CRC (2022)
59. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
60. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. The Computing Research Repository (CoRR) (2017). https://doi.org/10.48550/arxiv.1707.06347
61. Soutchanski, M.: High-Level Robot Programming in Dynamic and Incompletely Known Environments. Ph.D. thesis, Department of Computer Science, University of Toronto (2003)
62. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press (2018)
63. Vogelsang, A., Borg, M.: Requirements Engineering for Machine Learning: Perspectives from Data Scientists. In: Proceedigns of the 6th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE 2019). pp. 245–251 (2019). https://doi.org/10.1109/REW.2019.00050

64. Wei, H., Zheng, G., Yao, H., Li, Z.: IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18). pp. 2496–2505. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3219819.3220096
65. Yang, W.C., Marra, G., Rens, G., De Raedt, L.: Safe Reinforcement Learning via Probabilistic Logic Shields. In: Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI'23), pp. 5739–5749 (2023). https://doi.org/10.24963/ijcai.2023/637
66. Yohannis, A., Kolovos, D.: Towards model-based bias mitigation in machine learning. Proceedings of the 25th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2022) pp. 143–153 (2022). https://doi.org/10.1145/3550355.3552401