VNU Journal of Science: Comp. Science & Com. Eng. Vol. VV, No. NN (2026) 1–32

VNU Journal of Science:
Computer Science and Communication Engineering

Journal homepage: http://www.jcsce.vnu.edu.vn/index.php/jcsce

Original Article

A Formal Semantic Framework for Goal-Oriented Requirements Modeling

Hoang-Viet Trinh La, Duc-Hanh Dang*

Faculty of Information Technology, VNU University of Engineering and Technology, Hanoi, Viet Nam

Received 29 Apr 2026;
Revised 12 Nov 2026; Accepted 16 Nov 2026

Abstract: Goal-oriented languages such as iStar capture stakeholder goals and dependencies but do not natively specify executable state transitions or trace-level temporal obligations. This paper proposes gStar, an extension of iStar with explicit operations, domain assumptions, obstacles, and structured temporal requirements. For a well-formed operationalized fragment, we define typed source constructs and a translation-based formal interpretation into Event-B state-transition models and Linear Temporal Logic (LTL) formulae. The interpretation specifies the correspondence of preconditions, updates, goal predicates, dependency/refinement constraints, and global-scope temporal patterns with the generated artifacts. Event-B supports invariant checking and behavioral exploration, while LTL expresses progress, response, and ordering obligations over the associated traces. Two case studies, MinePump and MeetingScheduler, illustrate representational coverage and the use of Rodin and ProB for formal analysis. The evidence concerns feasibility and consistency of the examined generated artifacts, not a general proof of end-to-end semantic preservation. This explicit scope distinguishes the proposed systematic formalization process from a semantics-preserving compiler and identifies conditions for stronger verification claims.

Keywords: Goal-oriented Requirements Engineering; iStar; Model Transformation; Event-B; LTL.

1. Introduction

Recent studies emphasize the continuing role of goal models and the need for systematic evaluation and traceability in automated requirements engineering [1, 2]. In modern software engineering, many systems must address not only what should be done, but also when, in what order, and under what conditions behaviors should occur. In this context, goal-oriented requirements engineering (GORE) [3, 4] provides a conceptual basis for capturing stakeholders' goals, intentions, and dependency relationships; within GORE approaches, one such language, iStar [5], enables modeling of goals, tasks, and actor dependencies. However, such languages remain fundamentally declarative and centered on the actor–goal perspective; as a result, they do not provide adequate means to explicitly represent

*Corresponding author.
E-mail address: hanhdd@vnu.edu.vn

system behavior, nor do they offer a clear formal semantic basis for state-related, behavioral, and temporal aspects [6, 7]. A number of studies have attempted to enhance the formal rigor of iStar models by mapping them to different formal languages. Several approaches [8] translate iStar into Z to derive mathematical specifications for constraint analysis and correctness verification. Another line of work [9] uses Alloy to support constraint-based automated checking and counterexample generation. In addition, ontology- and description-logic-based representations, such as iStar2.0-OWL, aim to standardize conceptual semantics and enable formal reasoning [10]. These efforts are valuable for clarifying constraints, detecting inconsistencies, and standardizing semantics at the structural or conceptual level. Several approaches [11] focus on requirements-driven adaptation under uncertainty, while others [12] extend iStar/GRL to improve modeling capabilities for collaborative cyber-physical systems. These efforts enhance the expressive power of goal models.

Event-B [13–15] provides a formal basis for state-based modeling, and several studies have derived Event-B specifications from goal-oriented requirements models [16–19]. In parallel, Linear Temporal Logic (LTL) provides a classical foundation for expressing temporal properties such as safety ($\Box P$, a condition must always hold) and liveness ($\Diamond P$, a condition must eventually be satisfied) [6, 7]. Despite their complementary nature, these two formalisms are rarely combined within a coherent semantic framework for goal-oriented models; as a result, the relationship between state-based and temporal aspects is not systematically established.

Motivated by these gaps, this paper proposes gStar as an extension of iStar, thereby enabling goal models to capture state-related, behavioral, and temporal aspects of system requirements. Within gStar models, temporal properties are specified using a structured natural language notation inspired by SpecPro [20], which allows requirements engineers to express temporal constraints in a readable yet formally grounded form. For a specified operationalized fragment, we give typed definitions and a translation-based formal interpretation into Event-B and Linear Temporal Logic (LTL). Event-B captures explicit state and transition relations, while LTL specifies requirements over the same execution traces. The paper defines local correspondence criteria, but does not claim a machine-checked semantics-preservation theorem for the entire transformation.

This article is a substantially extended version of our earlier work [21]. Compared with the preliminary version, which mainly presented a two-step pipeline from operationalized iStar models to LTL formulas and then to Event-B specifications, the present work takes a more explicit, precisely scoped formalization direction. Instead of treating Event-B as the primary transformation target and LTL as a secondary analytical layer, this work places both Event-B and LTL within a single formal semantic framework for gStar models. The main contributions of this work are as follows:

- gStar, an extension of iStar incorporating with explicit behavioral constructs and temporal-requirement constructs for representing time-related requirements and their traceability links.
- Typed definitions and a translation-based interpretation of the operationalized gStar fragment, with Event-B and LTL serving as complementary state-based and trace-based formal domains.
- Explicit formal rule schemata and a tool-supported transformation for deriving traceable Event-B and LTL artifacts, with stated correctness conditions and limitations.
- Two case studies assessing explicit state, behavioral, and temporal representations and the analyzability of the generated artifacts.

The remainder of this paper is organized as follows. Section 2 presents the background and research motivation. Section 3 describes the gStar metamodel. Section 4 presents a formal semantics framework for gStar. Section 5 presents the case study and tool support. Section 6 reports the experimental results. Section 7 discusses threats to validity. Section 8 discusses related work. Finally, Section 9 concludes the paper and outlines directions for future work.

2. Motivation and Background

This section motivates our study with an example and provides the necessary background.

2.1. Motivating Example

Many software systems operate in settings where achieving a common objective requires coordination among multiple actors, information exchange, and mutual dependencies. In this paper, we use the meeting-scheduling problem as a motivating example to illustrate these issues.

This example is adapted from the classical meeting-scheduling case used in foundational studies on iStar and early-phase requirements engineering by Eric Yu [22]. In those works, meeting scheduling is employed to illustrate how strategic dependencies and internal rationales can be modeled in organizational settings. Building on this example provides a well-established and credible basis for the present study.

Figure 1 shows the meeting-scheduling setting using iStar with three main actors: *Meeting Initiator*, *Participant*, and *MeetingScheduler System*. The *Meeting Initiator* starts the process by requesting a meeting to be arranged. The *Participant* provides availability information and responds to the proposed slot. The *MeetingScheduler System* acts as the central actor that receives the request, processes relevant information, and supports the coordination needed to schedule the meeting.

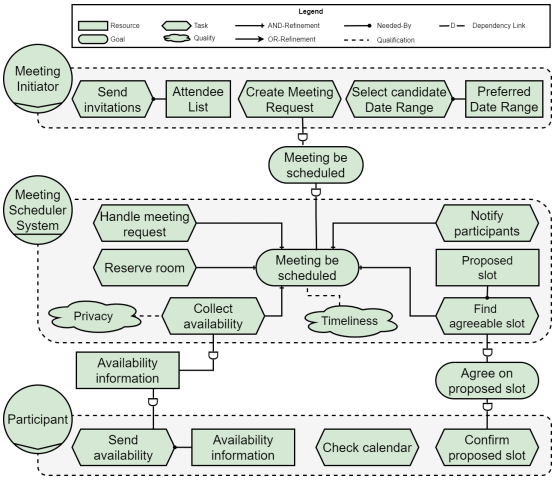


Figure 1. iStar model of meeting scheduling.

The central objective of the example is *Meeting be scheduled*. In Figure 1, this objective appears as a goal dependency between the *Meeting Initiator* and the *MeetingScheduler System*, indicating that the initiator depends on the scheduler system to achieve the desired outcome. This modeling choice is consistent with classical meeting-scheduling formulations in iStar, where the initiator delegates most of the scheduling effort to a specialized supporting actor.

Within the boundary of the *MeetingScheduler System*, the goal *Meeting be scheduled* is associated with several main tasks, including *Handle meeting request*, *Collect availability*, *Find agreeable slot*, *Reserve room*, and *Notify participants*. The resource *Proposed slot* represents an intermediate result produced during the slot-selection process. On the side of the *Meeting Initiator*, the main tasks are *Create Meeting Request*, *Send invitations*, and *Select candidate Date Range*, together with the resources *Attendee List* and *Preferred Date Range*. On the side of the *Participant*, the model includes the tasks *Check calendar*, *Send availability*, and *Confirm proposed slot*, as well as the resource *Availability information*.

The example also highlights several important dependencies. First, the *Meeting Initiator*

depends on the *MeetingScheduler System* for the goal *Meeting be scheduled*. Second, the *MeetingScheduler System* depends on the *Participant* for the resource *Availability information*. Third, the *MeetingScheduler System* depends on the *Participant* for the goal *Agree on proposed slot*. These dependencies show that meeting scheduling is inherently collaborative. At the same time, they also indicate that, although iStar makes such inter-actor relationships explicit at the structural level, their precise behavioral interpretation remains implicit, which becomes problematic when a formal semantic account is required.

In addition to functional concerns, the example also reflects quality-related concerns such as *Timeliness* and *Privacy*. *Timeliness* expresses the expectation that the scheduling process should progress without undue delay, while *Privacy* reflects the sensitivity of availability information. Their presence indicates that the requirements problem involves not only what should be achieved, but also additional constraints whose precise formal interpretation is not made explicit in standard iStar.

Figure 1 shows that iStar is effective for capturing actors' intentions and dependencies. At the same time, the example also reveals the challenges of iStar when a formal semantic interpretation is required. Although the model clearly represents goals, tasks, resources, and dependencies, it does not make explicit such aspects as states, operations, temporal ordering between coordination steps, or exceptional situations. As a result, both the behavioral and the temporal meaning of these constructs remain under-specified for a consistent formal interpretation. This observation motivates the need for a more system-oriented extension that makes such information explicit and prepares the source model for both state-based and trace-based semantic interpretation in the subsequent sections.

2.2. Goal-Oriented Requirements Engineering

Goal-oriented requirements engineering views requirements not merely as a list of functions to be implemented, but as a structured set of intentions, objectives, and dependencies among stakeholders. This perspective is particularly useful in the early stages of requirements engineering, where the main concern is not yet the detailed operational behavior of the system, but rather the clarification of why the system is needed, whose interests it serves, and how responsibilities are distributed across actors.

Among goal-oriented approaches, iStar is one of the most widely used frameworks. Its main strength lies in the explicit representation of actors, intentional elements, and dependency relationships in organizational settings. As a result, iStar is especially suitable for modeling requirements at a stage where strategic concerns and stakeholder relationships are more central than detailed behavioral specification.

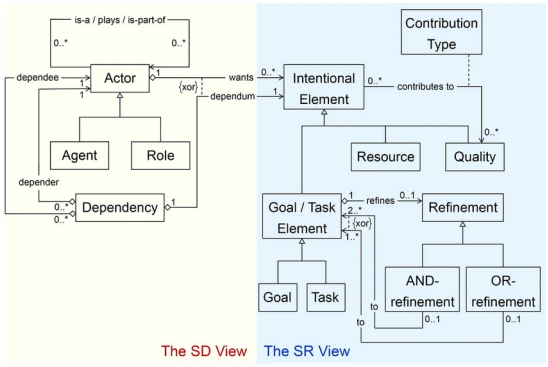


Figure 2. iStar metamodel.

Structurally, iStar is organized around two complementary views. The *Strategic Dependency* view focuses on dependency relationships among actors. The *Strategic Rationale* view captures the internal intentional structure of each actor, including its goals, tasks, resources, qualities, and refinement relationships. Figure 2 presents the core iStar metamodel from these two perspectives.

The main concepts of iStar can be observed directly in Figure 1. An *actor* is an

intentional entity with a role in the organizational setting. In the meeting-scheduling example, *Meeting Initiator*, *MeetingScheduler System*, and *Participant* are the main actors. A *goal* denotes a desired state to be achieved; in this example, *Meeting be scheduled* is the central goal. A *task* denotes a way of achieving a goal; tasks such as *Handle meeting request*, *Collect availability*, *Find agreeable slot*, and *Send availability* illustrate this notion. A *resource* represents an informational or physical entity used or exchanged among actors, such as *Availability information*, *Proposed slot*, *Attendee List*, *Preferred Date Range*. A *quality* as *Timeliness* expresses an evaluative concern.

A particularly important notion in iStar is *dependency*. A *dependency* expresses that one actor relies on another actor to achieve a goal, perform a task, or provide a resource. In the meeting-scheduling example, the *Meeting Initiator* depends on the *MeetingScheduler System* for the goal *Meeting be scheduled*. Likewise, the *MeetingScheduler System* depends on the *Participant* for the resource *Availability information*, and also for reaching agreement on the proposed slot. These dependencies make explicit the collaborative nature of the problem and show how responsibilities are distributed across actors.

In addition, iStar supports internal structuring through refinement links and relationships between tasks, resources, and qualities. In Figure 1, the goal *Meeting be scheduled* inside the *MeetingScheduler System* is associated with a set of tasks that represent possible means for achieving it. Qualities such as *Timeliness* and *Privacy* provide an additional evaluative perspective on these tasks. Through such constructs, iStar offers a suitable conceptual basis for reasoning about stakeholder intentions and organizational dependencies in the early phases of requirements engineering.

However, as suggested by the motivating example, the strength of iStar lies primarily in representing intentional and organizational structure rather than explicit operational behavior. It captures what actors want to achieve and how they depend on one another, but it does not make explicit intermediate states, behavioral progression, temporal ordering, or exception handling. Consequently, although iStar provides an appropriate starting point for goal-oriented modeling, an additional modeling layer is needed when one aims to move from requirements models toward formally analyzable behavioral specifications.

2.3. Event-B for State-Based Formal Semantics

Event-B [13] is a formal method based on *invariant-based development*, in which safety properties are specified as invariants and system behavior is represented by events. This approach is particularly suitable for problems where the system must be modeled at an abstract level while still supporting formal analysis and verification. Rather than describing execution as a procedural workflow, Event-B models a system in terms of valid states and possible state transitions, together with proofs that these transitions preserve the required constraints.

From a structural perspective, an Event-B specification is organized around several core meta-concepts as shown in Figure 3. At the highest level, a model is typically contained in a *project*. Within it, the two central components are *context* and *machine*. A *context* captures the static part of the model, including sets, constants, and axioms. It is used to define domain concepts and assumptions that remain unchanged during execution. A *machine*, in contrast, captures the dynamic part of the model, including state variables, invariants, and events.

Variables in a *machine* represent the current state of the system. Invariants specify the conditions that must always hold over that state. Events describe the possible transitions between states. Each event is typically defined by two main components: a *guard* and an *action*. A *guard* specifies the enabling condition under which the

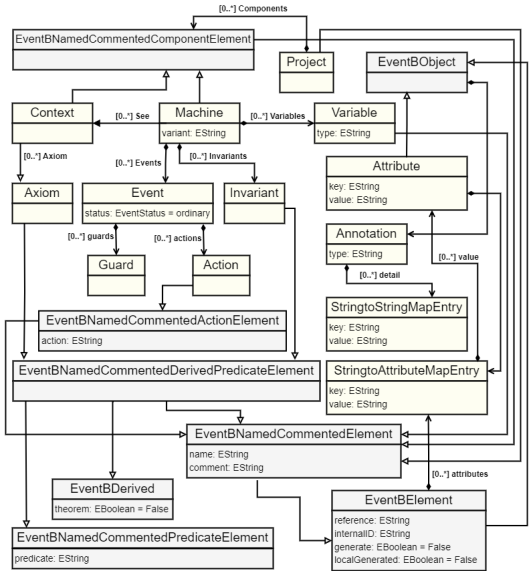


Figure 3. A simplified metamodel of Event-B.

event may occur, whereas an *action* describes how the system state is updated when the event takes place. This guard/action structure allows Event-B to model system behavior as a sequence of state transitions while providing the basis for checking whether invariants are preserved after each transition.

Another important relation in Event-B is the *sees* relation between a *machine* and a *context*. This relation indicates that the machine may use the static concepts declared in the context. In this way, Event-B clearly separates domain-level knowledge from dynamic system behavior.

Built on top of Event-B, Rodin [14] and ProB [15] form an important verification ecosystem. Rodin is an integrated environment for Event-B that supports proof-based development. Its main role is to generate proof obligations from the model and assist in proving or discharging them. Through this mechanism, Rodin makes it possible to check whether events indeed preserve the declared invariants. In other words, Rodin helps establish that a model is not only syntactically correct but also logically consistent with respect to its safety constraints.

Complementing Rodin, ProB provides animation and model checking for Event-B. Instead of operating solely at the level of proof, ProB enables the exploration of the model's state space, the inspection of reachable states, the detection of deadlocks, the identification of invariant violations, and the observation of possible execution traces. As such, ProB is especially useful for analyzing the dynamic behavior of a model, particularly when one needs to investigate progress-related or reachability-related properties. In practice, Rodin and ProB are often used in a complementary manner: Rodin provides proof-based assurance, whereas ProB provides behavior-oriented exploration and checking.

2.4. LTL for Trace-Based Temporal Semantics

Linear Temporal Logic (LTL) [6] is a linear-time temporal logic for specifying system properties over time, where time is viewed as a sequence of successive states, often assumed to be infinite. LTL is well suited to expressing temporal requirements such as "something must always hold" and "something will eventually happen," which are not fully captured by invariant-based specifications, especially when one needs to represent progress, response, and ordering relations over execution traces.

In LTL, atomic propositions are state predicates evaluated as true or false in each state, for example when a Boolean flag is true or a condition over state variables holds. LTL provides standard temporal operators [7, 23] for relating propositions over time, including *Globally* (G), *Eventually* (F), *Next* (X), and *Until* (U). Using these operators, LTL commonly specifies (i) *safety* conditions, often written as $G\,\neg bad$, and (ii) *liveness* or *progress* conditions, often written as $F\,good$ or $G(trigger \rightarrow F\,goal)$.

For the purposes of this paper, LTL is considered not merely as a property-checking notation, but as a *trace-based temporal semantic domain*. Under this view, temporal requirements

are interpreted over execution traces through atomic propositions associated with state predicates and behavioral milestones. This makes LTL particularly suitable for giving a formal temporal interpretation to ordering, response, and progress relations extracted from the source model, while maintaining a clear semantic connection to the corresponding state-based representation.

2.5. Structured Temporal Requirements

In addition to LTL as a temporal logic, it is useful to consider a more structured layer for representing temporal requirements before and alongside their formal temporal interpretation. A well-known basis for this idea is provided by *Property Specification Patterns* (PSPs), which were introduced to capture recurring classes of temporal requirements in a form that is easier to write, classify, and reuse than raw temporal-logic formulae [24]. In this perspective, patterns such as *absence*, *existence*, *response*, and *precedence* provide an intermediate but semantically meaningful representation between informal requirements and temporal logic.

Subsequent work has extended PSP-based specifications with Boolean and constrained numerical signals, and has shown how such structured requirements can be encoded into LTL formulae for formal analysis [25, 26]. This line of work is particularly relevant when temporal requirements must remain readable at the requirements level while still admitting a precise formal interpretation.

In this context, SpecPro [20] is an open-source Java library for the analysis and development of formal requirements. According to its official repository, SpecPro currently supports both a structured language based on PSPs and direct LTL formulae, and provides dedicated grammars for requirement patterns and LTL syntax [20]. In the present work, SpecPro is not introduced as a replacement for LTL. Rather, a SpecPro-oriented style is adopted as a *structured way of representing temporal requirements* for representing temporal requirements prior to, and in correspondence with, their LTL interpretation.

This role is important for the present work. The temporal-requirement constructs introduced in the extended gStar metamodel is intended to represent temporal requirements in a structured form, using concepts such as kind, scope, trigger, and response. In this sense, a SpecPro-oriented style provides an appropriate semantic basis for relating source-level temporal requirements to their corresponding LTL interpretations, while maintaining explicit traceability to the source-model elements from which they originate.

2.6. Research Challenges

We identify two main research challenges for this work as follows. The first challenge is to extend iStar so that it can represent the necessary information and aspects required for requirements models in GORE methods. In iStar, model elements are used primarily to represent actors' intentions and dependency relationships. However, as the *MeetingScheduler* example illustrates, this is not sufficient for establishing formal representations of behavior and temporality. Information such as the system–environment boundary, operational steps, progress milestones, properties and constraints, domain assumptions, obstacles, and temporal requirements must be made explicit within requirements models. The first challenge is therefore to systematically identify the categories of information that a goal model must contain. This identification is grounded in a synthesis of GORE approaches, particularly iStar and system-oriented approaches such as KAOS [3, 4].

The second challenge is to establish a formal semantics for the proposed language in a manner that is both theoretically precise and operationalizable through model transformation. In the present setting, semantics cannot remain at the level of intuitive explanation or informal concept mappings. Rather, a formal semantic

basis is required so that the elements of the source model can be systematically interpreted within two complementary formal domains: Event-B for state-based and behavioral aspects, and LTL for temporal aspects. Moreover, because this semantics is to be operationalized through model transformation, the correctness and semantic consistency of the transformation rules themselves also become part of the research problem, as they must faithfully reflect the defined semantics. Specifically, it must be determined how the consistency of gStar models can be checked and ensured, particularly regarding potential conflicts between the behavioral specification and the temporal-property constraints stated over the model.

3. The gStar Metamodel

This section presents gStar as an extension of iStar, intended to make explicit the categories of information that requirements models in GORE must capture [3, 4], including operations, domain properties, domain assumptions, obstacles, and temporal requirements. Accordingly, this section introduces the extensions of gStar from three perspectives: the representation of operations, the representation of properties, and the representation of structured temporal requirements, before consolidating these extensions into the gStar metamodel.

3.1. Operations in gStar

An important limitation of iStar is that it mainly represents `Task` at the intentional level, whereas the operations performed by the system are not represented as a separate category in the model. As a result, iStar alone is often not sufficient when the model must indicate what the system does, under which conditions, and through which operational elements.

To address this issue, gStar introduces two main extensions. First, it clarifies the distinction between actors that belong to the system and

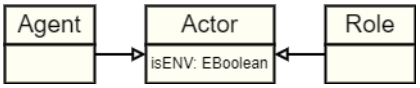


Figure 4. System–environment boundary in gStar.

actors that belong to the environment or to external services. As shown in Figure 4, `Actor` remains organized as an abstract class with specializations such as `Agent` and `Role`, but is extended with the attribute `isEnv`. This allows the model to distinguish the actors whose operations are part of the system from those that represent external participation.

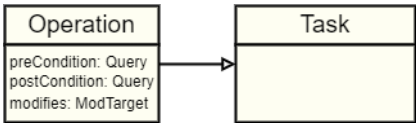


Figure 5. Operation-related extensions in gStar.

Second, gStar specializes `Task` into `Operation` in order to represent behavioral steps at the operational level. As shown in Figure 5, each `Operation` is associated with `preCondition`, `postCondition`, and a set of `modifies` targets, thereby describing its enabling condition, post-state condition, and affected elements. With these two extensions, gStar preserves the role of `Task` in iStar while adding a more suitable representation for the operations carried out by the system.

3.2. Properties in gStar

Besides operations, gStar also extends the representation of *properties* within the model. In this context, properties include not only constraints to be maintained, but also progress milestones, hindering conditions, and domain assumptions that are part of the requirements model.

To begin with, gStar introduces `GoalMode` to distinguish between *achieve* and *maintain* goals. This distinction indicates whether a goal denotes a state or milestone to be reached, or a

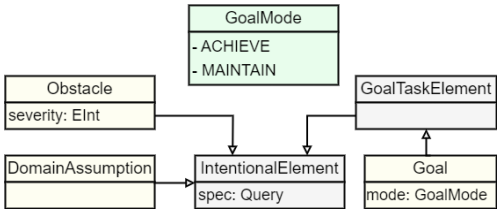


Figure 6. Property-related extensions in gStar.

property to be preserved throughout execution. gStar also introduces `Obstacle` as a separate construct for hindering situations that may disrupt progression or affect goal satisfaction. In addition, `DomainAssumption` is included as a separate model element so that assumptions about the problem domain can be recorded directly within the model rather than left scattered across textual descriptions. These three constructs are summarized in Figure 6.

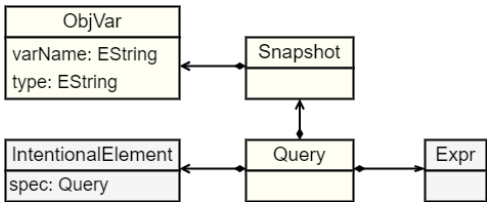


Figure 7. Query and expression support in gStar.

To support the expression of such properties, gStar further introduces a structured query–expression layer. As shown in Figure 7, this layer is organized around classes such as `Query`, `Snapshot`, `ObjVar`, and the expression hierarchy rooted at `Expr`. Its role is to provide a uniform mechanism for stating the conditions that appear in goals, qualities, domain assumptions, obstacles, and operations. Together, these extensions provide gStar with a more adequate basis within the model for representing what must be achieved, what must be maintained, what may obstruct progress, and what is assumed from the domain.

Clarification of intention versus goal. In iStar terminology, *intentional elements* are model elements associated with an actor's rationale; the class name does not mean that goals are intentions in the Belief–Desire–Intention (BDI) sense. In particular, a `Goal` represents a desired state, whereas a `Task` specifies a means and an `Operation` adds explicit execution conditions. The current gStar metamodel does not define a first-class BDI `Intention` or `Plan` class and does not equate an intention to act with a state predicate. This distinction is made precise by the goal, plan, and commitment interpretations in Definitions 2 and 3, consistently with planning accounts of intention [27].

3.3. Structured Temporal Requirements in gStar

In addition to operations and properties, gStar introduces a dedicated set of constructs for specifying requirements about temporal progression and ordering. Although goals, operations, obstacles, and domain assumptions provide an important basis for describing states and behavior, they are not sufficient to state concerns such as progress, response, ordering, or exclusion directly in the model.

The design of this part of the metamodel is influenced by structured temporal-requirement notations, including the SpecPro style, which motivate the use of constructs such as kind, scope, trigger, and response. Accordingly, gStar introduces `TemporalRequirement` as a dedicated construct. In the current metamodel, `TemporalRequirement` includes attributes such as `kind`, `scope`, `pattern`, `component`, `formalism`, `fretishText`, and `ltlText`, together with core predicates such as `trigger`, `response`, `statePredicate`, and `until`.

gStar also introduces `TemporalVariable` for naming state predicates, milestones, or conditions relevant to temporal requirements, and `TemporalLink` for connecting a `TemporalRequirement` to the corresponding intentional element in the model. Figure 8 shows these three constructs as the core constructs for specifying temporal requirements. In this way, temporal requirements become part of the model

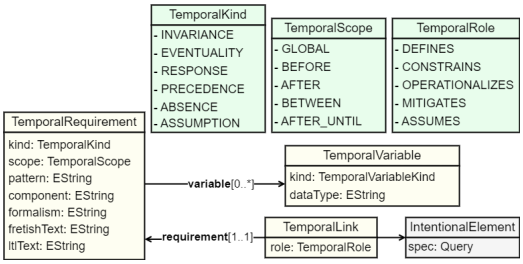


Figure 8. Temporal-requirement constructs in gStar.

as elements linked to the corresponding model elements rather than being added only afterwards.

3.4. The Unified gStar Metamodel

The extensions presented above are consolidated into a unified metamodel, as shown in Figure 9. This metamodel preserves the iStar core while adding the structures required for operations, properties, domain assumptions, obstacles, and temporal requirements.

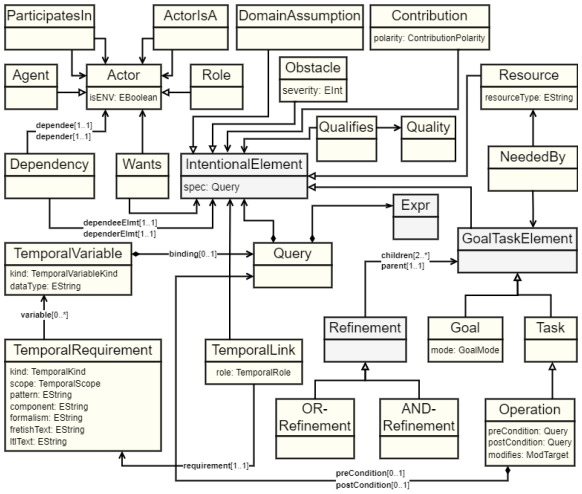


Figure 9. The gStar metamodel.

At the overall organizational level, the metamodel is centered on `GStarModel`, in which elements are grouped into three main collections: `nodes`, `links`, and the temporal-requirement constructs composed of `temporalRequirements`, `temporalLinks`, and `temporalVariables`. This organization shows that temporal information is kept within the same metamodel rather than treated separately.

Beyond its conceptual structure, the metamodel is also constrained by well-formedness conditions that help ensure the consistency of the model. For instance, each `IntentionalElement` must belong to exactly one actor through the `Wants` relation:

```
context IntentionalElement
inv exactlyOneOwner:
  self.wants->size() = 1
```

Similarly, elements that carry condition-like content, such as `Goal`, `Quality`, `DomainAssumption`, and `Obstacle`, are associated with a `spec` expressed as a `Query`, whereas an `Operation` is well formed only if both a precondition and a postcondition are provided:

```
context Operation
inv HasPreAndPost:
  not self.preCondition.oclIsUndefined() and
  not self.postCondition.oclIsUndefined()
```

For temporal requirements, `TemporalRequirement` is subject to formation conditions corresponding to the type of requirement being stated. For example, a response- or precedence-type requirement must include both a `trigger` and a `response`:

```
context TemporalRequirement
inv triggerResponseShape:
  (self.kind = TemporalKind::RESPONSE or
   self.kind = TemporalKind::PRECEDENCE)
implies (not self.trigger.oclIsUndefined()
and not self.response.oclIsUndefined())
```

These OCL expressions are included as representative examples of central well-formedness conditions enforced by the metamodel. The complete metamodel and additional OCL constraints are provided with the tool-support artifacts described later in the paper.

Overall, gStar combines the elements retained from the iStar core with additional structures for operations, properties, domain assumptions, obstacles, and temporal requirements. In this

sense, gStar does not replace iStar; rather, it extends iStar so that the model can capture these categories of information directly.

4. A Formal Semantic Framework for gStar

This section establishes a formal semantic framework for gStar. On that basis, state-based and behavioral aspects are interpreted in Event-B, whereas temporal requirements are interpreted in LTL.

4.1. Scope and Semantic Principles

The framework supplies a typed, translation-based semantics for the operationalized fragment of gStar. Its claim is a precise interpretation of explicitly modeled state, operational, and temporal information, rather than the unqualified preservation of all iStar/BDI semantics. The focus is on elements that represent states, behavior, constraints, and temporal requirements.

In the Event-B interpretation, the focus is on constructs related to state, behavior, and constraints, including state variables, invariants, state transitions, enabling conditions, update effects, domain assumptions, and hindering or handling conditions whenever they must be reflected in state-based analysis. This interpretation is centered mainly on the system actor, whereas environment-related aspects are reflected through domain assumptions, guards, environment variables, or explicit environment events when needed.

In the LTL interpretation, the focus is on temporal requirements represented by the temporal-requirement constructs of gStar. In gStar, these requirements are specified in a SpecPro-oriented style; within the semantic framework, they are mapped to LTL formulae. These requirements concern progress, response, ordering, exclusion, and related temporal obligations over execution traces. This interpretation is not defined independently of the state-based interpretation; rather, it relies on the same state basis established in the Event-B interpretation, through the corresponding states, milestones, and behavioral conditions.

At the realization level, the formal interpretations established above are implemented through model transformation and artifact generation. The specification part of the gStar model that corresponds to Event-B is realized through model transformation followed by text generation, whereas the temporal part is realized through model-to-text generation of the corresponding temporal artifacts. In this way, the gStar model provides the common basis for generating the formal artifacts established in this paper.

4.2. Typed Semantic Domain and Core Definitions

The contribution is a *translation-based formal interpretation* of a well-formed, operationalized fragment of gStar, rather than a proof that an arbitrary iStar model is translated with full semantic equivalence. We distinguish (i) the intentional source model, (ii) the state-transition semantics induced by explicit operational annotations, and (iii) Event-B and LTL artifacts realizing that semantics. The following definitions fix the domains and the meaning of the constructs involved in the rules of Section 4.3.

Definition 1 (Typed gStar fragment). *A source model is a finite structure*

$$\mathcal{G} = (A, E, W, D, F, O, Pre, Post, Mod, \Theta), \quad (1)$$

where $A = A_{\text{ag}} \uplus A_{\text{role}}$ *is a set of actors classified as* agents *(concrete intentional actors) or* roles *(abstract responsibilities);* $E = G \uplus T \uplus Q \uplus R$ *comprises goals, tasks, qualities, and resources;* $W \subseteq A \times E$ *assigns intentional elements to actors;* $D \subseteq A \times A \times E$ *denotes depender–dependee relationships; and F is a finite collection of AND/OR refinements over goal/task elements. A role may be played by an agent, modeled by plays* $\subseteq A_{\text{ag}} \times A_{\text{role}}$. *The subset* $O \subseteq T$

consists of operations *with explicit preconditions, postconditions, and modification sets. Finally,* Θ *is a collection of temporal requirements, each linked to source elements through* `TemporalLink`.

Here R denotes *resources*, not roles; each goal $g \in G$ has a mode $mode(g) \in \{\mathsf{achieve}, \mathsf{maintain}\}$ and a state predicate γ_g. The sets are mathematical abstractions of the concepts visible in the metamodel, not additional implementation classes. The source metamodel uses `isEnv` to separate environment actors from system actors; in the semantic domain the latter form $A_{\text{sys}} \subseteq A$. The actor condition in Listing 2 is therefore interpreted as membership in A_{sys} (equivalently, $\neg$`isEnv`).

Definition 2 (Tasks, actions, and plans). *A* task $t \in T$ *specifies an intentional means of pursuing a goal and need not have executable semantics. An* operation $o \in O$ *is a task refined with a guard and a state-update relation. An* action *is an atomic update in that relation (an Event-B assignment at the target level); it is not a synonym for a goal or a first-class iStar intention. A* plan *for actor a and goal g is an admissible finite sequence $\pi = o_1 \cdots o_n \in O^*$ consistent with modeled precedence/refinement constraints. The set $\Pi(a, g) \subseteq O^*$ denotes its permitted operationalizations. Plans are therefore* derived *from tasks, operations, and constraints; gStar does not claim to introduce a separate BDI plan object.*

Definition 3 (Intention and goal distinction). *A goal g is a proposition $\gamma_g : S \to \mathbb{B}$ describing a* desired state *or maintenance condition. Following the planning view of intention [27], an intention is instead a commitment to perform or sustain an action plan, rather than a state predicate: in the present abstract account, $Intends(a, \pi)$ relates an actor to a selected plan $\pi \in \Pi(a, g)$. The relation is* conceptual*, not an executable BDI mental-state component of the current metamodel. In particular, a* `Wants` *link $W(a, g)$ indicates goal ownership/motivation; it does* not *logically entail $Intends(a, \pi)$, since choosing and committing to a plan requires additional information. The present transformation interprets goal and operation predicates, not BDI belief revision, deliberation, or intention persistence.*

Definition 4 (Operational state semantics). *Let V be the typed state variables, S their valuations, and $S_0 \subseteq S$ the initial states. An operation $o \in O$ has a precondition $pre_o : S \to \mathbb{B}$, a postcondition $post_o : S \times S \to \mathbb{B}$, and a set $mod_o \subseteq V$. Its transition relation is*

$$R_o(s, s') \Leftrightarrow pre_o(s) \wedge post_o(s, s') \wedge \bigwedge_{v \notin mod_o} s'(v) = s(v). \tag{2}$$

The operational semantics is the labeled transition system $\mathcal{K}_{\mathcal{G}} = (S, S_0, \{R_o\}_{o \in O \cup O_{\text{env}}}, L)$, where O_{env} represents explicitly modeled environment steps and $L(s)$ gives the truth of state and milestone propositions at s. This semantics is defined only when predicates are well typed and updates are consistent. A merely declarative task without operational annotations is not automatically assigned an arbitrary state transition.

The basic semantic relations are made explicit as follows. A maintain goal requires $\forall s \in Reach(\mathcal{K}_{\mathcal{G}}) : \gamma_g(s)$; an achieve goal requires existence of a reachable s with $\gamma_g(s)$ for the weak reachability reading, or $\mathbf{F}\gamma_g$ for an explicitly stated progress obligation. These readings are *not equivalent*. For a complete AND refinement $F(g) = \mathsf{AND}(g_1, \ldots, g_n)$, the modeled satisfaction condition is $\gamma_g(s) \leftrightarrow \bigwedge_i \gamma_{g_i}(s)$; for an OR refinement it is $\gamma_g(s) \leftrightarrow \bigvee_i \gamma_{g_i}(s)$. Such biconditionals apply only when the modeler marks the decomposition complete; otherwise the direction from child satisfaction to parent satisfaction is the supported implication. A dependency $D(a, b, e)$ indicates that the depender a requires satisfaction or provision of e by b; a target-side guard or state relation can encode this only after the domain-specific fulfillment predicate has been supplied. No single universal guard is inferred from the dependency edge alone.

Definition 5 (Temporal interpretation). *A (totalized) trace* $\sigma = s_0 s_1 \ldots \in S^\omega$ *starts in* S_0 *and follows the transition relation; finite terminating executions are extended with stuttering steps solely to interpret standard infinite-trace LTL. A temporal variable* x *denotes a predicate* $[\![x]\!] : S \to \mathbb{B}$. *Satisfaction* $\sigma, i \models \varphi$ *is defined by standard LTL semantics; in particular,* $\sigma, i \models \mathbf{F}p$ *iff* $\exists j \geq i : \sigma, j \models p$ *and* $\sigma, i \models \mathbf{G}p$ *iff* $\forall j \geq i : \sigma, j \models p$. *The structured patterns used here assume* global *scope unless a different scope has been explicitly encoded.*

The definitions address the state, behavior, and trace aspects of the *operationalized fragment*. They do not imply that every intention, role-playing relationship, informal task, or scoped temporal requirement has a complete generated Event-B counterpart. Their explicit boundaries prevent conflating a source metamodel, formal semantic definitions, and illustrative ATL implementation rules.

4.3. Mapping gStar to Event-B

The Event-B interpretation is established through a set of rules that map the considered gStar constructs into the Event-B domain in accordance with their intended source-level meaning. In this work, the rules are constrained by the typed source and target interpretations of Section 4.2. The associated local correspondence criteria are stated precisely in Section 4.5; a global proof of semantic preservation for all implementations is outside the current scope.

At the realization level, these rules are implemented using the ATLAS Transformation Language (ATL), a language for model-to-model transformation that specifies mappings between source and target metamodels [28]. In this section, the ATL snippets are explanatory code sketches of selected mapping rules, not verified executable ATL modules. Their metamodel concepts correspond to the illustrated `gstar.ecore` fragment, while helper functions are abstract; the formal rule specifications and correctness conditions are stated separately in Section 4.5.

The rules are organized into three groups. The first group determines the overall structure of the target model in Event-B. The second group specifies how the considered gStar constructs are interpreted in this paper. The third group focuses on behavioral rules, in which operations are interpreted as state transitions and mapped to events.

4.3.1. Structural Rules

The first rule group determines how a gStar model is reorganized in the formal structure of Event-B. Table 1 summarizes the structural rules for organizing the target model in the Event-B *Project–Context–Machine* structure.

Table 1. Structural rules for organizing gStar models in Event-B

Rule[1]	**Source (gStar)**	**Event-B interpretation**
RS1	gStar model	Project
RS2	Domain elements and static assumptions	Context
RS3	System actor	Machine
RS4	Domain usage relation	Machine sees Context

RS1 - Mapping "gStar model" to "Project". The source of this rule is a complete gStar model at the root level. Its target is an Event-B *Project*. Semantically, this rule establishes the formal container for the whole target representation. The *Project* does not carry behavioral meaning directly; rather, it acts as the enclosing element for the *Context* and *Machine* generated by the subsequent rules. The rule is applied once for each root-level gStar model.

RS2 - Mapping "domain elements and static assumptions" to "Context". The source of

[1]RS denotes *Structural Rule*.

this rule consists of domain elements, type-level elements, domain constants, and static assumptions. Its target is an Event-B *Context* containing carrier sets, constants, and axioms. This rule reflects the principle that the static part of the problem domain must be placed in the *Context*. Accordingly, those source elements that do not evolve during execution but are required to define the value domain and the foundational assumptions of the system are interpreted as the static component of the target model. Rule RS2 is realized in ATL as shown in Listing 1.

Listing 1 Rule RS2 is realized in ATL

```
rule Domain2Context {
  from s :  GSR!Model
  to t :  EB!Context (
    name <- 'C_' + s.name,
    sets <- mapDomainVocabulary(s),
    constants <- mapStaticConstants(s),
    axioms <- mapStaticAxioms(s)
  )
}
```

This rule materializes the static meaning of the source model as a target *Context*; the semantic role of the rule is established at the conceptual level.

RS3 - Mapping "system actor" to "Machine". The source of this rule is a gStar actor classified as a system actor. Its target is an Event-B *Machine*. The rule reflects the principle that the behavior under system control must be interpreted in the dynamic component of Event-B. Hence, the system actor is not preserved as an organizational notion in the target model, but transformed into a *Machine* carrying state variables, invariants, and events. Rule RS3 is realized in ATL as shown in Listing 2.

This rule shows how the system actor in the source model is operationalized as a *Machine* in the target model.

RS4 - Mapping "domain usage relation" to "Machine–Context visibility relation". The source of this rule is the relation expressing that

Listing 2 Rule RS3 is realized in ATL

```
rule SystemActor2Machine {
  from s :  GSR!Actor(not s.isEnv)
  to t :  EB!Machine (
    name <- 'M_' + s.name
  )
}
```

the behavioral part of the system uses the static domain basis. Its target is the *sees* relation from *Machine* to *Context*. This rule ensures that the dynamic part of the system is connected to the static basis on which it depends. As a result, the sets, constants, and axioms declared in the *Context* become available for interpreting and analyzing the state variables, invariants, and events of the *Machine*.

Taken together, the structural rules reflect a basic Event-B principle: the static part of the problem domain is placed in the *Context*, whereas the dynamic part of the system is placed in the *Machine*. The *Project* acts only as a container, while the actual formal meaning is carried by the *Context–Machine* pair.

4.3.2. Construct Interpretation Rules

The second rule group specifies how the considered gStar constructs are interpreted in this paper. Table 2 summarizes the rules used to interpret these constructs in the Event-B domain.

RC1 - Mapping "maintain goal" to "invariant". The source of this rule is a maintain goal in gStar. Its target is an invariant in the generated *Machine*. Semantically, a maintain goal is understood as a state property that must hold in all reachable states of the system. In Event-B, this meaning is reflected naturally by an invariant, since invariants specify properties that must remain true throughout execution. Rule RC1 is realized in ATL as shown in Listing 4. This rule shows how the meaning of a maintain goal as

[2]RC denotes *Construct Interpretation Rule*.

Table 2. Rules for interpreting gStar constructs in Event-B

Rule[2]	**Source (gStar)**	**Event-B interpretation**
RC1	Maintain goal	Invariant
RC2	Achieve goal / milestone-like goal	State variable or state predicate
RC3	Domain assumption	Context axiom, machine invariant, or event guard
RC4	Obstacle	Hindering condition, exceptional path, or handling event
RC5	Dependency	State relation, invariant, or guard
RC6	Refinement	Conjunctive or alternative constraint structure

something that must always hold is materialized as an Event-B invariant. **RC2 - Mapping "achieve goal or milestone-like goal" to "state variable or state predicate".** The source of this rule is an achieve goal or a goal playing the role of a milestone in the process. Its target is a state variable, milestone flag, or state predicate in the *Machine*. Unlike maintain goals, achieve goals do not denote properties that must always hold; rather, they denote states or milestones that the system should be able to reach during execution. For this reason, they are not interpreted as invariants, but as elements of the state space that can be updated by events. Rule RC2 is realized in ATL as shown in Listing 3.

This rule shows how a progress-relevant milestone is introduced into the target model as part of the state space rather than as a purely intentional label.

RC3 - Mapping "domain assumption" to "axiom, invariant, or guard". The source of this rule is a domain assumption in gStar. Its target may be a *Context* axiom, a *Machine* invariant, or an event guard. Semantically, a domain assumption is understood as a domain condition

Listing 3 Rule RC2 is realized in ATL

```
rule MilestoneGoal2Variable {
  from s :  GSR!Goal (s.mode =
GSR!GoalMode#ACHIEVE)
  to v :  EB!Variable (
    name <- normalizeName(s.name)
  ),
  inv :  EB!Invariant (
    name <- 'type_' + normalizeName(s.name),
    predicate <- boolTypingOf(s)
  )
}
```

Listing 4 Rule RC1 is realized in ATL

```
rule MaintainGoal2Invariant {
  from s :  GSR!Goal(s.mode =
GSR!GoalMode#MAINTAIN)
  to t :  EB!Invariant (
    name <- 'inv_' + s.name,
    predicate <- toPredicate(s.spec)
  )
}
```

constraining either interpretation or behavior. If the assumption is static and foundational, it is placed in the *Context*; if it directly constrains behavior, it is reflected in the *Machine*. Rule RC3 is realized in ATL as shown in Listing 5.

Listing 5 Rule RC3 is realized in ATL

```
rule DomainAssumption2Carrier {
  from s :  GSR!DomainAssumption
  to a :  EB!Axiom (
    name <- 'axm_' + normalizeName(s.name),
    predicate <- toPredicate(s.spec)
  )
}
```

RC4 - Mapping "obstacle" to "hindering condition, exceptional path, or handling event". The source of this rule is an obstacle in gStar. Its target is a hindering condition, an exceptional path, or a handling event in Event-B. Semantically, an obstacle is understood as a condition that blocks

nominal progress, diverts execution, or forces the system into a non-nominal handling path. Hence, it is not preserved merely as an annotation, but as a source of analyzable non-nominal behavior in the target model. Rule RC4 is realized in ATL as shown in Listing 6.

Listing 6 Rule RC4 is realized in ATL

```
rule Obstacle2HandlingEvent {
  from s :  GSR!Obstacle
  to t :  EB!Event (
    name <- 'handle_' +
normalizeName(s.name),
    guards <- mapObstacleGuard(s),
    actions <- mapObstacleResolution(s)
  )
}
```

RC5 - Mapping "dependency" to "state relation, invariant, or guard". The source of this rule is a dependency relation in gStar. Its target is a state relation, invariant, or guard in the Event-B-based representation. The purpose of this rule is not to reproduce the original dependency graph literally, but to encode those operational consequences of the dependency that are relevant to state-based analysis.

RC6 - Mapping "refinement" to "conjunctive or alternative constraint structure". The source of this rule is an AND/OR refinement relation in gStar. Its target is a set of constraints over invariants, guards, or event structures in Event-B. Semantically, refinement is interpreted as a constraint over the satisfaction structure of the model. In the AND case, the generated representation must preserve the logic of joint necessity; in the OR case, it must preserve the existence of alternative satisfaction paths.

Taken together, these rules show that interpreting gStar constructs in Event-B is not aimed at preserving the graphical form of the source model, but rather those meanings that are relevant to state, constraints, progress, and hindering conditions in the target domain.

4.3.3. Behavioral Rules

The third rule group focuses on behavioral semantics, in which an operation is interpreted as a conditional state transition with an enabling condition and an effect. Table 3 summarizes the behavioral rules used to introduce the semantics of operations and handling behavior into the Event-B model.

Table 3. Behavioral rules for interpreting operations in Event-B

Rule[3]	**Source (gStar)**	**Event-B interpretation**
RB1	Operation	Event
RB2	Enabling condition / precondition	Guard
RB3	Effect / state update	Action
RB4	Obstacle-handling behavior	Handling event

RB1 - Mapping "operation" to "event". The source of this rule is an operation in gStar. Its target is an event in the generated *Machine*. Semantically, an operation denotes performing something when suitable conditions hold and thereby changing the system state. An Event-B event carries precisely this meaning: it is a transition enabled by guards and realized through actions. Rule RB1 is realized in ATL as shown in Listing 7.

This rule shows how the semantics of an operation as a conditional state transition is materialized in Event-B.

RB2 - Mapping "enabling condition" to "guard". The source of this rule consists of the preconditions or enabling conditions attached to an operation. Its target is the set of guards of the corresponding event. This rule preserves the execution meaning of the operation: if an operation is allowed to occur only under certain conditions, then the generated event must also

[3]RB denotes *Behavioral Rule*.

Listing 7 Rule RB1 is realized in ATL

```
rule Operation2Event {
  from s :  GSR!Operation
  to t :  EB!Event (
    name <- 'op_' +
      thisModule.normalizeName(s.name),
    guards <- toGuards(s.preCondition),
    actions <- toActions(s.postCondition,
      s.modifies)
  )
}
```

be enabled only under those same conditions. Rule RB2 is realized in ATL as shown in Listing 8.

Listing 8 Rule RB2 is realized in ATL

```
rule Precondition2Guard {
  from s :  GSR!Expr
  to g :  EB!Guard (
    name <- 'grd_' + thisModule.freshId(),
    predicate <- toPredicate(s)
  )
}
```

RB3 - Mapping "effect" to "action". The source of this rule consists of the effects or state updates associated with an operation. Its target is the set of actions of the corresponding event. This rule preserves the state-update meaning of the operation by introducing the source behavioral effect into the action structure of Event-B. Rule RB3 is realized in ATL as shown in Listing 9.

Listing 9 Rule RB3 is realized in ATL

```
rule Effect2Action {
  from s :  GSR!Expr
  to a :  EB!Action (
    name <- 'act_' + thisModule.freshId(),
    assignment <- toAssignment(s)
  )
}
```

RB4 - Mapping "Obstacle-handling behavior" to "handling event". The source of this rule is an Obstacle-handling behavior or an exceptional behavioral fragment associated with an obstacle. Its target is a handling event in the generated *Machine*. This rule ensures that the target model captures not only nominal behavior, but also explicit exceptional or recovery behavior. Rule RB4 is realized in ATL as shown in Listing 10.

Listing 10 Rule RB4 is realized in ATL

```
rule ObstacleHandling2Event {
  from s :  GSR!Obstacle
  to t :  EB!Event (
    name <- 'handle_' +
      normalizeName(s.name),
    guards <- mapObstacleGuard(s),
    actions <- mapRecoveryEffects(s)
  )
}
```

This rule shows how an obstacle is not lost during interpretation, but reflected as a source of exceptional or recovery behavior in the target model.

4.4. Mapping Temporal Requirements to LTL

Besides the state-based and behavioral aspects interpreted in Event-B, the considered gStar fragment also includes requirements with an inherently temporal meaning, such as progress, response, ordering, and exclusion over execution traces. In this paper, these requirements are interpreted in LTL using a SpecPro-oriented style, since property specification patterns provide a well-established basis for expressing temporal obligations in a structured form before encoding them as formal temporal formulae, while SpecPro directly supports both pattern-based structured requirements and their corresponding LTL representations [20, 24–26]. Within this scope, the interpretation is centered on the gStar constructs used for temporal requirements, namely

`TemporalVariable`, `TemporalRequirement`, and `TemporalLink`, and is constructed on the same state basis already made explicit in the Event-B interpretation. On this basis, the temporal interpretation rules are summarized in Table 4.

Table 4. Rules for interpreting temporal requirements in LTL

Rule[4]	**Applies to**	**LTL interpretation**
RL1	`Temporal Variable`	atomic temporal proposition
RL2	`EVENTUALITY` requirement	eventuality formula
RL3	`RESPONSE` requirement	response formula
RL4	`INVARIANCE` requirement	global invariance formula
RL5	`ASSUMPTION` requirement	assumption-style temporal formula
RL6	`TemporalLink`	traceability to source element

Among these rules, RL1 establishes the anchoring basis between the temporal-requirement constructs and the states or milestones that are observable in the model. RL2 to RL5 interpret the main classes of temporal requirements as the corresponding LTL formulae, including obligations to eventually reach a state, obligations to provide a response after a triggering condition, constraints that must hold globally over execution traces, and assumptions about expected behavior from the environment or the dependee side. RL6 preserves the traceability relation between each temporal requirement and the corresponding source-model element. At the realization level, this part is realized as a model-to-text transformation using Acceleo in order to generate the corresponding temporal artifacts.

Taken together, the Event-B interpretation establishes the state basis of the considered fragment, whereas the temporal interpretation makes explicit the obligations defined over its execution traces.

4.5. Formal Rule Schemata and Consistency Conditions

The following mathematical schemata specify the meaning of the rule groups, complementing the representative ATL excerpts. Let a well-formed gStar model $\mathcal{G}$ be given as defined in Definition 1. Let $\mathsf{EB}(\mathcal{G}) = (C, M)$ denote its generated Event-B Context–Machine pair and $\mathsf{LTL}(\mathcal{G})$ the generated set of LTL formulae. In the mapping notation $\mathsf{pred}(q)$ translates a typed source query into a target predicate on the same variable valuation; $\mathsf{assign}(o)$ translates the state relation of an operation into a simultaneous Event-B generalized substitution. These translations are defined only where such predicates and substitutions can be constructed without ambiguity.

Structural and construct rules (RS1–RS4; RC1–RC6). The target structure and the core interpretations are specified by

$$\begin{aligned} \mathsf{Project}(\mathcal{G}) &\supseteq \{C, M\}, \quad M \text{ sees } C, \\ \mathsf{Axioms}(C) &\supseteq \{\mathsf{pred}(d) : d \in D_{\text{static}}\}, \\ \mathsf{Inv}(M) &\supseteq \{\mathsf{pred}(\gamma_g) : g \in G_{\text{maintain}}\}, \\ \mathsf{Sat}_M(g, s) &\Leftrightarrow \mathsf{pred}(\gamma_g)(s), \quad g \in G_{\text{achieve}}. \end{aligned} \tag{3}$$

Here D_{static} is the set of static domain assumptions (distinct from the dependency relation D), and $G_{\text{achieve}} = \{g \in G : \mathit{mode}(g) = \mathsf{achieve}\}$. The final line defines the satisfaction predicate of an achieve goal Sat_M over a typed Event-B state s; it does not require adding a Boolean variable for each achieve goal. An explicit state variable may still be used if the modeled milestone cannot otherwise be expressed by an existing state predicate. Thus, this correspondence specifies the semantics of goal satisfaction, rather than an unconditional rule for creating new state variables. Dynamic assumptions are added as guards or invariants according to their explicitly given scope. Obstacles are mapped to their supplied hindering

[4]RL denotes *Temporal Interpretation Rule*.

predicate and to a handling event *only if* a recovery operation is modeled. Dependency constraints $\mathsf{fulfill}(b, e, s)$ and AND/OR refinement constraints are mapped using the predicate interpretations defined in Section 4.2; there is no semantics-free default event for an abstract link.

Operational rules (RB1–RB4). For each operationalized $o \in O$, the generated Event-B event E_o has guard $\mathsf{pred}(pre_o)$ and generalized substitution $\mathsf{assign}(o)$, subject to

$$(s, s') \in [\![E_o]\!]_{\mathrm{EB}} \;\Leftrightarrow\; R_o(s, s') \quad (s, s' \in S_{\mathrm{typed}}). \tag{4}$$

The right-hand side includes the frame condition in Eq. (2). Equation (4) is the *local correctness criterion* that a concrete implementation of RB1–RB3 must satisfy; it is not asserted here as a mechanically established theorem for the whole generator. The Event-B invariant-preservation obligation for invariant I is

$$I(s) \wedge pre_o(s) \wedge R_o(s, s') \;\Rightarrow\; I(s'). \tag{5}$$

When an obstacle h has a modeled recovery o_h, RB4 uses $pre_{o_h}(s) \wedge h(s)$ as its enabling condition; the recovery effect must satisfy the same transition and frame conditions. A static domain assumption does not automatically imply a recovery plan.

Temporal rules (RL1–RL6). For globally scoped temporal requirements with atomic state predicates p and q, the principal translations are

$$\begin{aligned}
\mathsf{TL}(\mathsf{EVENTUALITY}(q)) &= \mathbf{F}q,\\
\mathsf{TL}(\mathsf{RESPONSE}(p, q)) &= \mathbf{G}(p \rightarrow \mathbf{F}q),\\
\mathsf{TL}(\mathsf{INVARIANCE}(q)) &= \mathbf{G}q,\\
\mathsf{TL}(\mathsf{PRECEDENCE}(p, q)) &= \neg q \;\mathbf{W}\; p,\\
\mathsf{TL}(\mathsf{ASSUMPTION}(a, \varphi)) &= (\mathbf{G}a) \rightarrow \varphi.
\end{aligned} \tag{6}$$

The weak-until operator is $u \;\mathbf{W}\; v \equiv (u \;\mathbf{U}\; v) \vee \mathbf{G}u$. Thus, precedence prevents q before p without additionally requiring that p eventually occur. The assumption translation is *assume–guarantee* reasoning: a is a separately identified environmental assumption and φ its conditional guarantee, not a new LTL operator. RL1 binds each proposition to a state predicate in L; RL6 records a provenance link, which has no additional truth condition. Other scopes and patterns require explicit scope delimiters and are outside the displayed global-scope schemata.

Cross-domain coherence and the status of preservation. For every generated atomic proposition p, the cross-domain coherence condition is defined as follows:

$$L(s)(p) = \mathsf{true} \quad \Leftrightarrow \quad \mathsf{pred}(p)(s), \qquad s \in S_{\mathrm{typed}}. \tag{7}$$

If Eq. (4) holds for every event, initial states coincide, and Eq. (7) holds, then the source operational transition system and the generated machine admit corresponding labeled traces; under the same fairness and environment assumptions, the corresponding LTL properties have the same truth values. This is a *conditional preservation argument*, not a proof that the current ATL/Acceleo implementation meets the premises in every possible input. Rodin establishes specified Event-B proof obligations and ProB checks chosen bounded/finite-state models; neither result alone establishes end-to-end translation correctness.

Why separate LTL rather than only temporal Event-B extensions? Standard Event-B emphasizes invariants and state transitions; temporal properties may nevertheless be analyzed using ProB's LTL support or specialized Event-B temporal extensions and liveness-refinement methods [29–31]. We deliberately retain a separate, traceable LTL artifact because the `TemporalRequirement` metamodel already expresses structured patterns, and the target Event-B model can serve as the transition-system basis for checking those formulae. This is a *toolchain and traceability design choice*, not a claim that Event-B intrinsically cannot handle temporal logic. Fairness is not inferred from ordinary

invariants: a response formula such as $\mathbf{G}(p \rightarrow \mathbf{F}q)$ needs an appropriate scheduling/environment assumption when unconditional eventual progress cannot be guaranteed.

4.6. Semantic Properties of the Transformation

Having established the interpretation rules for both the Event-B interpretation and the temporal interpretation, it is necessary to state the semantic properties that guide the evaluation reported in Section 6. These properties are not intended to constitute a complete semantic proof for the whole of gStar. Rather, they identify, within the scope of this paper, the main meanings of the source model that should be reflected in the generated formal artifacts. The corresponding Event-B specifications, LTL formulae, and analysis artifacts used to witness these properties are provided in the accompanying tool-support repository described in Section 5.3.

P1 - Maintain-goal preservation. Maintain goals are interpreted as state constraints in the Event-B interpretation. Accordingly, their meaning is no longer retained only at the level of goal descriptions in the source model, but is realized as invariants that characterize those conditions required to hold throughout the evolution of the target model.

P2 - Operational correspondence. Operations are interpreted as state-transition units with explicit enabling conditions and update effects. In the Event-B interpretation, the enabling conditions of an operation are reflected as guards, whereas its effects are reflected as the actions of the corresponding event. In this way, the behavioral meaning of operations is brought into the target model in a form suitable for state-based interpretation and analysis.

P3 - Obstacle-handling correspondence. Obstacle-related handling behavior is not treated as an auxiliary annotation during interpretation, but is made explicit through hindering conditions, non-nominal paths, or handling events. This ensures that the target model reflects not only nominal behavior, but also those structures related to obstruction, recovery, or redirected execution whenever they fall within the scope of the considered fragment.

P4 - Assumption-aware mapping. Domain assumptions are carried into the target model as background conditions of the interpretation. Depending on their nature, these assumptions may be realized as axioms in the *Context*, or as invariants and guards in the *Machine*. Consequently, any results obtained from the formal model must be understood under these explicitly represented domain assumptions.

P5 - Refinement mapping. AND/OR refinement is preserved in terms of decomposition logic rather than graphical structure. In the AND case, the target model reflects joint necessity; in the OR case, it reflects alternative satisfaction paths. Refinement is therefore interpreted as a logical constraint structure over invariants, guards, and event structures.

P6 - Temporal coverage. In the temporal interpretation, temporal requirements are interpreted as LTL-based specifications in a SpecPro-oriented style. These requirements are not constructed as a layer detached from the state-based part of the model. Instead, they rely on the same state basis already made explicit in the Event-B interpretation. Accordingly, obligations such as eventuality, response, ordering, invariance, and assumption are expressed as temporal formulae over the milestones, states, and behavioral conditions already established in the state-based interpretation.

Taken together, these properties define the principal anchors between the source model and the formal artifacts generated by the two interpretations. On that basis, Section 6 does not re-establish the interpretation rules at the conceptual level, but instead evaluates two more specific issues: the extent to which these rules cover the construct groups of gStar considered in the evaluation, and the extent to which those artifacts support formal analysis.

5. Case Study and Tool Support

This section presents the two case studies used in the paper, together with the corresponding tool support. The first case study is MinePump, which is developed in gStar on the basis of a corresponding MinePump case previously used in a study on transformation from GORE to Event-B [19], and is used for comparison with the Event-B interpretation. MeetingScheduler is the main case study, used to illustrate the proposed approach more fully, from the source model and temporal requirements to the generated formal artifacts. This section also presents the accompanying tool support, focusing on the main choices involved in model specification, transformation, and artifact generation.

5.1. Case Study 1: MinePump

MinePump is used as an auxiliary case study to support comparison with the Event-B interpretation. The case is developed in gStar from a corresponding case study previously used in research on transformation from GORE to Event-B [19]. In this recasting step, the goal and actor structure of the original case is preserved at the corresponding level: goals expressing safety maintenance are represented as *maintain* goals in gStar, goals expressing the achievement of a state or condition are represented as *achieve* goals, and those elements that are already described in operational terms are represented as *operations* so that they support the behavioral interpretation. The main agents, including `PumpController`, `AlarmController`, `HighWaterSensor`, `LowWaterSensor`, and `MethaneSensor`, are also retained so as to preserve the source-level distribution of responsibilities.

Figure 10 shows the resulting MinePump model after recasting it into gStar. In this model, `SafeMine`, `NoFlooding`, `NoPumpDamaged`, and `NoExplosion` are represented as maintain goals; `HighWaterDetected`, `LowWaterDetected`, and `MethaneDetected` are represented as achieve goals; and `PumpTurnOn`, `PumpTurnOff`, `PumpOffMethane`, and `AlarmOnMethane` are represented as operations. The purpose of this reconstruction is not to reproduce the earlier model at the notation level, but to preserve its core safety logic and domain content within the set of source-level constructs supported by the current gStar metamodel.

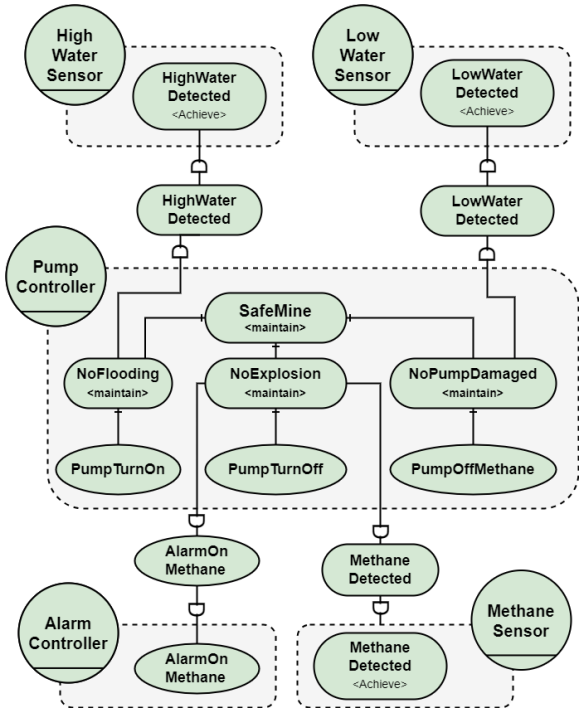


Figure 10. MinePump recast in gStar.

5.2. Case Study 2: MeetingScheduler

MeetingScheduler is the main case study of the paper. As introduced in Section 2, it is used to present the proposed approach more fully in terms of the source model, the temporal requirements, and the generated formal artifacts. Figure 11 depicts the gStar model of MeetingScheduler, organized around the central goal `Meeting be scheduled`, together with the goals and operations related to request registration, availability collection, candidate-slot

proposal, slot confirmation, room reservation, and participant notification. The model also makes explicit the relevant actors and external services, including `Meeting Initiator`, `Participant`, `Calendar Service`, `Notification Service`, and `Room Booking Service`, as well as domain assumptions and obstacles such as `Room unavailable`, `No participant response`, `Calendar service down`, and `No common slot`.

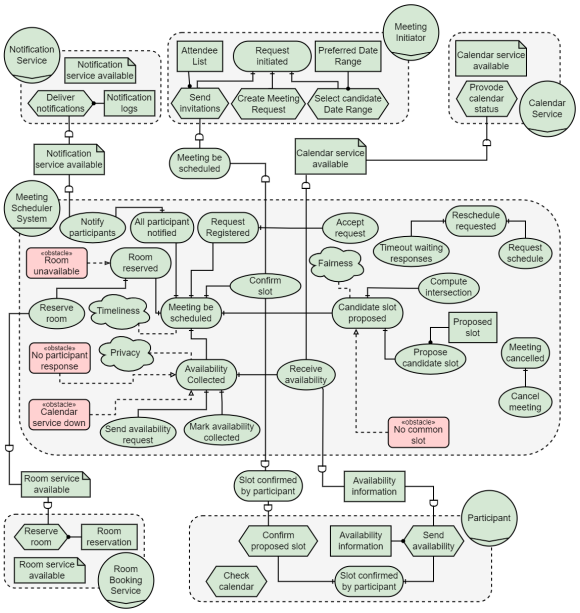


Figure 11. MeetingScheduler model in gStar.

At the modeling level, MeetingScheduler is constructed with both goal-oriented constructs and temporal-requirement constructs. Goals such as `Meeting be scheduled`, `Request Registered`, `Availability Collected`, `Candidate Slot Proposed`, `Room Reserved`, and `All participant notified` make explicit the important states and milestones of the system, whereas operations such as `Accept request`, `Reserve room`, `Notify participants`, `Confirm slot`, `Request schedule`, `Propose candidate slot`, and `Cancel meeting` carry the behavioral meaning of the model. On this basis, MeetingScheduler provides a suitable case for illustrating how the source model is organized simultaneously in terms of state, behavior, dependency, and temporal concerns, thereby serving as input to both the Event-B interpretation and the temporal interpretation of the paper.

5.3. Tool Support

The realization of the proposed approach is organized as a toolchain based on Eclipse/EMF,[5] in which the gStar model is kept as the common input to both the Event-B interpretation and the temporal interpretation. This organization ensures that the resulting formal artifacts are derived from a shared source-model basis rather than from separate processes. In this paper, the case studies are specified according to the `gstar.ecore` metamodel, including goal-oriented elements, behavioral elements, and temporal-requirement constructs when present. From this model, the Event-B and temporal parts are realized as two different artifact-generation paths that nevertheless originate from the same representational basis.

For Event-B artifact generation, the realization comprises a model-to-model step followed by a model-to-text step. First, the gStar model is transformed into a target model conforming to `eventb.ecore`, through which the source structures related to state, behavior, and constraints are brought into the corresponding Event-B model elements. The textual Event-B artifacts are then generated from this target model. For temporal-artifact generation, the realization proceeds from the temporal-requirement constructs of gStar to the corresponding temporal artifacts, also through model-to-text generation, in order to produce the relevant temporal formulae and specifications from `TemporalVariable`, `TemporalRequirement`, and `TemporalLink`.

At the technological level, Eclipse/EMF is used to manage the metamodels and models. ATL supports the model-to-model transformation from gStar to Event-B, whereas Acceleo is used

[5]https://github.com/laviet2208/gstar2eventb

to generate the textual artifacts for both parts. The resulting artifacts are then analyzed using Rodin and ProB. Rodin provides the environment for handling proof obligations and checking the consistency of the Event-B models, while ProB supports model checking, state-space exploration, and related analysis tasks. The toolchain realizes the rule families described in Section 4 for the implemented fragment. Generated Event-B artifacts are subject to Rodin/ProB analysis, while the mapping itself has not been independently proved correct for every possible gStar input. The semantic schemata in Section 4.5 therefore serve as correctness conditions against which future generator-level testing and proof can be developed.

6. Evaluation

This section presents the evaluation of the proposed approach on the two case studies introduced in Section 5. The evaluation focuses not only on the ability to generate formal artifacts from the source model, but also on the extent to which these artifacts support analysis in the two interpretations established earlier, namely the Event-B interpretation and the temporal interpretation. On this basis, the section first defines the evaluation questions, then describes the evaluation setup, presents the results, and finally discusses the significance and limitations of the obtained findings.

6.1. Research Questions

This study investigates the extent to which the proposed approach can realize the formal interpretations established from gStar models while producing artifacts that are meaningful for formal analysis. To guide this investigation, we consider the following two research questions:

- **RQ1 (explicit representation and traceability).** For a concrete case study, which state-related, behavioral, and temporal concerns remain implicit in its original actor–goal view, and which can be explicitly represented in gStar and traced to Event-B/LTL artifacts?
- **RQ2 (analyzability of the added information).** For the generated artifacts, what evidence supports formal analysis of state invariants, operational behavior, and temporal response/order properties, and what does that evidence not establish?

RQ1 directly addresses the lack of explicit state, behavioral, and temporal representation identified in the abstract; RQ2 addresses whether these additional representations can be examined with formal tools. The questions are formulated for a concrete modeled situation and instantiated in the two selected case studies; their results are not universal claims about all gStar models.

6.2. Case Study Design

The evaluation is designed around two case studies that play complementary roles. MinePump is used as a baseline case for comparison with the Event-B interpretation. As this is a well-known problem in GORE and Formal Methods research, it provides a suitable point of reference for examining whether the artifacts generated from gStar preserve the core state-based and behavioral logic of the problem. Accordingly, MinePump mainly serves the comparison purpose in the Event-B interpretation rather than illustrating the full scope of the proposed approach. By contrast, MeetingScheduler is used as the main case study for observing the proposed approach on a model that simultaneously covers state, behavior, and temporal concerns. The combination of these two cases provides both a relatively stable comparison point for the Event-B interpretation and a sufficiently rich setting for observing how the source model is organized and brought into both interpretations considered in the paper.

At the operational level, the evaluation is conducted on a pipeline based on gStar models as

inputs. Specifically, the case studies are specified as XMI models conforming to the `gstar.ecore` metamodel. From there, ATL supports the model-to-model transformation from gStar to the target models, while Acceleo performs the generation of textual artifacts. The outputs of this process consist of two main groups: (i) Event-B artifacts, including context and machine, which are used in Rodin for handling proof obligations and in ProB for behavior animation; and (ii) temporal artifacts generated from the temporal-requirement constructs of gStar as the corresponding LTL formulae and specifications. With this organization, the evaluation does not only consider the outputs produced by the pipeline, but also observes the correspondence from the source model to the generated artifacts, as well as their value for formal analysis.

6.3. Evaluation Criteria and Formal Witnesses

Building on the interpretation rules and semantic properties established in Section 4, this subsection defines the criteria used to evaluate the generated formal artifacts. The purpose here is not to restate the semantics of the proposed framework, but to identify the formal witnesses through which the results for RQ1 and RQ2 are assessed.

For the Event-B interpretation, the evaluation is grounded in explicit target-side witnesses such as invariants, events, guards, actions, axioms, and decomposition-related structures. For the temporal interpretation, the evaluation is grounded in the LTL-based artifacts generated from temporal-requirement constructs, following Property Specification Patterns (PSP) and the SpecPro-oriented style [20, 24]. RQ1 combines a qualitative comparison against the original actor–goal viewpoint with a traceability check in the generated artifacts; RQ2 examines property-specific proof/model-checking witnesses. This separates expressiveness evidence from tool-verification evidence.

Table 5 summarizes the evaluation properties, their corresponding formal witnesses, and the way they are used in the assessment.

These properties do not constitute an independent semantic proof. Rather, they provide an operational evaluation basis for examining whether the generated artifacts make the relevant construct groups of the considered gStar fragment explicit and whether these artifacts can be used in subsequent formal analysis.

6.4. Results

The results are organized by the two evaluation questions. For RQ1, the comparison concerns explicitly available modeling information, not measured user effort or a controlled statistical comparison. For RQ2, reported Rodin/ProB findings are unchanged from the original experiments and are restricted to the analyzed configurations.

6.4.1. Explicit Representation and Construct-Group Coverage (RQ1)

RQ1 is assessed in two complementary ways: a qualitative comparison of what the models explicitly contain, followed by source-to-target traceability evidence. In the original iStar-style MeetingScheduler view described in Section 2, actors, goals, tasks and dependencies are directly represented, but operation guards, state updates and explicit LTL obligations are not. The enriched gStar model makes these additional concerns first-class specifications. This observation concerns the visible notations of these models; it does not prove that no other iStar extension could express such concerns.

For *MinePump*, coverage is examined over two construct groups, namely state-based and behavioral constructs, since this case study does not include explicit temporal-requirement constructs in the source model. As shown in Table 7, representative safety goals are reflected as state constraints in the generated machine, whereas representative control actions

Table 5. Evaluation properties and formal witnesses.

Property	**Formal witness**	**Assessment focus**
P1. Maintain-goal preservation	Invariants in the generated *Machine*	Used to examine state-based coverage in RQ1 and invariant preservation through proof obligations in RQ2.
P2. Operational correspondence	Events with guards and actions	Used to examine the mapping of operations in RQ1 and the executability of the generated behavioral model in RQ2.
P3. Obstacle-handling correspondence	Hindering guards, non-nominal paths, or handling events	Used to examine the explicit representation of obstacle-related behavior in RQ1 and the handling of non-nominal scenarios in RQ2.
P4. Assumption-aware mapping	Axioms, invariants, or guards	Used to examine whether assumptions are made explicit in RQ1 and how they constrain analysis in RQ2.
P5. Refinement mapping	Layered invariants, guards, and event structures	Used to examine whether decomposition logic remains visible in the generated artifacts for RQ1 and RQ2.
P6. Temporal coverage	LTL-based specifications generated from temporal-requirement constructs	Used to examine temporal construct coverage in RQ1 and temporal model-checking support in RQ2.

Table 6. Explicit information in the examined MeetingScheduler representations (RQ1).

Aspect	**Original iStar viewpoint**	**gStar / formal witness**
State	Goal labels and qualities; no typed state-update semantics	Goal predicates/milestones; Event-B variables and invariants
Behavior	Intentional tasks and dependencies; execution conditions implicit	Operations with guards and effects; Event-B events
Time	Coordination order implicit; no source-level LTL pattern	Linked temporal requirements; LTL response/ordering formulae

and environment-triggered behaviors are reflected as corresponding Event-B events. Although the table reports only representative mappings, the generated model shows that the main construct groups of *MinePump* are covered in the Event-B artifacts.

For *MeetingScheduler*, the coverage scope extends to temporal constructs in addition to state-based and behavioral ones. Table 8 shows that representative milestones and operational steps are reflected as corresponding elements in the Event-B machine. Here again, the listed mappings are representative rather than exhaustive, but they indicate that the main state-based and behavioral construct groups of the source model are carried into the Event-B artifacts according to the same transformation logic.

Table 7. Representative mappings for MinePump.

Construct group	**Source element**	**Generated artifact**
State-based constructs	`NoExplosion`	`explosion_cond = safe`
State-based constructs	`NoFlooding`	`flood_cond = safe`
Behavioral constructs	`PumpTurnOn`	`op_pump_turn_on`
Behavioral constructs	`PumpOff Methane`	`op_pump_off_methane`
Behavioral constructs	`Methane Detected`	`ENV_METHANE_DETECTED`

Table 8. MeetingScheduler mappings.

Group	Source element	Generated artifact
State	`Meeting Scheduled`	`scheduled`
	`Availability Collected`	`availCollected`
Behavior	`ProposeSlot`	`op_sys_propose_slot`
	`Notify Participants`	`op_sys_notify`
	`NoResponse / Reschedule Handling`	`op_sys_request_reschedule`

Table 9. Temporal mappings.

Source element	Generated artifact
`TemporalVariable`	proposition `scheduled`
`TemporalRequirement` (response)	`G(requestReceived -> F(scheduled))`
`TemporalRequirement` (ordering)	corresponding LTL ordering formula
`TemporalLink`	traceability links in temporal artifacts

Unlike *MinePump*, *MeetingScheduler* also includes explicit temporal-requirement constructs in the source model. Therefore, coverage must also be examined for temporal constructs. Table 9 confirms that these constructs are reflected on the target side, from propositional variables to representative temporal patterns. In this sense, the temporal-requirement constructs are not omitted during model realization, but are mapped into corresponding LTL-based artifacts.

Taken together, Tables 7, 8, and 9 show that the proposed semantic rules cover all construct groups of the considered gStar fragment included in the evaluation. In *MinePump*, the covered groups are state-based and behavioral. In *MeetingScheduler*, the covered groups include state-based, behavioral, and temporal constructs. Within this scope, the mapping tables show a witness in the generated artifacts for each of the three *examined* concern categories in MeetingScheduler, and two in MinePump. This category-level observation is not a quantitative claim that every instance or every possible construct has been translated correctly, nor is it a direct proof of semantic preservation. It provides the structural basis for RQ2.

6.4.2. Generated Artifacts as a Basis for Formal Analysis (RQ2)

The objective of this research question is to evaluate the extent to which the generated formal artifacts support the analysis and verification of both state-based and temporal properties defined in the source model. Following the evaluation criteria and witnesses established in Section 6.3, the results are analyzed across three primary dimensions.

Static Logic Verification. The verification results obtained in Rodin demonstrate the consistency of the generated artifacts with respect to the system's static constraints. For the *MinePump* case study, 24 Proof Obligations (POs) were generated, whereas *MeetingScheduler* produced 46 POs. These POs are associated primarily with invariant preservation and event consistency, directly addressing criteria **P1** and **P2**. An automatic discharge rate of **100%** was achieved for both studies, with zero interactively discharged POs. This result indicates that the generated Event-B structures are well-formed and suitable for automated proof support in Rodin.

Behavioral Consistency and Model Checking. Using the ProB model checker, we assessed the behavioral consistency of the target models within the explored state space. For *MinePump* (48 states, 96 transitions) and *MeetingScheduler* (312 states, 864 transitions), the verification process found no deadlocks and no invariant violations. This provides evidence that the explored executions remain consistent with the declared safety constraints in the checked configurations, and suggests that the mappings for obstacle handling (**P3**) and domain assumptions (**P4**) are consistently realized in the generated models. The observed

state spaces also indicate that the generated models remain amenable to practical verification in the considered case studies.

Temporal Consistency. The analysis of temporal properties was conducted by checking LTL formulae derived from the temporal-requirement constructs of *MeetingScheduler*. The four reported LTL formulae, representing progress and ordering patterns, were verified as **TRUE** under the model and checker settings used. Key properties, such as the response obligation (`G(requestReceived -> F(scheduled))`) and ordering constraints derived from these temporal-requirement constructs, were satisfied (**P6**). These results provide evidence that the generated temporal artifacts remain consistent with the state basis established in the Event-B model.

Table 10. Technical statistics for formal analysis (RQ2).

Evaluation Metric	***MinePump***	***Meeting Scheduler***
Total Proof Obligations (POs)	24	46
Automatically Discharged	100% (24/24)	100% (46/46)
Interactively Discharged	0	0
State Space (States, Transitions)	48, 96	312, 864
Deadlock Check	No deadlock	No deadlock
Invariant Violations Found	None	None
Number of LTL Formulae	N/A	4
LTL Verification Result	N/A	TRUE: 4

In summary, the results show that the proposed framework provides a workable basis for formal analysis. The generated artifacts can be analyzed with Rodin and ProB for both state-based and temporal properties, thereby supporting the early detection of logical inconsistencies and sequencing issues.

Interpretation relative to the abstract's claims. The RQ1 comparison (Table 6) directly connects the limitations identified in the abstract to explicit modeling mechanisms: state conditions are represented by predicates and milestone variables; operational behavior is represented by guarded transitions; and temporal obligations are represented by linked formulae. The RQ2 results support the *analyzability* of these mechanisms in the selected models, not a demonstrated increase in industrial requirements quality, a general comparison of expressiveness against all iStar variants, or a proof of semantic equivalence. In particular, 100% automatic discharge of the reported proof obligations checks the generated Event-B invariants; it does not certify that the initial stakeholder goals were correctly elicited. Likewise, four satisfied LTL formulae do not establish the completeness of the temporal requirements set. A controlled evaluation of modeling effort, independent trace checking, and toolchain-level transformation verification remain future work.

7. Threats to Validity

This section discusses factors that may affect the interpretation of the reported results, particularly with respect to the robustness of the evaluation basis and the degree to which the empirical evidence can support broader generalization. In the present study, two points deserve particular attention: the set of properties P1–P6 used to organize the evaluation and the number of case studies currently included in the empirical part.

Threat 1: Robustness of the evaluation-property basis. The evaluation is organized around the six properties P1–P6 in order to assess both structural coverage and analytical support of the generated formal artifacts. This design provides an explicit and internally consistent evaluation basis that is directly anchored in target-side formal witnesses, thereby giving a clear foundation for answering RQ1 and RQ2. At the same time, however, P1–P6 remain a

set of properties defined within the present study rather than a standardized or broadly established evaluation scheme for all approaches to the formalization of goal-oriented requirements models. Consequently, although this property set is well aligned with the objectives and scope of the paper, it may not capture all dimensions that could be relevant in other semantic frameworks or application contexts. This threat is partly mitigated by stating the properties explicitly, relating them to concrete formal witnesses, and applying them consistently throughout the evaluation process. Nevertheless, the current results should primarily be understood as evidence for the suitability of the evaluation basis adopted in this paper, rather than as evidence for the universal completeness of that property set.

Threat 2: Limited scope of the empirical basis. The empirical evaluation is currently based on two case studies only, namely *MinePump* and *MeetingScheduler*. This choice is deliberate, since the two cases play complementary roles in the assessment: *MinePump* mainly supports observation of the Event-B interpretation, whereas *MeetingScheduler* enables the simultaneous examination of state-based, behavioral, and temporal aspects. As a result, the evaluation does not stop at checking whether formal artifacts can be generated at the structural level, but also considers whether they support analysis in two settings with different characteristics. Even so, from the standpoint of generalization, two case studies still constitute a relatively narrow empirical basis for drawing broader conclusions about the full space of goal-oriented requirements problems. More precisely, the present results provide evidence that the proposed approach is feasible, that the generated artifacts are amenable to formal analysis, and that the proposed semantic framework is workable in the settings examined. They are not yet sufficient to justify stronger claims about generality across larger models, other application domains, or richer gStar fragments. This threat is mitigated to some extent by the use of two case studies with different characteristics and different evaluative roles, but additional studies on further models and domains would still be needed to strengthen the generalizability of the conclusions.

Threat 3: Translation correctness and temporal assumptions. The source-to-target equations are formal specification criteria, not automatically discharged correctness theorems for the ATL/Acceleo implementation. The existing results assess the generated artifacts, so a translation defect could remain undetected even if the resulting model passes its internal checks. Moreover, liveness properties depend on finite-state abstraction, environment assumptions, and possible fairness restrictions. Without documented fairness and scheduling assumptions, a result about a response property should not be generalized to all operational schedules. A rigorous simulation proof, differential checking against source traces, and a documented fairness configuration are needed before stronger semantic-preservation claims can be made.

8. Related Work

This work is related to four strands of research: goal modeling and expressiveness in GORE, formalization of iStar and related goal-oriented models, Event-B-oriented derivation from goal models, and the use of temporal logic and structured temporal patterns in requirements analysis.

Goal modeling and expressiveness in GORE. iStar is a widely used GORE framework for representing stakeholders' intentions, actor dependencies, and rationale structures during early requirements analysis [5, 22]. Its strength lies in capturing why particular requirements matter and how actors depend on one another. However, the core language remains largely declarative and centered on the actor–goal perspective. As a result, it does not directly provide constructs for representing behavioral concerns in terms of

states and transitions, nor for specifying temporal requirements in a structured and traceable manner.

Several studies have extended goal models to capture concerns that are not fully covered by the core notations. FLAGS, for example, introduces fuzzy goals to support requirements-driven adaptation under uncertainty [11]. Other extensions of iStar/GRL for collaborative cyber-physical systems enrich modeling support for settings involving coordination needs and environmental dependencies [12]. These studies show that richer modeling constructs are often needed when goal models describe systems operating under constraints or uncertainty. Recent syntheses of GORE identify both continuing uses of goal models and a need for stronger empirical assessment [1]; recent work on LLM-assisted goal model generation illustrates a different, complementary direction for improving modeling support [32].

Nevertheless, increased modeling expressiveness still leaves open the question of how the added constructs should be given precise meanings for analysis. Many extensions improve representation power, but do not systematically relate the resulting constructs to analyzable state-based or temporal artifacts. The present work addresses this gap by introducing *gStar*, an extension of iStar that enhances the expressiveness of goal models with operations, state-relevant goals, domain assumptions, obstacles, and temporal requirements. On this modeling basis, the paper then defines how these constructs are interpreted in Event-B and LTL as complementary formal domains.

Formalization of iStar and related goal-oriented models. A separate line of research has focused on increasing the formal rigor of iStar and related goal-oriented models by translating them into established formal languages. Some approaches map iStar to Z to obtain mathematical specifications for analyzing constraints and consistency [8]. Others use Alloy to support model checking and counterexample generation with respect to goal-model constraints [9]. Ontology- and description-logic-based representations, such as iStar2.0-OWL, aim to standardize conceptual knowledge and support knowledge-level reasoning [10].

These approaches provide important foundations for structural consistency, conceptual clarity, and static reasoning. However, their focus is mainly on structural or conceptual formalization. They do not primarily address how goal-model constructs should be given a semantic account that connects them to both state-based behavior and temporal properties. In contrast, the present work treats formalization as a subsequent step built on the enriched modeling basis provided by *gStar*.

Goal models and Event-B. Several studies have used goal-oriented requirements models to guide the construction of Event-B specifications. Recent requirements-automation reviews reinforce the need to distinguish artifact-generation success from evidence of requirements correctness [2]. Aziz et al. propose a bridge from goal-oriented requirements to Event-B specifications, showing how goals can inform the derivation of formal models [16]. Matoussi et al. present a goal-based approach for guiding the design of abstract Event-B specifications [17]. Ponsard and Devroey generate high-level Event-B system models from KAOS requirements models [18]. More recently, Saharudin and Said define ATL-based transformation rules from KAOS to Event-B within a model-transformation workflow [19].

These studies demonstrate the value of connecting goal-oriented requirements to Event-B. However, many are based on KAOS rather than iStar, or focus mainly on deriving Event-B target models. Since KAOS provides a stronger basis for operationalization, whereas iStar centers on intentions, dependencies, and rationale, the present work starts from iStar, extends it into *gStar* to improve the expressiveness of goal models, and then defines a formal semantic framework relating the resulting constructs to Event-B artifacts and

LTL-based temporal artifacts.

Event-B, temporal logic, and structured temporal requirements. Event-B provides a state-based formal domain centered on invariants, refinement, and proof obligations [13, 14]. Rodin supports proof-obligation generation and discharge, while ProB supports animation, model checking, deadlock detection, and property checking over Event-B models [14, 15]. These characteristics make Event-B suitable for representing states, transitions, and safety-related constraints. Importantly, Event-B also has dedicated approaches for temporal analysis and trace preservation, including fairness-aware liveness refinement [29], trace-refinement checking [30], and newer work on failure-divergence refinement [31]. These approaches differ from the present choice to produce separately traceable structured LTL requirements; hence, the use of a separate LTL artifact is not motivated by an alleged absence of temporal verification tools for Event-B.

Beyond state-based reasoning, Linear Temporal Logic (LTL) provides a classical basis for expressing properties over execution traces [6, 7, 23]. In requirements engineering, however, the practical use of LTL depends on a clear correspondence between requirements, behavioral milestones, and the propositions over which temporal formulae are interpreted. For this reason, the present work uses a SpecPro-oriented style for specifying temporal requirements in *gStar*, while LTL serves as the semantic domain for these properties. By relying on property specification patterns, the approach connects structured temporal requirements with their corresponding LTL representations while preserving traceability to the temporal-requirement constructs introduced in *gStar*.

Summary. In summary, prior work has provided important foundations for enriching goal models and for connecting requirements models to formal domains. The present work differs by first introducing *gStar* as an extension of iStar for enhancing the expressiveness of goal models through constructs for operations, state-relevant goals, domain assumptions, obstacles, and temporal requirements. On this basis, it defines a formal semantic framework based on Event-B and LTL, and realizes this framework through a traceable MDE process that links *gStar* constructs to generated formal artifacts.

9. Conclusion and Future Work

This paper has presented *gStar* as an extension of iStar for enhancing the expressiveness of goal-oriented requirements models within the GORE methodology. This extension introduces structured constructs for operations, state-relevant goals, domain assumptions, obstacles, and temporal requirements, allowing goal models to capture richer system-related information while remaining rooted in stakeholders' intentions, dependencies, and rationale. On this modeling basis, the paper defines a translation-based, precisely scoped formal interpretation for *gStar*, in which Event-B serves as the semantic domain for state-based and behavioral aspects, while Linear Temporal Logic (LTL) serves as the semantic domain for temporal properties specified in a SpecPro-oriented style.

From a methodological perspective, the contribution extends beyond the definition of the semantic framework itself by realizing it through a tool-supported model-driven engineering (MDE) process. This process transforms *gStar* models into traceable Event-B artifacts and corresponding temporal artifacts, while recording traceability links between source-level constructs and their generated formal representations. In this way, the transformation is treated not merely as a technical translation, but as a testable realization of the stated interpretation rules, helping to maintain the relationship between intentional requirements and their formal analysis artifacts.

The experimental results, evaluated through

the *MinePump* and *MeetingScheduler* case studies, support the feasibility of the proposed approach. The generated Event-B models were analyzed using Rodin and ProB, achieving a high rate of automatic discharge for proof obligations in the considered scenarios, while the temporal artifacts supported the analysis of safety-, progress-, and response-related properties. These results provide evidence that the generated formal artifacts can help identify logical and behavioral issues, such as invariant violations or deadlocks, as well as temporal issues related to progress and response obligations. Thus, the evaluation suggests that the proposed approach can support early requirements validation through formal analysis.

Several directions remain for future work. First, the semantic scope and mapping coverage should be extended to include additional iStar/*gStar* constructs and more complex actor dependencies. Second, the representation and interpretation of recovery-oriented behavior should be strengthened to better address environmental uncertainty and non-nominal execution scenarios. Furthermore, increasing the automation of temporal-artifact construction and conducting evaluations on more diverse case studies would help assess the generality and practical usefulness of the approach. Finally, an implementation-level proof of trace correspondence, combined with broader case studies and explicit fairness assumptions, is needed before asserting general semantic preservation for larger fragments of *gStar*.

References

[1] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, P. Giorgini, A systematic mapping of goal-oriented requirements engineering: Reflections, impact, and the way forward, Requirements Engineering, Vol. 31, 2026, pp. 20. doi:10.1007/s00766-026-00476-7.

[2] M. A. Umar, K. Lano, Advances in automated support for requirements engineering: A systematic literature review, Requirements Engineering, Vol. 29, 2024, pp. 177–207. doi:10.1007/s00766-023-00411-0.

[3] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley, 2009.

[4] A. van Lamsweerde, Goal-oriented requirements engineering: A guided tour, in: Proc. 5th Int. Symp. Requirements Engineering (RE), IEEE Computer Society, 2001, pp. 249–262.

[5] F. Dalpiaz, X. Franch, J. Horkoff, iStar 2.0 language guide (2016). arXiv:1605.07767, doi:10.48550/arXiv.1605.07767.

[6] A. Pnueli, The temporal logic of programs, in: Proc. 18th Annu. Symp. Foundations of Computer Science (FOCS), IEEE Computer Society, 1977, pp. 46–57. doi:10.1109/SFCS.1977.32.

[7] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.

[8] L. Kadakolmath, U. D. Ramu, iStar goal model to Z formal model translation and model checking of CBTC moving block interlocking system, Formal Aspects of Computing, Vol. 36, No. 1, 2024, pp. 6:1–6:45. doi:10.1145/3633065.

[9] P. O. Ating'a, A. Krishna, Verification of i* models using Alloy, in: Proc. Information Systems Development: Challenges in Practice, Theory, and Education, Springer, 2010, pp. 63–74.

[10] C. Almendra, C. Silva, V. E. S. Souza, R. Guizzardi, C. Bernabé, iStar 2.0-OWL: An operational ontology for iStar, in: Proc. 12th Int. i* Workshop (iStar), Vol. 2490 of CEUR Workshop Proceedings, CEUR-WS.org, 2019.

[11] L. Baresi, L. Pasquale, P. Spoletini, Fuzzy goals for requirements-driven adaptation, in: Proc. 18th Int. Requirements Engineering Conf. (RE), IEEE Computer Society, 2010. doi:10.1109/RE.2010.25.

[12] M. Daun, J. Brings, L. Krajinski, V. Stenkova, T. Bandyszak, A GRL-compliant iStar extension for collaborative cyber-physical systems, Requirements Engineering, Vol. 26, No. 3, 2021, pp. 325–370. doi:10.1007/s00766-021-00347-3.

[13] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010. doi:10.1017/CBO9781139195881.

[14] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, L. Voisin, Rodin: An open toolset for modelling and reasoning in Event-B, Software Tools for Technology Transfer, Vol. 12, No. 6, 2010, pp. 447–466. doi:10.1007/s10009-010-0145-y.

[15] M. Leuschel, M. J. Butler, ProB: A model checker for B, in: FME 2003: Formal Methods, Vol. 2805 of LNCS, Springer, 2003, pp. 855–874. doi:10.1007/978-3-540-45236-2_46.

[16] B. Aziz, A. Arenas, J. Bicarregui, C. Ponsard, P. Massonet, From goal-oriented requirements to Event-B specifications, in: Proc. 1st NASA Formal Methods Symp. (NFM), NASA, 2009, pp. 96–105.

[17] A. Matoussi, F. Gervais, R. Laleau, A goal-based approach to guide the design of an abstract Event-B specification, in: Proc. 16th Int. Conf. Engineering of Complex Computer Systems (ICECCS), IEEE Computer Society, 2011, pp. 139–148. doi:10.1109/ICECCS.2011.21.

[18] C. Ponsard, X. Devroey, Generating high-level Event-B system models from KAOS requirements models, in: Actes du XXIXème Congrès INFORSID, 2011, pp. 317–332.

[19] S. N. Saharudin, M. Y. Said, Keep all objectives satisfied (KAOS) to Event-B models transformation, International Journal on Advanced Science, Engineering and Information Technology, Vol. 12, No. 6, 2022, pp. 2443–2451. doi:10.18517/ijaseit.12.6.15875.

[20] S. Vuotto, M. Narizzano, L. Pulina, A. Tacchella, Automata based test generation with SPECPRO, in: Proc. 6th Int. Workshop Requirements Engineering and Testing (RET), IEEE, 2019, pp. 1–8. doi:10.1109/RET.2019.00010.

[21] H.-V. L. Trinh, D.-H. Dang, Transforming operationalised iStar models into temporal semantics and Event-B for consistency assurance, in: Proc. 17th Int. Conf. Knowledge and System Engineering (KSE), IEEE, 2025. doi:10.1109/KSE68178.2025.11309478.

[22] E. S. K. Yu, Towards modelling and reasoning support for early-phase requirements engineering, in: Proc. 3rd Int. Symp. Requirements Engineering (RE), IEEE Computer Society, 1997, pp. 226–235. doi:10.1109/ISRE.1997.566873.

[23] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT Press, 1999.

[24] M. B. Dwyer, G. S. Avrunin, J. C. Corbett, Patterns in property specifications for finite-state verification, in: Proc. 21st Int. Conf. Software Engineering (ICSE), ACM, 1999, pp. 411–420. doi:10.1145/302405.302672.

[25] M. Narizzano, L. Pulina, A. Tacchella, S. Vuotto, Consistency of property specification patterns with boolean and constrained numerical signals, in: NASA Formal Methods, Vol. 10811 of LNCS, Springer, 2018, pp. 383–398. doi:10.1007/978-3-319-77935-5_26.

[26] M. Narizzano, L. Pulina, A. Tacchella, S. Vuotto, Property specification patterns at work: Verification and inconsistency explanation, Innovations in Systems and Software Engineering, Vol. 15, 2019, pp. 307–323. doi:10.1007/s11334-019-00339-1.

[27] M. E. Bratman, Intention, Plans, and Practical Reason, Harvard University Press, 1987.

[28] F. Jouault, I. Kurtev, Transforming models with ATL, in: Satellite Events at the MoDELS 2005 Conference, Vol. 3844 of LNCS, Springer, 2006, pp. 128–138. doi:10.1007/11663430_14.

[29] C. Zhu, M. Butler, C. Cirstea, T. S. Hoang, A fairness-based refinement strategy to transform liveness properties in Event-B models, Science of Computer Programming, Vol. 225, 2022, pp. 102907. doi:10.1016/j.scico.2022.102907.

[30] S. Stock, A. Mashkoor, M. Leuschel, A. Egyed, Trace preservation in B and Event-B refinements, Journal of Logical and Algebraic Methods in Programming, Vol. 137, 2024, pp. 100943. doi:10.1016/j.jlamp.2024.100943.

[31] S. Stock, M. Leuschel, A. Mashkoor, Failure divergence refinement for Event-B, arXiv preprint arXiv:2505.19069 (2025). arXiv:2505.19069. URL https://arxiv.org/abs/2505.19069

[32] Y. Hirabayashi, M. Saeki, Leveraging an LLM to generate goal models, IEICE Transactions on Information and Systems, Vol. E109.D, No. 9, 2026, pp. 1341–1357. doi:10.1587/transinf.2025KBP0001.