


Towards an iStar 2.0 Extension for Analyzing Goal Variability

Sotirios Liaskos 

Shakil M. Khan 

John Mylopoulos 

Abstract Goal models of the i^* family have been shown to be suitable for concisely representing and analyzing goal variability. In standard i^* , goal variability emerges due to the presence of OR-refinements, which describe alternative ways by which stakeholder goals can be fulfilled. On closer inspection, however, variability exists in goal models beyond what can be represented by OR-refinements. Firstly, given one set of tasks that fulfill the root goals, there are many alternative orderings by which those tasks can be performed. Secondly, tasks can have many alternative outcomes. In the past, we have proposed various approaches for analyzing these two types of variability, each accompanied by an extension to the i^* notation that allows translation of the visual goal model into a formal specification suitable for automated identification of goal-fulfilling alternatives. In this chapter, we propose a consolidation of these extensions into *iStar T*, a unified extension to the current *iStar 2.0* modeling language. *iStar T* proposes a minimum set of added modeling constructs and a core set of rules that allow translation of models into specifications usable by different reasoning systems. We describe the extensions and, as examples, we sketch the rules to translate into Hierarchical Task Network and Golog specifications and the kinds of automated analysis that can be done with the result.

1 Introduction

Since its introduction in the mid-90's, the i^* modeling language [27] has had a substantial influence in the study of the engineering of software-intensive systems.

Sotirios Liaskos

School of IT, York University, Toronto, ON, Canada e-mail: liaskos@yorku.ca

Shakil M. Khan

University of Regina, Regina, SK, Canada e-mail: shakil.khan@uregina.ca

John Mylopoulos

University of Toronto, Toronto, ON, Canada e-mail: jm@cs.toronto.edu

Its ability to combine modeling of intra-agent intentional structures with inter-agent social dependency ones in an accessible semi-formal manner has found a large variety of applications in a broad range of areas. It has further been the central reference point of a large and diverse ecosystem of related languages, tools, methods, and analysis techniques. One of the many attractive aspects of i^* and the languages it has inspired is their ability to model *goal variability*, that is, alternative ways by which actor goals can be fulfilled [22]. In i^* , goal variability is modeled both concisely, allowing encoding of large numbers of alternatives within relatively small and readable decomposition structures, and in a way that is amenable to useful analysis by allowing characterization of the alternatives subject to criteria.

In this chapter we discuss the notion of goal variability and review the ways by which it occurs in goal models. We then propose *iStar T*, an extension to i^* 's latest standard *iStar 2.0* [6], aimed at facilitating representation of the variability that emerges along temporal and causal dimensions of goal models. The *iStar T* dialect combines and consolidates disparate extensions we have introduced over the past years for different variability modeling and analysis purposes. It introduces a minimum set of core constructs that has proved useful for a variety of variability modeling applications and is amenable to translation into various formal specifications to enable automated reasoning and analysis. As examples, we sketch how *iStar T* can be translated to Hierarchical Task Networks (HTN) [23] and Golog [16] specifications and how the latter can be used for automated reasoning about goal variability. *iStar T* can hence be the basis for developing core modeling and translation tools that can be extended into more specialized languages and toolsets.

We organize the chapter as follows. In Section 2 we describe how variability is encoded in i^* -type goal models. In Section 3 we present the *iStar T* extension. Then in Sections 4 and 5 we outline how *iStar T* diagrams can be translated to HTN and Golog specifications and allow for useful automated reasoning. We conclude with pointers to related efforts and discuss opportunities for future work in Section 6.

2 Variability in i^* models

Let us demonstrate how variability emerges in goal models through an example. In Figure 1, an *iStar 2.0* [6] (our reference i^* language specification henceforth) hybrid strategic dependency (SD) - strategic rationale (SR) diagram can be viewed depicting the intentional structure of an interaction between a merchant and a customer. The latter depends on the former to receive a quote for a product of interest, and, following order placement and payment completion, to receive the product shipment. The merchant depends on the buyer for receiving order and payment. Both actors want to fulfill their top-level goals, *Have Order Fulfilled* and *Have Product Acquired*.

We use the term *goal variability* to express both the notion that there are alternative ways by which such root goals can be fulfilled [22] and to refer to the set of these alternatives. Such variability is primarily represented through OR-refinements of goals and tasks. In the example, OR-refinements are used to represent different

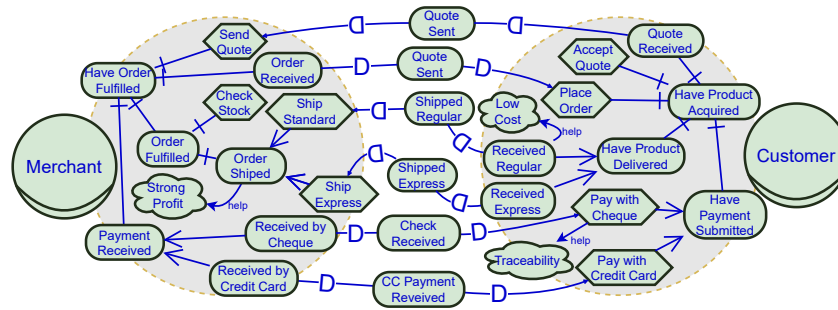


Fig. 1 A merchant-customer iStar 2.0 diagram.

speeds by which *Merchant's Order Shipped* can be fulfilled (express and regular), as well as different ways by which the *Customer can Have Payment Submitted*.

The meaning of OR-refinement-based variability can vary depending on the content of the participating goals and the aspect in which the refinement children differ. For example, variability in methods of paying for the order can be seen as *instrument* variability whereas variability in delivery times is *temporal* variability. Additional examples of kinds of variability that an OR-refinement represents includes *agentive* (different actors responsible for fulfilling the goal), *locational* (different locations at which a goal can be fulfilled) or *extent* (different degrees by which goal-fulfillment activities can be performed - e.g., intensity, level, amount etc.). These categories, inspired by case frame literature [8], have been proposed to be used as tools for eliciting OR-refinements [20, 25]. Moreover, the presence of contribution links to quality goals allows representation of the rationale for certain choices among OR-refinement sibling goals, or conversely, guide the choice when a preference of one or more quality goals over others is given [21, 22].

However, variability in *i**-type intentional structures may exist beyond what an OR-refinement can represent. Two additional types of variability that are of interest here are *ordering variability* and *outcome variability*. Ordering variability refers to the multiplicity by which sets of actions to fulfill goals can be ordered. In our example, a product can be shipped before or after a payment has been received. Outcome variability, on the other hand, is based on the acknowledgment that, in reality, tasks do not bring about the same outcome every time they are attempted. Rather, attempt of a task leads to the occurrence of one out of multiple outcomes. For example, *Ship Express* may fail due to a mishap in the shipping process or either of the outcomes of a task (not seen in the figure) *Choose Payment Option*, e.g., cash or credit card, may occur. Importantly, different outcomes may influence how variability is handled elsewhere in the goal model. For example, if *Customer* chooses to pay by credit card, tasks such as *Apply Discount* may become disabled.

To represent and reason with ordering and outcome variability we propose *iStar T*, an extension to the standard *iStar* notation with constructs that allow us to represent, firstly, constraints to the order by which tasks can be executed, and, secondly, alternative outcomes of tasks. We discuss these extensions next.

Causal Constraints and Non-Determinism. Tasks and effects are connected with each other using effect links which represent causal constraints. Effect links towards single effects convey that attempt of the task from where the link originates deterministically (i.e., with no uncertainty) makes the propositions listed in the effect true. In Figure 2, *Customer* task *Pay with Credit Card* points to effect *ccPayReceived*, meaning that attempt of the task makes the proposition in the effect true.

When we want to represent that performance of a task has multiple possible effects, we use effect groups. Effect groups are represented through a filled circular element on which the set of possible effects are connected through a plain effect group line. When the group is *activated*, exactly one of the effects will non-deterministically occur. When available, the probability of each effect occurring upon effect group activation can be represented through a numeric label on the corresponding effect group link; probabilities in each effect group must add up to 1. When an effect link connects a task to an effect group, this conveys that when the task is attempted, the corresponding effect group is activated, which, in turn, means that one of the constituent effects of the group will occur. In our example, the *Customer* task *Accept Quote* is connected to an effect group that includes two possible outcomes: *quoteAcceptable*, and *quoteUnacceptable*. Upon performance of the task, exactly one of these two propositions will become true with probabilities 0.6 and 0.4, respectively.

Moreover, individual effects of an effect group may be deemed either *success* or *failure* effects, marked with a blue/solid or a red/dashed outline, respectively. The former signify that, when they occur, the task that points to the effect group is assumed successfully performed. The latter signify that the corresponding task attempt has instead failed. In our *Accept Quote* example, of the two effects it may have, only one is assumed to mark successful performance of the task (*quoteAcceptable*). However, a task is considered to be *attempted*, if any of its effects occurs.

A failed task does not contribute to the fulfillment of its parent goal, though it marks its attempt. Specifically, let a goal g be AND- (respectively, OR-) refined into tasks t_1, t_2, \dots and goals g_1, g_2, \dots . Goal g is satisfied iff all (respectively, at least one) of tasks t_i and goals g_i are (resp., is) successfully performed (rather than merely attempted) and (resp., or), in the case of goals, satisfied. Likewise, a goal g is attempted iff at least one of the tasks t_i and goals g_i is successfully attempted. By applying these rules while recursively traversing the AND/OR refinement tree, we can associate any goal or task element in the diagram with two Boolean formulae grounded on domain propositions, a *satisfaction formula* and an *attempt formula*. Then, the goal/task is satisfied/performed (resp. attempted) iff the satisfaction formula (resp. attempt formula) holds. Given a value configuration of all propositions mentioned in effect elements, the satisfaction and attempt status of any goal in the diagram, including those of root goals can be decided.

Temporal Constraints. Precedence and negative precedence links can be drawn from a condition element, a task, or a goal, and target a task, a goal, or an effect within an effect group. A precedence link from a condition element, task or goal to a task means, respectively, that the condition element has to be true, the task has to be successfully performed, or the goal has to be satisfied before the task can be attempted. A precedence link towards a goal can be seen as a shorthand to drawing

separate precedence links from the same origin to each of the tasks in the model that are descendants of the goal. When a precedence link targets an effect within an effect group, the effect cannot be brought about unless the origin is respectively true/satisfied/performed. A negative precedence link from a condition element, task or goal targeting an effect in effect group, a task, or a goal, means, respectively, that the targeted effect cannot be brought about, the targeted task cannot be attempted and none or the leaf-level successor tasks of the goal can be attempted, if, respectively for the origin possibilities, the condition element is true, the origin task has been successfully performed, or the origin goal has been *attempted* (rather than satisfied).

In Figure 2, the precedence between goal *Quote Received* and task *Accept Quote* implies that the goal must be deemed satisfied before the task can be attempted. This will be true if Merchant’s task *Send Quote* is successfully performed, meaning that the *quoteSent* outcome occurs. If *quoteLost* occurs instead, *Quote Received* is not considered satisfied and, as such, *Accept Quote* cannot be attempted.

Effect Histories and Initializations. As actors act in the domain to satisfy their goals, tasks are assumed to be attempted bringing about deterministic or non-deterministic effects, which, in turn, decide if the tasks and their parent goals can be assumed performed and satisfied, enabling or disabling the possibility of attempting other tasks or goals in the next step. An *effect (occurrence) history* is a constraint-satisfying sequence of effects resulting from a corresponding sequence of tasks (the *task history*) from a state in which no task has been attempted. A *goal satisfying history* with respect to a goal is an effect history leading to a state in which the goal has been fulfilled. Such a history describes one way to bind goal variability, where specific OR-refinement alternatives are chosen, a specific execution order is attempted, and specific outcomes have come about.

Further, initializations contain a vocabulary of propositions that must be set to true or false prior to the calculation of histories. Their content is of the form $\{p_1\}$, $\{p_2\}$, ..., where p_i are propositional atoms to be initialized.

Contribution Links. In *iStar T* contribution links can be drawn from effects to quality goals, signifying that it is the outcomes of tasks that contribute to qualities, rather than their mere attempt. For example, that *Ship Express* has been attempted does not offer any information with regards to whether quality *Happy Customer* is positively affected. The latter depends on the outcome of the attempt. For example, if the shipment is lost, that has a very negative contribution to the quality goal in question despite the intent. Contributions can be seen as activated and deactivated by task attempts taking place throughout a satisfying history, through enabling propositions contained in the task’s effects or arbitrary conditions. Consequently, a goal satisfying history is associated with a trajectory of activations and deactivations of contribution links that target each quality goal.

Structural Restrictions. We finally note that a number of structural constraints and scope restrictions are introduced in the current version of *iStar T* to simplify the formalization and analysis process. The most important are (i) that every task must be connected through an effect link to an effect or effect group, (ii) that tasks are not further refined into other tasks or goals, and (iii) that goals must either be refined or delegated; no goal can exist at the leaf level. Thus, meaningful *iStar T*

diagrams consist of one or more goal refinement trees, rooted at goals which are progressively refined and/or delegated into lower-level subgoals and eventually tasks, each of which is associated with an effect or an effect group.

4 Formalizing *iStar T*

One of the purposes for defining *iStar T* is to allow automated reasoning about goal variability via translation into suitable formal specification languages. The authors have proposed various ways in the past for performing such translations based on different extensions to *i** catering for different variability analysis purposes [18, 19, 21]. Our goal is to distill *baseline translation rules* which can be extended to fulfill specific needs. The baseline rules depend on the target specification language. In this chapter, we showcase development of rules for two targets: Hierarchical Task Networks (HTNs) [23] and Golog [16]. Both target languages have been shown to be useful for a variety of tasks including identification of optimal solutions based on different criteria [19, 21], behavioral customization of software [18], and development of simulations for reinforcement learning [17].

Our presentation is informal and example-based and aimed at showcasing the principles behind the translation rules and their capabilities in terms of the analysis they support. An authoritative report with a formal presentation and analysis of the core translation rules is upcoming. For better comprehension of the presentation that follows, the reader may find useful to refer to the examples of Figure 3.

4.1 HTN Planning

We start by showing how the extended goal models can be translated to HTN specifications, which in turn can be used to identify goal satisfying histories.

HTN planners receive a *domain specification* and a *problem specification* as input and generate a solution to the latter as output. The domain specification consists of a set of domain predicates $v \in V$ describing properties of objects $b \in B$ that are of interest in the domain, a set of *operators* $o \in O$, representing actions in the domain, and a set of *methods* $\in M$ describing ways by which higher level planning objectives can be decomposed into lower level operators or other methods.

Domain predicates, such as `has(customer, quote)`, are used to describe the state of the domain as agents perform actions described by operators. Moreover, *axioms* show that the predicate that is the *head* of the axiom is true if the expression in the *tail* of the axiom is satisfied. Thus, the following axiom says that 0-ary predicate `orderShipped` is true if any of the propositions in the tail is satisfied:

```
(:- (orderShipped) ; head
    (orderShippedByAir ∨ orderShippedByLand)) ; tail
```

<i>iStar T</i>	<i>HTN</i>	<i>Golog</i>
	<pre> (:method productAcquired () (:unordered (productDelivered) (paymentSubmitted))) (:- productAcquiredSat (and (productDeliveredSat) (paymentSubmittedSat))) (:- productAcquiredAtt (or (productDeliveredAtt) (paymentSubmittedAtt))) </pre>	<pre> proc(productAcquired, (productDelivered:paymentSubmitted) # (paymentSubmitted:productDelivered)). productAcquiredSat(S) :- productDeliveredSat(S), % and paymentSubmittedSat(S). productAcquiredAtt(S) :- productDeliveredAtt(S); % or paymentSubmittedAtt(S). </pre>
	<pre> (:method orderShipped () ((shipExpress))) (:method orderShipped () ((shipStandard))) (:- orderShippedSat (or (shipExpressSat) (shipStandardSat))) (:- orderShippedAtt (or (shipExpressAtt) (shipStandardAtt))) </pre>	<pre> proc(orderShipped, shipExpress # shipStandard). % pick orderShippedSat(S) :- shipExpressSat(S); shipStandardSat(S). orderShippedAtt(S) :- shipExpressAtt(S); shipStandardAtt(S). </pre>
	<pre> (:operator (!quoteSent) (...) ; precondition () ; delete list (quoteSent)); add list (quoteLost) (:method sendQuote () ((!quoteSent))) (:method sendQuote () ((!quoteLost))) (:- sendQuoteSat ((quoteSent))) (:- sendQuoteAtt (or (quoteSent) (quoteLost))) </pre>	<pre> proc(sendQuote, quoteSent # quoteLost). poss(quoteSent) :- ... poss(quoteLost) :- ... quoteSent(do(A,S)) :- quoteSent(S); A = quoteSent. quoteLost(do(A,S)) :- quoteLost(S); A = quoteLost. sendQuoteSat(S) :- quoteSent(S). sendQuoteAtt(S) :- quoteSent(S);quoteLost(S). </pre>
	<pre> (:method paywithCreditCard () ((!ccPayReveivedNat) (q_contr traceability plus))) (:operator (!ccPayReceivedNat) () (ccPayReceived)) ... (:method (q_contr ?quality help) (sat ?goal FS); if ((!!fullSat ?goal)); tasks (); else ((!partSat ?goal))) </pre>	<pre> proc(paywithCreditCard, ccPayReceived). poss(ccPayReceived) :- ... ccPayReceived(do(A,S)) :- ccPayReceived(S); A = ccPayReceived. sat(traceability,part,do(A,S)) :- sat(traceability,part,S); A = ccPayReceived; A = ... % other help contributors </pre>

Fig. 3 Translation rules by example.

Operators represent the transitioning of the domain from one state to another, through changes in the truth values of domain predicates. Each operator o is associated with an *add list* and a *delete list*, which list predicates that turn true and false, respectively, upon application of the operator. Operators also have a *precondition* formula, constructed with elements from V , which denote that the operator cannot be applied at a given state, unless the precondition is satisfied at that state. The following is an example of an operator:

```

(:operator (!payWithCreditCard) ; Operator name
  (hasCC  $\wedge$  ( $\neg$ priceExceedsLimit  $\vee$  limitSuspended)) ; precondition
  ((paymentPending)) ; Delete list
  ((ccPayReceived))) ; Add list

```


Methods are used to guide the planner into searching for sequences of actions that are plausible in the domain of interest. An example of a method is as follows:

```
(:method (orderfulfilled)          ; method name
  (orderReceived)                 ; method precondition
  (:unordered (!checkStock) (orderShipped) )) ; task list
```

In the above, the method `orderfulfilled`, which can be utilized if `orderReceived` is true, shows how the problem of having the order fulfilled decomposes into the application of operator `!checkStock` and application of another method called `orderShipped`; in no specific order. There may be many alternative methods named `orderfulfilled` for the planner to choose from when constructing a plan.

4.1.1 Specification Generation

We now turn to how we translate *iStar T* models into HTN domain specifications.

Goals and tasks. Each goal and each task in the goal diagram is, firstly, mapped into two propositions (0-ary predicates) in the HTN domain: one that signifies its successful attainment (*attainment proposition*) and one that signifies its attempt (*attempt proposition*). Secondly, each goal and task is mapped to a method name.

Thirdly, we look at how the goal or task is refined. If a goal is AND-decomposed into n other goals and/or tasks (sub-elements) a *refinement method* is introduced, named after the goal being decomposed and containing an unordered list of method or operator names corresponding to all n sub-elements. When a goal is OR-decomposed into the n sub-elements, n refinement methods are introduced instead, each named after the goal being decomposed. The task list of each of the n methods contains one method or operator name corresponding to each of the sub-elements of the OR-decomposition. Further, we treat the dependency as a one-subgoal OR-refinement of the depender goal to the dependee element.

When the element is a task it is associated with an effect or an effect group. The translation then treats this association as an OR-decomposition and introduces as many methods as the effects in the group (one in the case of a deterministic effect), whose name is the method name associated with the task and the action list in each method has the name of the effect operator associated with the effect (more below).

Fourthly, an *attainment axiom* is introduced associating the attainment proposition of the parent goal with the conjunction (AND-decompositions) or disjunction (OR-decompositions) of attainment propositions of the sub-elements. Fifthly, an *attempt axiom* is introduced that associates the attempt proposition of the parent goal with the disjunction of attempt propositions of the sub-elements independent on whether it is an AND- or OR-decomposition.

Effects. Each effect element e is translated into a distinct *effect operator*. The add list of the effect operator contains the domain predicates listed in the effect shape and the delete list is empty. Calculation of the precondition of the operator is as follows. Consider first the set A_e of all the goal and task elements that are ancestors to effect element e , i.e., that exist in the path between the effect and the root goal. Consider next the sources of all precedence links that target the elements in A_e . These

sources can be goals, tasks, or conditions. Construct then the conjunction ϕ_{pre} of the attainment propositions (goals and tasks) and condition formulae (condition elements) that correspond to these sources. Consider next all the sources of the negative precedence links that target the elements in A_e , which are again goals, tasks or conditions. Construct the disjunction ϕ_{npr} of the *attempt* propositions (goal and task elements) and condition formulae (condition elements) that correspond to these sources. Assign then $\phi_{pre} \wedge \neg\phi_{npr}$ as the precondition to the effect operator.

Qualities. Each quality q is translated into a domain object q and two predicates $\text{sat}(q, l)$ and $\text{den}(q, l)$, which carry the level l of satisfaction and denial of quality q , respectively. Consider then a contribution link from effect element e towards a quality q . The effect element is either associated with a task (deterministic case) or is part of an effect group (non-deterministic case). The contribution link is translated into an invocation of a reserved (`quality_contr ?cont ?dest`) method, where `?cont` describes the kind of contribution and `?dest` the destination. This is done by simply adding the method name `quality_contr` in the task list of the method that is associated with the task that brings about the effect. For example, if task t sends an effect link to an effect group e_1, e_2, e_3 , and, say, e_2 sends a *helps* contribution link to quality q , then the method that decomposes t into e_2 has a task list that includes both the effect operator for e_2 and method (`quality_contr help q`).

The `quality_contr` method, together with associated helper reserved operators, is responsible for affecting the $\text{sat}(q, l)$ and $\text{den}(q, l)$ predicates based, e.g., on the label of the contribution link and the existing satisfaction and values. The exact structure of these methods and operators depends on the framework adopted for representing and aggregating contribution and satisfaction/denial values. For example, established label propagation rules [12] can be applied for updating the value of $\text{sat}(q, l)$ and $\text{den}(q, l)$ every time `quality_contr` is invoked.

4.2 Golog

We now turn to how *iStar T* goal models can be translated to specifications of the Golog family. Golog [16] is a high-level agent programming language based on the situation calculus [15], a language for modeling dynamic domains. Golog has been the basis of many extensions including ConGolog [14] which allows for concurrency, interrupts and exogenous actions, Indigolog [7] which allows for interleaved action, sensing and planning, and DT-Golog [3] that allows for reasoning with probabilistic actions. In addition, simulating dynamic domains encoded in the underlying situation calculus can be used for model-free reinforcement learning [17]. Hence, having a core set of rules for translating *iStar T* models to Golog facilitates accessing diverse analysis, reasoning, and execution tools.

Core concepts of Golog are *fluents*, *actions*, and *situations*. Fluents are used to describe what is true in the domain at a given situation. They are represented through n-ary predicates with a situation term as their last argument, as in `hasRead(quote, customer, s)`. A situation is a first-order term that represents a se-

quence of actions from an initial situation s_0 where no action has been performed. The function symbol $do(a, s)$ is used to denote the situation which results from performing action a in situation s . Situations emerge from nested application of the function as in: $do(a_n, do(a_{n-1}, \dots, do(a_1, s_0) \dots))$. Finally, a special predicate $poss(a, s)$ is used to state that action a is executable in situation s .

To describe a domain a set of axioms are defined over the above constructs. The most important are *action precondition axioms* and *successor state axioms*. The former are defined for each action a and describe the conditions under which actions can be performed. For example, the following axiom says that for customer c to pay a merchant m by cheque in situation s the former needs to have a checking account and the latter needs to be accepting of cheques in the same situation:

$$poss(\text{payWithCheque}(c, m), s) \leftrightarrow \text{hasCheckingAccount}(c, s) \wedge \text{acceptsCheques}(m, s)$$

Successor state axioms are defined for each fluent and describe how the fluent's truth value changes due to the performance of tasks. For example, the fluent $quoteSent(s)$ holds in a given situation if it was already true in the previous situation or the latest action has made it true:

$$quoteSent(do(a, s)) \leftrightarrow quoteSent(s) \vee (a = \text{sendQuote})$$

Golog allows the development of high-level programs that describe agent behavior while abiding by the action theory defined by the axioms. Programming constructs of interest are *sequences* and *nondeterministic choices* of actions which are used in *procedures*. Of the following procedures, `orderFulfilled` contains an action sequence in its body comprising of an action and another procedure to be executed in that order, and `orderShipped` is a non-deterministic choice between two actions:

```
proc(orderFulfilled, checkStock : orderShipped)
proc(orderShipped, shipExpress | shipStandard)
```

4.2.1 Specification Generation

We now sketch the rules for translating *iStar T* models into Golog specifications.

Goals and Tasks. Each goal and task in the model is, firstly, translated into one attainment fluent and one attempt fluent. Secondly, a Golog procedure named after the element in question is added. If the element is an OR-decomposed goal, or a depender goal, the body of the procedure contains a non-deterministic action choice among a set of procedure names referring to the sub-elements or the one dependee element. If the element is a task, the body contains a non-deterministic choice among the effect actions (more below) that correspond to the one or more effects associated with the task. If the element is an AND-decomposed goal, the procedure body contains a non-deterministic choice among a set of action sequences that constitute all possible permutations of the sub-elements, except for those that are impossible due to temporal constraints among the subgoals.

Effects. Each effect is translated into (a) an *effect action*, (b) as many *effect fluents* as the propositions that are contained in the effect. For each of the actions an action precondition axiom is constructed, exactly as in the case of HTN translation: the sources of precedence (respectively, negative precedence) links targeting every

ancestor of the effect are collected and together form a conjunction (resp., disjunction) of the corresponding attainment (resp., attempt) formulae. The conjunction of the two formulae, with the latter (the one collecting constraints from negative precedence links) negated, are then placed as precondition for the introduced action.

Furthermore, each of the introduced fluents effFluent_i relating to the effect corresponds to a successor state axiom of the form:

$$\text{effFluent}_i(\text{do}(a,s)) \leftrightarrow \text{effFluent}_i(s) \vee \\ (a = \text{effAction}_1) \vee (a = \text{effAction}_2) \vee \dots$$

where effAction_i are effect actions corresponding to effects that mention effFluent_i . Finally, the truth values of attainment and attempt fluents associated with goals and tasks are determined, via appropriate axioms, by AND/OR formulae grounded on effect fluents and reflecting the underlying refinement structure.

Qualities. For each quality q two fluents $\text{sat}(q,l,s)$ and $\text{den}(q,l,s)$ are introduced representing the level of satisfaction and denial evidence associated with the quality in situation s . For each of these fluents, a number of successor state axioms is constructed showing how different quality satisfaction values are acquired due to the occurrence of effects that have a contribution to the quality. For example, following again the qualitative two-level satisfaction representation system (partially/fully satisfied/denied) [12] the axiom of the form:

$$\text{sat}(q,\text{full},\text{do}(a,s)) \leftrightarrow \text{sat}(q,\text{full},s) \vee \\ (a = \text{ppEffect}_1) \vee (a = \text{ppEffect}_2) \vee \dots$$

says that quality q is fully satisfied in the next situation if it already were in the previous situation or one of the effects ppEffect_i from where a *make* contribution link originates targeting q has occurred.

5 Automated Identification of Solutions

The specifications resulting from the above translations can be used as a basis for automatically generating goal-satisfying effect histories (solutions) using the corresponding reasoning tools – HTN planners and Golog interpreters. At least three kinds of solutions can be generated: arbitrary solutions, solutions under given quality constraints, and solutions that optimize a preference specification over qualities.

Arbitrary solutions. The tools are useful for identifying any constraint satisfying solution when no other desiderata exist. In the HTN case, given the resulting domain specification, a planner is able to search for plans that satisfy a specific problem specification. The latter consists of a list of initially true predicates and a top-level task to be fulfilled. Hence, by instantiating the *iStar T* model’s initialization into a set of HTN predicates and assigning a root goal of interest to be the top-level task, we can get the HTN planner to generate plans, each constituent operator of which mapping to an effect in a goal satisfying history. In the presence of effect groups, the planner returns non-deterministically chosen scenarios of execution. In the Golog case, the interpreter is capable of running Golog programs relative to an initial situation, outputting a situation (i.e., a sequence of actions) at which the

<i>HTN</i>	<i>Golog</i>
1: (!QUOTESENT)	1: s0
2: (!QUOTEACCEPTABLE)	2: quoteSent
3: (!ORDERRECEIVED)	3: quoteAcceptable
4: (!ITEMINSTOCK)	4: orderReceived
5: (!ARRIVEDINTIME)	5: itemInStock
6: (!CHEQUERECEIVED)	6: arrivedLateEx
7: (!FINAL)	7: paywithCreditCard
8: (!SAT HAPPYCUSTOMER PS)	[sat(happyCustomer, pd, ...),
9: (!SAT STRONGPROFIT PS)	sat(strongProfit, 'ps, ...)]
...	

Fig. 4 Reasoner output format (different solutions).

program terminates. Thus, by setting the fluents corresponding to an instantiation of the *iStar T* model's initialization to hold in the initial situation S_0 we can run the procedure that represents the top-level goal we are interested in generating a solution for. Given the mapping of actions to leaf-level effects, the results represent scenarios of task outcomes.

Solutions under constraints. We often desire to generate solutions under hard constraints, and, specifically, desired quality satisfaction values. For example, we may be interested in solutions which never receive a negative contribution to quality goal *Safety*, or in which, at the end only, goal *Happy Customer* is fully satisfied. To find such solutions relative to a top-level goal g , in both HTN and Golog cases we work as follows. Firstly, we create a formula ϕ_q describing all such constraints. Then in the *iStar T* model we introduce a dummy terminal task t_r and a dummy root goal g_r . Next, we set g and t_r be AND-refinements of g_r and connect them with a precedence link from g to t_r . We then translate as per the corresponding rules and add ϕ_q as a precondition to t_r . We note that there is more nuance than we can cover here: for example universal/existential constraints on the entire history (e.g.: there does not exist a negative contribution to the *Safety* quality) are addressed differently from those that are posed on aggregates of contributions (e.g.: the cumulative contribution to *Low Cost* at the end is above a threshold). A general framework for quality satisfaction analysis in light of time-ordered contributions to qualities would be a useful future extension of *iStar T*.

Optimal solutions. With the appropriate extensions and revisions, the translation rules can also produce specifications that allow identifying solutions that are optimal with respect to an objective function. It has been shown that the Golog specification can be augmented with a relatively simple reward structures to allow decision theoretic optimization using DT-Golog [19]. Further, the Golog programs are particularly useful for performing simulations of the intentional structure, which can be useful in reinforcement learning applications [17]. Moreover, HTN operators can accept simple cost parameters allowing HTN planners to search for minimum-cost plans. Rules can, hence, be defined that map contributions to qualities to such cost values. A complete treatment of how minimum-cost HTN planning can be used for optimizing quality goal satisfaction desiderata is another promising extension of *iStar T*.

Output Format. Sample outputs of the tools can be seen in Figure 4. The action lists correspond to effect histories in Figure 2. In the HTN output, impacts to quality goals are displayed as separate trailing actions. In the Golog output, following solution generation, quality satisfaction fluents that are true in the situation that corresponds to the solution are printed. Post-processing procedures can use the

quality satisfaction aspect of the output as an input to more advanced satisfaction propagation analysis algorithms [12] and visualizations.

6 Concluding Remarks

A great many extensions to i^* have been introduced to make the language usable for specific purposes [13], and many procedures have also been proposed for automated reasoning about goal models via using established reasoning tools and for different purposes, e.g., SMTs [24] and model checkers [9] to name two. Our proposed extension, *iStar T*, attempts to systematize and consolidate extensions that have been introduced for the purpose of supporting analysis of different kinds of goal variability, including ordering and outcome variability. It further complements similar work on goal model formalization (e.g. [4, 10, 26]) in that the set of extensions it introduces is minimal enough to allow formalization into different formal specification languages. In this chapter we showcased HTN and Golog, but similar procedures for formalizing *iStar T* to other languages appear to be possible. This includes different subsets and versions of PDDL [11] for compliant planners as well as specification languages for model-checking [5]. This allows researchers with different analysis needs to have a common starting point. An additional opportunity for future work is a baseline for advanced reasoning with quality goals, including different kinds of preference specifications and a treatment of satisfaction propagation in quality goal graphs within the context of time-ordered actions.

Further, the visual complexity of *iStar T* is deserving of some investigation. The additional elements required for forming a complete model arguably affect the comprehensibility of the resulting diagram. To address this, interactive model slicing can be investigated whereby parts of the diagram (precedence links, effect groups, etc.) are displayed only on demand. Likewise, the definition of separate viewpoints, such as a temporal and/or a causal one, can be explored for separating concerns, as per common practice – e.g., URN [2] introduces two sub-languages, GRL and UCM, for modeling intentional and causal aspects, respectively. Finally, the introduction and use of a textual syntax, in the spirit of TGRL [1], could prove useful for taming complexity, particularly during model development.

References

1. Abdelzad, V., Amyot, D., Alwidian, S., Lethbridge, T.: A Textual Syntax with Tool Support for the Goal-Oriented Requirement Language. In: Proc. of the 8th Intl. i^* Workshop (2015)
2. 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)
3. Boutillier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In: Proc. of the 17th Conference on Artificial Intelligence (AAAI-00). pp. 355–362. AAAI Press, Austin, TX (2000)

4. Bryl, V., Massacci, F., Mylopoulos, J., Zannone, N.: Designing security requirements models through planning. In: Proc. of the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06) (2006). https://doi.org/10.1007/11767138_4
5. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: In Proc. of Int. Conf. on Computer-Aided Verification (CAV 2002). Copenhagen, Denmark (jul 2002)
6. 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>
7. De Giacomo, G., Lespérance, Y., Levesque, H.J., Sardina, S.: IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents, pp. 31–72. Boston, MA (2009)
8. Fillmore, C.J.: The Case for Case. In Bach and Harms (Ed.): *Universals in Linguistic Theory*. New York: Holt, Rinehart, and Winston pp. 1–88 (1968)
9. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P.: Specifying and analyzing early requirements in Tropos. *Requirements Engineering* **9**(2), 132–150 (2004)
10. Gans, G., Jarke, M., Lakemeyer, G., Vits, T.: SNet: A Modeling and Simulation Environment for Agent Networks Based on i* and ConGolog. In: Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering CAiSE'02. pp. 328–343. Toronto, Canada (2002)
11. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL - The Planning Domain Definition Language. Tech. rep., Yale Center for Computational Vision and Control. Technical Report CVC TR-98-003/DCS TR-1165. (1997)
12. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with Goal Models. In: Proc. of the 21st Int. Conf. on Conceptual Modeling (ER'02). pp. 167–181 (2002)
13. Gonçalves, E., Castro, J., Araújo, J., Heineck, T.: A Systematic Literature Review of iStar extensions. *Journal of Systems and Software* **137**, 1–33 (2018)
14. Lespérance, Y., Kelly, T.G., Mylopoulos, J., Yu, E.S.K.: Modeling Dynamic Domains with ConGolog. In: Proc. of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99). Heidelberg, Germany (jun 1999)
15. Levesque, H., Pirri, F., Reiter, R.: Foundations for a Calculus of Situations. *Electronic Transactions on AI (ETAI)* **2**(3–4), 159–178 (1998)
16. 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)
17. Liaskos, S., Khan, S.M., Golipour, R., Mylopoulos, J.: Towards Goal-based Generation of Reinforcement Learning Domain Simulations. In: Proc. of the 15th International i* Workshop. CEUR Workshop Proceedings (2022)
18. Liaskos, S., Khan, S.M., Litoiu, M., Jungblut, M.D., Rogozhkin, V., Mylopoulos, J.: Behavioral adaptation of information systems through goal models. *Inf. Syst. (IS)* **37**(8), 767–783 (2012)
19. 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)
20. Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., Mylopoulos, J.: On Goal-based Variability Acquisition and Analysis. In: Proc. of the 14th IEEE International Requirements Engineering Conference (RE'06). IEEE Computer Society, Minneapolis, Minnesota (sep 2006)
21. Liaskos, S., McIlraith, S., Sohrabi, S., Mylopoulos, J.: Representing and reasoning about preferences in requirements engineering. *Requirements Eng. Journal (REJ)* **16**, 227–249 (2011)
22. Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Exploring Alternatives During Requirements Analysis. *IEEE Software* **18**(1), 92–96 (2001)
23. 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)
24. Nguyen, C.M., Sebastiani, R., Giorgini, P., Mylopoulos, J.: Multi-objective reasoning with constrained goal models. *Requirements Engineering* **23**(2), 189–225 (2018)
25. Rolland, C., Souveyet, C., Achour, C.B.: Guiding Goal Modeling Using Scenarios. *IEEE Transactions on Software Engineering* **24**(12), 1055–1071 (1998)
26. Wang, X., Lespérance, Y.: Agent-oriented requirements engineering using ConGolog and i*. In: In Bi-Conference Workshop at Agents 2001 and CAiSE'01 (AOIS-2001). (2001)
27. Yu, E.S.K.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proc. of the 3rd IEEE International Symposium on Requirements Engineering (RE'97). pp. 226–235. Annapolis, MD (1997)