Volume 66, Issue 2, 2022
Journal of Scientific Research
of
The Banaras Hindu University

i*-Based Goal-Oriented Modeling and Requirements Specification of an Urban Railway Interlocking System

Lokanna Kadakolmath[1][0000-0001-8500-9431] and Umesh D R[2][0000-0002-6813-3876]

[1]Acharya Institute of Technology, Bengaluru 560 107, Karnataka, India
[2]P.E.S College of Engineering, Mandya 571 401, Karnataka, India
[1, 2]Visvesvaraya Technological University, Belagavi 590 018, Karnataka, India
[1]lokanna.kadakolmath@gmail.com
[2]drumesh@pesce.ac.in

Abstract. Urban railway interlocking system is a safety-critical system, for which interlocking rules are well-defined by international standards to assure safe operations. However, it is very difficult to extract all of the safety requirements in the requirements engineering stage itself and construct a goal model as well. One approach to fulfill this goal is to extract safety requirements from regulations and analyze them at the early stage of the model. Improper specifications regarding the environs of an interlocking system are realized to be accountable for any faults in requirements specifications. Furthermore, non-functional requirements are kept aside from requirements specifications. To make an interlocking system operate in an ultra-dependable range we need to address variability issues, so in this paper, we used i*-based goal-oriented requirements language called TGRL to address variability issues as well as blend early and late requirements. The developed model can also be visualized and analyzed using jUCMNav. Finally, the safety requirements to be verified are specified with Formal Tropos, a formal specification language for the i* model.

Keywords: Early Requirements, Formal Specification, Goal Modeling, Model Checking, Railway Interlocking, Requirements Elicitation, Safety-critical Systems, Smart Mass Transportation, Verification.

1 Introduction

To assure that a software program accurately resolves a specific problem, we should first properly understand and describe what problem requirements to be solved. This looks like common sense at first sight. But, as we will see, identifying what the precise problem is can be hard. We must determine, grasp, convey, examine, and decide on what problem would be solved, why such a problem has to be solved, and who should be required in the accountability of resolving that problem. Generally, this process is called requirements engineering [1].

Many studies [2], [3] have demonstrated that one of the key causes of malfunction in system development projects originates from a fault in the requirements engineering stage. On the website *“Forum on Risks to The Public in Computers and Related Systems,”* [4] several cases are demonstrating inadequate requirements analysis [3]. An average of 40% of software projects flop or do not encounter all the predictable requirements for the reason of a fault in requirements engineering. In 1976, Bell and Thayer have indicated that the nonconformity of system functions about user requirements, the incompleteness, inconsistency, and haziness of requirements documentation is generally reducing the quality of software [5]. Bell and Thayer conclude that “*the requirements for a system do not arise naturally; instead, they need to be engineered and have continuing review and revision.”*

Nowadays, the reputation of goal-oriented requirements engineering methods has improved significantly because of the inefficiency of the typical systems analysis methods when dealing with safety-critical systems [6]. In the requirements phase, these methods consider requirements as encompassing only methods and data and do not express the validation to the certain high-level context in the problem area. Maximum methods emphasize only on modeling and specification of the software system. As a result, they need help for analyzing the complex systems consisting of the system-to-be and its environs. Also, when system functionalities are automatic or dissimilar tasks of obligation are discovered to be denoted and assessed then typical modeling and analysis methods do not permit other system formations. Goal-Oriented Requirements Engineering (GORE) tries to resolve these significant issues [7]. GORE emphasizes the behaviors that lead to the preparation of system requirements. The major behaviors usually existing in GORE methods are goal extraction, goal enhancement and distinct forms of goal evaluation, and the allotment of goal fulfillment responsibility to actors. According to Axel van Lamsweerde [8] *“A goal is an objective that the system should achieve through the cooperation of agents in the software-to-be and the environment.”*

GORE visualize the given system and their environs as a set of active agents known as actors or stakeholders. To ensure the constraints they are assigned, each actor may restrict their behavior. In our railway interlocking system, these actors are LockManager, FrameAxioms, ATS, OnboardCBTC, etc. A goal is an aim that the system should accomplish via the assistance of actors in the system-to-be and the environs [9, 10]. A requirement is a goal. To fulfill this goal a system is

DOI: 10.37398/JSR.2022.660205

dependent on a single active agent, and also it is the responsibility of an agent. For example, *'ManagePointLocking'* is a goal, whose fulfillment is depending on the actor LockManager. An assumption is also a goal whose fulfillment is dependent on fulfilling another goal. That is to fulfill the assumption system deputes another single active agent in the environment. For example, to fulfill the goal *'MovePointPosition'* it depends on another goal task called *'EnableRoute'* of ATS actor. Unlike goals, expectations cannot be fulfilled by actors, they are fulfilled by organizational standard norms and rules. For example, dwell time and the train speed limit are depending on railway organizational rules and norms. Unlike goals, softgoals are non-functional requirements, whose fulfillment is not stated in a clear-cut manner. For example, safety is a non-functional requirement, to fulfill this requirement some tasks have a positive impact and others have negative impacts on it.

Making complex technological systems such as urban railway interlocking systems may require understanding, interpreting, and refining a large informal requirement into a concrete system. Nowadays urban railway interlocking system is completely dependent on wayside Intelligent Transport System (ITS) technologies like Automatic Train Controller (ATC), Global Positioning System (GPS), Transponder tags, Balise, Odometer, Wee-Z bond, signals, point switches, etc. Therefore, the urban railway interlocking system has to operate in an ultra-dependable range, so to model these dependencies and capture safety requirements, in this paper we used i*-based goal-oriented requirements language called Textual Syntax for GRL (TGRL) [6] because it integrates both Strategic Rationale (SR) and Strategic Dependency (SD) models [6]. To verify the safety requirements of the railway interlocking system they must be specified formally. Therefore, in our paper, safety requirements of railway interlocking system are specified formally by using Formal Tropos, a formal specification language for the i* model. The main objectives of our paper are (i) addressing variability issues to satisfy the reliability and safety of an interlocking system, (ii) integrating early and late requirements at the goal level. Early requirements are usually informal and deal with non-functional or system environment requirements. The late requirements are usually formal and emphases on verification, consistency, and completeness of requirements [11].

Semmak F., Laleau R., and et al. [12] within the context of the Cycab domain enhanced requirements engineering by using KAOS goal-oriented method. Cycab is a fully automated public vehicle. Also, they have extended the KAOS method to address the variability issues. This augmentation allows them to design a goal model for variable requirements. Also, they validated their goal model by using a software prototype developed on a partial version of the Cycab application domain.

Ponsard C., Massonet P., and et al. [13] for requirements extraction and supervision of mission-critical systems they have applied the KAOS goal-oriented requirements engineering method. The model they presented is an extract from the larger railways signaling specification. Their early work document was built on the active report of state machine diagrams with the safety requirements. For verification and validation related to requirements analysis, they used the FAUST toolbox.

Liaskos S., Lapouchnian A., and et al. [14] introduced a variability-intensive method to a goal decomposition that is adapted to help requirements identification for highly customizable software. Their recommend method is designed by using OR-decompositions of goals.

Gonzales-Baixauli B., Laguna M.A., and et al. [15] described how to deal with variability issues by using a goal model and aspect orientation. They promoted that aspect orientation is one of the best approaches to address complex issues.

Griss M., Favaro J., and et al. [16] described Reuse-Driven Software Engineering Business (RSEB) method. It is a use case-driven methodical reuse process architecture. In RSEB, the variability issues are addressed by structuring use case and object models using variation point with variants. They integrated the Feature-Oriented Domain Analysis (FODA) method into the processes and work products of the RSEB method.

Pohl K., Bockle G., and et al. [17] described the Software Product Line (SPL) method and the specification of a variability model to support the development and reuse of variable development artifacts. This method used both the concepts of FODA and RSEB. In SPL engineering, variability is a critical property of domain artifacts. Therefore, they used variability modeling to obtain the variability of domain requirements.

Dubois E., Yu E., and et al. [18] within the context of reactive systems described three well-defined and associated modeling activities at the requirements engineering stage. Also, they illustrated how, KAOS, AlbertII and Timed Automata formal languages support these three activities. To provide a high-level model and connecting different formal models they used the i* framework. They demonstrated their method by using a small process control system.

Fuxman A., Liu Lin, and et al. [19] designed i* goal model of a course-exam management case study to specify and analyze early requirements. He also used the NuSMV model checker to verify temporal constraints like LTL or CTL properties on goal models.

Ponsard, C., Ramon, V., and Deprez, J. [20] focused on a model-based method to present efficient support to the risk assessment stage. To get automotive assets and system properties to assess the effect of damage scenarios, detect threats, and evaluate their feasibility, they proposed a goal-oriented meta-model of car light control sub-system.

At the goal-oriented requirements stage, the methods like FODA, RSEB or SPL are not properly addressing the variability issues. To differentiate with the previous works, we can say that our method differs from them and make use of i*-based goal-oriented method called TGRL [6] to integrate both early and late requirements at the goal level, and variability issues are addressed by using strategic dependency model. To demonstrate how our method is useful for a safety-critical system we used an urban railway interlocking system as a case study. Also, we formalized requirements specifications by using Formal Tropos.

Section 2 describes the basic concepts used in the urban railway interlocking design process. Section 3 describes the goal-oriented modeling of an urban railway interlocking system. Section 4 describes requirements specification and we conclude the paper in section 5.

2 Urban Railway Interlocking Systems

Rail Rapid Transit is a long-lasting and well-known national urban transportation system. It transports a huge number of commuters at high speed. Rail Rapid Transit is a kind of transportation that runs separately on fixed guideways in an urban area at high speed. With technological improvement, today Rail Rapid Transit system is completely dependent on ITS technologies. Therefore, nowadays Rail Rapid Transit is also called the Smart Mass Transit System [21].

Nowadays, the Communications-Based Train Control (CBTC) signaling system is adopted in Urban Railway Interlocking System. It broadcast the signals among the trains and trackside objects for traffic management and train control. It also finds more accurately the precise location of the train. Which results in an efficient and safe way to manage the railway traffic flow and reduce the headway between trains [22].

The CBTC based urban railway control systems are in operation in Copenhagen, Paris, Singapore, Dubai, Barcelona, and Delhi. These are also good examples of driver-less commuter metros. Computerized systems optimize the passing time of the metro and rise the regular speed of the system, permitting the metros to operate more rapidly, decreasing the time it takes a metro to reduce the speed at stations, and rising reliability. As of 2018, Singapore's Smart Mass Transit is the long-sighted driverless metro in the globe, traversing 199 km [24].

2.1 Moving Block Interlocking System

Moving Block is one of the interlocking systems normally interconnected with the ATC system. In this interlocking system, a train moves with a well-defined encircled block. In this system, a train location and speed are transmitted to wayside CBTC, which then calculates the proper safety distance to be maintained between trains on the same railway track, and consequently manipulates the signal positions and speed controls for the trains on the railway track. The safe distance among trains is the space required by a train to a full stop. The fine control over the speeds and locations of the trains permits a tight headway and full utilization of the track [23]. The working procedure of moving block interlocking system is shown in Fig. 1.

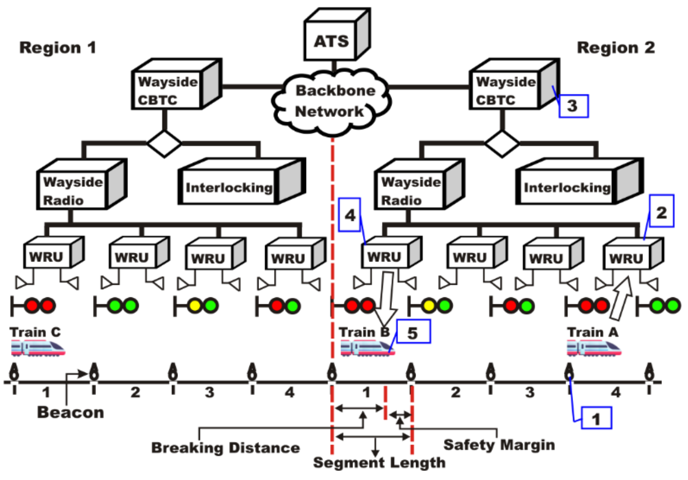

Fig. 1. Moving block interlocking in CBTC system.

The CBTC is based on the moving-block concept as shown in Fig. 1. The working principle of the CBTC system along with the moving block is listed below.

1. Train A computes its location with an onboard balise reader or odometer.
2. Train A sends its location to Wayside CBTC via radio (wireless radio unit).
3. The Wayside CBTC computes the safety distance for Train B according to the location of Train A.
4. The Wayside radio transmits the limit authority given by Wayside CBTC to train B.
5. Train B adjusts its speed according to the limit authority to obey safety distance.

2.2 Interlocking Components

In the domain of railway signaling, a railway interlocking is an automatic control system that controls the railway trackside objects to allow a riskless function of the train traffic flow. An interlocking state is usually well-defined by a sequence of specific states of its objects. Fig. 2 Illustrate a simple track layout of an urban railway interlocking system case study. The physical and logical objects of an interlocking system are as follows:

- **Tracks**: The pathway along which a train proceeds.
- **Points**: The railway forks allowing a train to move into a new track.
- **Signals**: Permits trains to enter the track. If red light glows, then a signal is on and it denies the train to enter into the track, else if the green light glows, then a signal is off and allows the train to enter into the track.
- **Track Segments**: Railway tracks are split into sectors called track segments with the help of a delimiter, and each segment is associated with a track circuit.
- **Track Circuit**: Continuously detect the presence of a train in each block segment. In the DTG (distance to go technology) version of the CBTC signaling system, each track segment has a relay on each termination called a "*Wee-Z Bond*". It acts as a transmitter for one segment and a receiver for the nearby segment. If a segment has an unbroken circuit, then the segment is considered unoccupied, and so safe for the train to pass into the segment. If a circuit is broken, then the segment is considered as occupied and preventing a train to pass into the segment. The wayside ATC receives broken or unbroken circuit information from each segment to determine which segments are occupied and based on this information it computes and sends safety distance to the trains [23, 25]. In the CBTC signaling system, it is used as a secondary device to determine train location.
- **Balise**: It is an electronic beacon similar to the transponder tag, placed between rails to determine train location in case of CBTC with a moving block system.
- **Routes**: Consist of sequentially joined track segments that begin at source signal track, and finish at destination signal track. (The routes can be set or unset).
- **Sub-routes**: Each track segment along the route is usually called a sub-route. The sub-route can be locked or free.

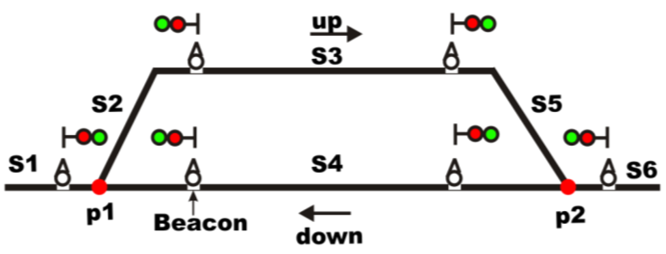

Fig. 2. Simple track layout of an urban railway interlocking system case study.

2.3 Goals of Interlocking

If the point shifts its position when a train crosses a point in the opposite direction, then there is a possibility that a train can derail with the train rooting two routes. Therefore, it is essential to lock the point once its position is set before allowing a train to move on it. To prevent collisions, it is essential to check the status of each segment in the track layout and the safe distance between trains. These tasks are fulfilled by *"Interlocking"* the signals with the points.

3 Goal-Oriented Modeling of an Urban Railway Interlocking System Case Study

i* is one of the Goal-Oriented Requirements Engineering methods. Its main aim is satisficing a goal via the assistance of actors in the system and the environs. Therefore, it is also known as an agent-oriented modeling framework. i* supports modeling both early and late requirements.

In the early requirements stage, the i* is used to model the environs of the system. It assists the analyst to analyze the domain by graphical representation of the stakeholders of the system, the relationship between the stakeholders and their goals. The i* model developed at the early stage helps us in understanding why an innovative system is required and how it works. In the late requirements stage, the i* is used to model both the functional and non-functional requirements of the stakeholders [11, 26]. To model both early and late requirements i* provides two main building blocks called Strategic Rationale (SR) model and Strategic Dependency (SD) model.

The SR model is used to discover the rationale behind the activities inside the system and its environs. The SR model assists the analyst to understand each stakeholder's requirements, and how these requirements are fulfilled. The SD model is used for analyzing dependency-based vulnerabilities in the system. It denotes the intentionality of the goals in the organization and different techniques of achieving a specific goal. For example, one way to fulfill the goal is through an intentional goal dependency, that is handing over the goal to another actor to achieve it, or handing over the goal to yet another actor. The goal refinement by using alternative techniques listed above not only address variability issues but also reduces combinational explosion problems.

TGRL is i*- based Goal-oriented Requirements Language (GRL) with textual syntax, which provides a tactic to textualize scenarios [27]. Unlike i*, In TGRL both SR and SD are intertwined. It supports agent-oriented modeling, goal analysis and guides the architectural design process [6]. It's a part of the User Requirements Notation (URN) standard [28]. In TGRL grammar was specified with Xtext [29] often used for the development of textual domain-specific languages.

In TGRL, modeling, analysis, and transformations are currently supported by jUCMNav [30], so a model can also be visualized and analyzed using jUCMNav. jUCMNav is an Eclipse plug-in for modeling, analysis, and visualization of URN models. It supports i* concepts and allows the user to analyze i* constraints. It is pronounced as juicy – em – nav (juice up your modeling).

Our proposed approach is divided into the following stages to extract all of the safety requirements in the requirements engineering stage itself and construct a goal model and formalize the safety requirements.

1. Generate a goal model called the i*-SR model to elicit early requirements.
2. Generate a goal model called the i*-SD model to fulfill our objective of addressing variability issues to satisfy the reliability and safety of an interlocking system.
3. Generate a goal model called i*-SR/SD model by merging both SR and SD models using TGRL.
4. Generate a graphical model of i*-SR/SD model by using jUCMNav for visualization and analysis purposes.
5. Generate Requirements Specification (RS) model to formalize the extracted early requirements in first-order linear-time temporal logic by using Formal Tropos. This stage fulfills our objective of integrating early and late requirements at the goal level.

3.1 i*-SR/SD Model

i*-SR/SD model is a goal-oriented model exercised to address the issues variability. Here we initiated our model by collecting functional and non-functional specifications of a track layout and environment of an urban rail at the early stage. We used TGRL to construct our proposed model. The actors reside in our model and their goals are listed below.

1. **LockManager actor** has a goal of *'ManagePointLocking'* and *'ManageSubRouteLocking.'* To achieve the *'ManagePointLocking'* goal, it makes use of two tasks called *'LockThePoint'* and *'UnlockThePoint.'* Points must be locked when a route is instructed and pass by a train. Before locking the points, they must be set to the proper position. Therefore, it depends on a goal *'MovePointPosition'*, of an actor FrameAxioms. Similarly, before unlocking the points the sub-routes must be unlocked. To achieve the *'ManageSubRouteLocking'* goal for up and down trains, it makes use of two tasks called *'LockSubRoute'* and *'UnlockSubRoute.'* To lock the sub-routes track segments must be free and points must be locked. To unlock the sub-routes, it depends on a task *'DisableRoute'* of an actor ATS.
2. **FrameAxioms actor** has a goal of *'MovePointPosition'* and *'UpdateSegmentStatus.'* To achieve the *'MovePointPosition'* goal, it makes use of two tasks called *'MovePointLeft'* and *'MovePointRight.'* Points are not moving to any position until they are instructed to do so. To move points a route must be enabled and points must be

unlocked. Therefore, it depends on tasks *'EnableRoute'* of the ATS actor and *'UnlockThePoint'* of the LockManager actor. To achieve the *'UpdateSegmentStatus'* goal, it makes use of two tasks called *'SetSegmentStatus'* and *'UnsetSegmentStatus.'* To update the status of a segment, that is whether a train is occupied a track segment or track segment is free, it depends on a resource *'Beacon.'*

3. **ATS actor** has a goal of *'ControlRoutes'* for controlling the routes of active trains. To achieve the *'ControlRoutes'* goal, it makes use of two tasks called *'EnableRoute'* and *'DisableRoute.'* To disable the route a preceding sub route of the leading train must be free. Therefore, it depends on a resource *'Beacon.'*
4. **SignalMan actor** has a goal of *'ExtinguishingSignal'* for switching the signals to provide entrance to the trains. In CBTC, instead of glowing the signal lights, the signals are extinguished. To achieve the *'ExtinguishingSignal'* goal, it makes use of two tasks called *'ExtinguishingAMarkerOff'* and *'ExtinguishingAMarkerOn.'* To extinguish the 'A' marker off, all sub-routes and points must be locked. Therefore, it depends on the task *'LockSubRoute'* of an actor LockManager. Similarly, to extinguish the 'A' marker on, all sub-routes and points must be unlocked. Therefore, it depends on the tasks *'UnlockSubRoute'* and *'UnlockThePoint'* of an actor LockManager. A goal *'ExtinguishingSignal'* also depends on a goal *'MaintainLMA'* of an actor OnboardCBTC, to obey the Limit Movement Authority (LMA) sent by wayside CBTC.
5. **Train actor** has a goal of *'MoveAndStopTrain'* to move and stop up and down trains over the adjacent track segments. To achieve the *'MoveAndStopTrain'* goal, it makes use of three tasks called *'MoveTrain,'* '*StopTrain,'* and *'AccelerateOrDeaccelerateSpeed.'* To move a train, it depends on the task *'ExtinguishingAMarkerOff'* of an actor SignalMan. To stop a train, it depends on the task *'ExtinguishingAMarkerOn'* of an actor SignalMan. To accelerate or deaccelerate speed, it depends on a task *'ExecuteLMA'* of an actor OnboardCBTC.
6. **OnboardCBTC** actor has a goal of *'MaintainLMA'* to maintain the LMA sent by the wayside CBTC. To achieve the *'MaintainLMA'* goal, it makes use of two tasks called *'ReceiveLMA'* and *'ExecuteLMA.'*
7. **WaysideCBTC** actor has a goal of *'MaintainWorstCaseStoppingDistance'* to maintain a safe distance between two trains to reduce the headway between them. To achieve the *'MaintainWorstCaseStoppingDistance'* goal, it makes use of two tasks called *'ComputeLMA'* and *'SendLMA.'* To compute LMA, it needs Train T1 and Train T2 current positions and speed, so it depends on resources *'Beacon'* and *'Tachometer.'* It also has a softgoal *'SafeTransport'* that has a positive impact on a goal *'MaintainWorstCaseStoppingDistance'* and negative impact on a goal *'ControlRoutes.'*

3.2 Graphical Model of i*-SR/SD model

For visualization and analysis purposes we converted our textual i*-SR/SD model into a graphical model using a jUCMNav converter [30] and it is shown in Fig. 3. As we can see in Fig. 3, we elicited early requirements of an urban railway interlocking system by using the SR model. In Fig. 3, we visualized the SR model of an urban railway interlocking system and its environs as a set of active agents known as actors. Each actor's requirements are described in terms of SR activity elements called goals, soft goals, tasks, resources, and relationships between them. These activity elements are related to each other by using decomposition links (AND-decomposition) or means-ends links (OR-decomposition) [31].

In Fig. 3, goals are shown in rounded rectangle shape, tasks are shown in a hexagon shape, and soft goals are shown in curved rectangle shape. For example, an actor WaysideCBTC has a goal of *'MaintainWorstCaseStoppingDistance'* to maintain a safe distance between two trains to reduce the headway between them. To achieve this goal, it makes use of two tasks called *'ComputeLMA'* and *'SendLMA.'* These tasks are related to the goal by using AND-decomposition. A task is a particular activity that the actor wants to perform to achieve the goal. A soft goal is a non-functional requirement. For example, actor WaysideCBTC has a soft goal of *'Safe Transportation.'* To fulfill this soft goal, the goal *'MaintainWorstCaseStoppingDistance'* of WaysideCBTC actor have a positive impact and the goal *'ControlRoutes'* of ATS actor have negative impacts on it. A resource is denoted by rectangle shape and it is a physical or conceptual entity. For example, actor OnboardCBTC has a physical entity called '*Tachometer'* to calculate the distance it travels.

In Fig. 3, the variability issues are addressed by using the SD model. Variability is an alternative way of goal satisfaction. The dependencies among the actors are said to be intentional if fulfillment of one actor's goal depends on another actor's intentional elements called goals, soft goals, tasks, or resources. The SD model denotes the intentionality of the goals in the organization and different techniques of achieving a specific goal.

The first one is to allow the actor to achieve the goal that they own. For example, allowing the actor ATS to fulfill the goal *'ControlRoutes,'* by fulfilling the task *'EnableRoute'* or fulfilling the task '*DisableRoute.'* The second one is through an intentional goal dependency, that is handing over the goal to another actor to achieve it. For example, the *'ManagePointLocking'* goal of the LockManager actor depends on the task '*UnsetSegmentStatus'* of goal *'UpdateSegmentStatus'* of an actor FrameAxioms. The third one is handing over the goal to yet another actor. For example, the task *'MoveTrain'* of goal *'MoveAndStopTrain'* of an actor Train depends on a task *'ExtinguishingAMarkerOff'* of goal *'ExtinguishingSignal'* of an actor SignalMan depends on a task *'LockSubRoute'* of goal *'ManageSubRouteLocking'* of an actor LockManager.

Now, let us check another example for the variability issue. In the case of the moving block CBTC system, the train stops only at a conflicting point. The LMA for a train always finishes at a conflicting point in front of the train. A spot along the track beyond which a train is not allowed to move is known as a conflicting point. It is either static in case of switch points or buffer stop at the station, or it is dynamic in case of the rear of the train in front. As shown in Fig. 3, the task '*StopTrain'* of goal *'MoveAndStopTrain'* of an actor Train depends on a task *'ExtinguishingAMarkerOn'* of a goal

'ExtinguishingSignal' of an actor SignalMan, it, in turn, depends on a goal *'MaintainLMA'* of an actor OnboardCBTC in case of dynamic conflicting point. In case of static conflicting points an actor SignalMan delegates goal satisficing to the actor LockManager.

If variability issues are not handled properly then the depender (an actor who depends on another actor called dependee to fulfill his goal) is said to be vulnerable because the dependee has not fulfilled the goal. In Fig. 3, LockManager (depender) who depend on another actor FrameAxioms (dependee) to fulfill its goal *'ManagePointLocking.'* While blending these early requirements with late requirements this SD model helps us to analyze these variability issues. The specification in Fig. 4 illustrates handling variability as well as blending early requirements with late requirements in Formal Tropos.

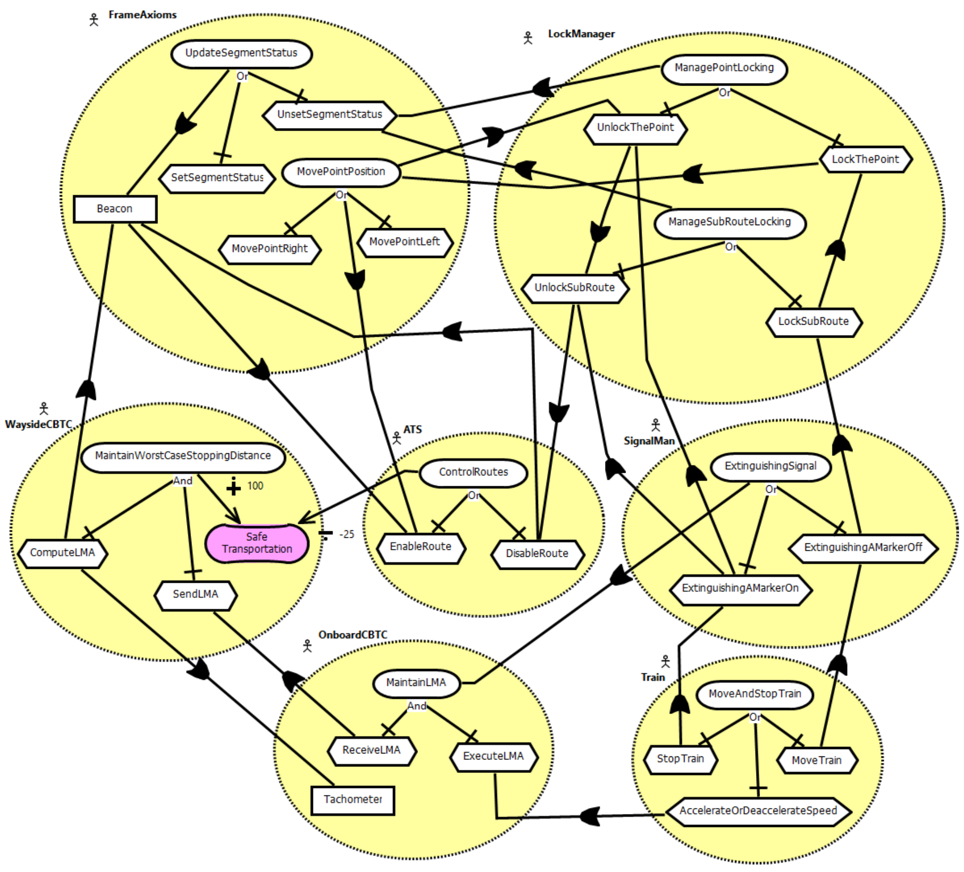

Fig. 3. A graphical model of i*-SR/SD goal model, validated using jUCMNav.

4 Requirements Specification

Formal Tropos is a requirements specification language for the i* model [32]. It implements primitive concepts for translating early requirements into late requirements, along with temporal specification. All the objects in the domain and their relationship are specified by using two tiers in Formal Tropos. The structure of each object along with its properties is specified by using the outer tier, it just looks like a class statement. A first-order linear-time temporal logic is used inside the inner tier to a constraint on the objects.

4.1 The Outer Tier

The outer tier of the Formal Tropos specification of an urban railway interlocking system case study is given below. The actors, entities, resources, and intentional elements of the i*-SR/SD model are mapped with class statements. Every single class holds one or more associated primitive or class type attributes. For example, the attribute lock of goal *'ManagePointLocking'* is a primitive attribute that has boolean properties, to decide whether the point is locked or not. An attribute mlp of task LockThePoint is a class type attribute that has constant properties, which determine references to a goal *'ManagePointLocking'* instance. The two actors active in a delegation relationship are represented with attributes of dependency, as shown in Fig.4.

The modality of intentional-elements satisfaction is described by a keyword mode. For example, achieve&maintain is the modality of goal *'ManagePointLocking'*. It indicates that points are locked properly and the state of the point is maintained continuously.

```
Specification: = (actor | entity |
intentional-element | global-properties |
dependency)
Actor LockManager
Goal Dependency ManagePointLocking Mode
achieve&maintain
```

```
     Depender LockManager Dependee
FrameAxioms
     Attribute optional latch: Boolean
Task Dependency LockThePoint Mode
achieve&maintain
     Depender LockManager Dependee
FrameAxioms
     Attribute optional lock: boolean
               constant mlp:
ManagePointLocking
               Constant mpp:
MovePointPosition
Task Dependency UnlockThePoint Mode
achieve&maintain
     Depender LockManager Dependee
LockManager
     Attribute optional unLock: boolean
               constant mlp:
ManagePointLocking
Goal   Dependency   ManageSubRouteLocking   Mode
achieve&maintain
     Depender LockManager Dependee FrameAxioms
     Attribute optional subRoute: Boolean
Task Dependency LockSubRoute Mode
achieve&maintain
     Depender LockManager Dependee
LockManager
     Attribute optional lock: boolean
               constant msr:
ManageSubRouteLocking
Task Dependency UnlockSubRoute Mode
achieve&maintain
     Depender LockManager Dependee ATS
     Attribute optional unLock: boolean
               constant msr:
ManageSubRouteLocking
               constant dr: DisableRoute
```

Fig. 4. Outer tier specification for the actor LockManager.

4.2 The Inner Tier

The inner tier of the Formal Tropos specification of an urban railway interlocking system case study is given below. It involves the constraints that express the dynamical properties of actors, entities, resources, and intentional elements. *Invariant* constraints describe specifications of a class that should be held during the period of all instances of a class. Usually, it describes relations between the feasible values of attributes. For example, task dependency *'LockThePoint'* invariant constraint describes a relation between the feasible values of attributes of *'ManagePointLocking'* instances. Also, invariant constraints describe cardinality on a specified class object. For example, the goal *'ManageSubRouteLocking'* invariant constraint declares that at most one route can be locked at a time. *Creation* constraints describe specifications of a class that should be held during the born of a new instance of a class. For example, the goal *'MovePointPosition'* creation constraint declares that all point instances should be in a normal position when they are born. *Fulfillment* constraints describe specifications of a class that should be held when the goal of all the instances of a class is satisfied. For example, the goal *'ManagePointLocking'* fulfillment constraint declares that, if a point is locked then it will not change its position or it must be unlocked to change its position.

The above of all constraints have property and event categories. *Property* categories may be constraint, assertions, or possibilities. Constraint states that specifications are implicitly prescribed. While assertions and possibilities are explicitly prescribed. Assertion states that specification should be held in all possible situations. Similarly, the possibility states that specification should be held in at least one situation. *Event* categories may be a condition, trigger, or definition. The condition states that specifications are essential. The definition states that specifications are essential and suitable. Trigger states that specifications are suitable for creation and fulfillment constraints. Formal Tropos specifications are described with first-order linear-time temporal logic formulas as shown in Table 1.

Table 1. First-order linear-time temporal logic formulas.

Temporal logic	Description
Xf	It states that the formula must be true in
Ff	It states that the formula is true now or finally it happens to be true in a certain
Gf	It states that the formula is true now and
$f1\,U\,f2$	It states that formula f1 is true until formula f2 is true in future states.
Yf	It states that the formula must be true in
Hf	It states that the formula is true now and
Pf	It states that the formula is true in some
$f1\,S\,f2$	It states that formula f1 is true since formula f2 is true in earlier states.
$JustFulfilled(t)$	It states that predicate is true if fulfillment constraint is true in the
$JustCreated(t)$	It states that predicate is true if the creation constraint is true in the present
$Fulfilled(t)$	It states that predicate is true if fulfillment constraint is satisfied for the

4.3 RS Model

Requirements Specification (RS) model is designed to formalize i*-SR/SD model by using a Formal Tropos, a formal specification language for i* model. Here, we formalize safety requirements by focusing on each intentional element, its environment, and its fulfillment specifications. The formal Tropos specification of the i*-SR/SD model is given in Fig. 5.

Requirement Specification 1: A goal of managing the lock point is fulfilled only if points are locked or unlocked.

```
Goal ManagePointLocking
Fulfillment constraint condition
Fulfilled (self) ⇒ (∀ lp: LockThePoint
Fulfilled (lp)) ∨ (∀up: UnLockThePoint
Fulfilled (up))
```

Requirement Specification 2: To lock the point it should be first set in a proper position.

```
Invariant constraint condition
(∀ mlp: ManagePointLocking (mlp.latch =
lock)) ⇒ (∀ mpp: MovePointPosition (Fulfilled
(mpp))
```

Requirement Specification 3: To unlock the point it should first unlock the sub-route.

```
Invariant constraint condition
(∀ mlp: ManagePointLocking (mlp.latch =
unLock)) ⇒ (∃ usr: UnlockSubRoute
(usr.depender = self)⇒ (Fulfilled (usr))
```

Requirement Specification 4: All point instances should be in normal position when they are born.

```
Creation constraint condition
∀mpp: MovePointPosition (mpp.pointPos = left)
∧ (mpp.actor = actor)
```

Requirement Specification 5: Point should move only if it is unlocked and route is enabled.

```
Fulfillment asseration condition
(¬pointPos   ∧   F   pointPos)   ⇒   (F   ∃
ulp:UnlockThePoint (ulp.mlp.latch = unLock)∧(F
∃ er:EnableRoute (er.cr.route = enable))
```

Requirement Specification 6: A point is positioned to the left only if it is commanded to do so.

```
Invariant constraint condition
(mpp.pointPos   =   left)   ⇒   (Fulfilled
(mpp.dependee))
```

Requirement Specification 7: A point is positioned to the right only if it is commanded to do so.

```
Invariant constraint condition
(mpp.pointPos   =   right)   ⇒   (Fulfilled
(mpp.dependee))
```

Requirement Specification 8: All sub-routes of a route must be locked if a route is enabled.

```
Invariant constraint condition
(cr.route = enable) ⇒ (∀ lsr: LockSubRoute
Fulfilled (lsr))
```

Requirement Specification 9: All track segment instances should be in clear position when they are born.

```
Creation constraint condition
∀uts: UpdateSegmentStatus (uts.segmentStatus =
clear) ∧ (uts.actor = actor)
```

Requirement Specification 10: To lock the sub-route, track segment and points are commanded to the proper position.

```
Invariant constraint condition
(msr.subRoute = lock) ⇒ (∀ uss:
UpdateSegmentStatus Fulfilled (uss)∧∀mpp:
MovePointPosition Fulfilled (mpp))
```

Requirement Specification 11: A goal of controlling routes is fulfilled only if routes are enabled or disabled.

```
Fulfillment constraint condition
Fulfilled (self) ⇒ (∀ er: EnableRoute
Fulfilled (er)) ∨ (∀ dr: DisableRoute
Fulfilled (dr))
```

Requirement Specification 12: To glow green signal (Extinguishing A marker off) all sub-routes must be locked.

```
Invariant constraint condition
∀es: ExtinguishingSignal (es.Amarker = off)⇒
((∀ ls: LockSubRoute Fulfilled (er)) ∧ (∀ lp:
LockThePoint   (lp.depender   =   ls.actor)   ⇒
Fulfilled (lp)))
```

Requirement Specification 13: A point not to change its position once the train occupies a track segment.

```
Global assertion
∀lp: LockThePoint (Fulfilled (lp) ⇒ ∀ mpp:
MovePointPosition ((mpp.actor = lp.dependee)
∧ Fulfilled (mpp)) ⇒ ∀ cp:
UpdateSegmentStatus (Fulfilled (cp)) ⇒ ∃ wz:
Beacon ((wz.actor = cp.depende) ∧
(wz.segmentStatus = cp.occupied))
```

Requirement Specification 14: To avoid collision each train has to maintain safe distance between them.

```
Global assertion
∀ t1, t2: MoveTrain G ((t1.mst.run = t1.move)
∧      (t2.mst.run      =      t2.move)      ⇒
(t1.mst.trainPosition ≠ t2.mst.trainPosition)
∧ (t1.mst.trainSpeed ≠ t2.mst.trainSpeed) ⇒
Fulfilled (t1) ∧ Fulfilled (t2)
```

Requirement Specification 15: To avoid derailment a point should be set in a proper position before allowing the train to continue its path.

```
Global assertion
∀ t: MoveTrain G ((t.mst.run = t.move) ⇒ (∃
gl: ExtinguishingAMarkerOff Fulfilled (gl)) ∧
(∃ e: EnableRoute Fulfilled (e)) ⇒ (∃ pp:
MovePointPosition     (¬pp.pointPos     ∧     F
pp.pointPos))   ⇒   (F   ∃   fa:   FrameAxioms
(fa.cmdpointPos   =   pp.pointPos))  ⇒  (∃  lp:
LockThePoint(lp.dependee    =    pp.actor    ∧
lp.mlp.latch = lock)))
```

Fig. 5. Formal Tropos specifications of an urban railway interlocking system.

5 Conclusion

In this paper, we described goal-oriented requirements engineering in the context of urban railway interlocking system, which has to do with the usage of goals for extracting, enlarging, organizing, identifying, analyzing, collaborating, documenting, and revising requirements as i*-SR/SD model. Our i*-SR/SD model is a textual model constructed using TGRL. For visualization and analysis purposes we also translated our textual model to a graphical model by using a jUCMNav converter. In the i*-SR/SD model by using alternative ways of goal satisfaction we addressed the variability issues like stopping the train at the conflicting point in case of moving block CBTC system. The conflicting point is static or dynamic, so we need to address this kind of variability issue carefully. Also, we addressed the vulnerability issues by using strategic dependencies. These strategic dependencies helped us to integrate early requirements with late requirements. Finally, we have chosen Formal Tropos to blend early requirements with late requirements and formalize our i*-SR/SD model, because Formal Tropos is a specification language for the i* model and it allows us to specify safety requirements in first-order linear-time temporal logic formulas. In future enhancement, we model check these early requirements specifications.

References

[1] Van Lamsweerde, A. Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley, 2nd Ed. (2011).

[2] Hall, T., Beechham, S., and Rainer, A. Requirements Problems in Twelve Companies – An Empirical Analysis. In Proceedings of the 6th International Conference on Empirical Assessment and Evaluation in Software Engineering (EASE 2002), (2002).

[3] Finkelstein, A., and Dowell, J. A comedy of errors – the London Ambulance Service Case Study. In Proceedings of the 8th International Workshop on Software specifications & Design, Los Alamitos, pp. 2-4, IEEE Computer Society Press, (1996).

[4] Ross, D.T., and Schoman Jr, K.E. Structured analysis for requirements definition. In IEEE Transactions on Software Engineering, pp. 6-15, (1977).

[5] Thayer, T.A., and Bell, T.E. Software requirements: Are they really a problem? In Proceedings of 2nd International Conference on Software Engineering, pp. 61–68, San Francisco, (1976).

[6] Abdelzad, V., Amyot, D., Alwidian, S. A., and Lethbridge T. C. A textual syntax with tool support for the goal-oriented requirement language. In Proceedings of the Eighth International i* Workshop (istar 2015), CEUR Vol-978, (2015).

[7] Ponsard, C., Massonet, P., Molderez, J.F., and et al. Early verification and validation of mission-critical systems. Formal Methods in System Design 30, 233, (2007). https://doi.org/10.1007/s10703-006-0028-8

[8] van Lamsweerde, A. Requirements engineering in the year 00: a research perspective. Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium, pp. 5-19, (2000). https://doi.org/10.1145/337180.337184

[9] Amyot, D. Goal modeling education with GRL: An Experience report. In Proceedings of the 8th international i* Workshop in Conjunction With the 23rd International Requirements Engineering Conference, Ottawa, Canada, pp. 1-6, (2015).

[10] Van Lamsweerde, A. Goal-oriented requirements engineering: A guided tour. In Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pp. 249-262. Toronto, Canada. (2001). https://doi.org/10.1109/ISRE.2001.948567

[11] Yu, E. S. K. Towards modelling and reasoning support for early-phase requirements engineering. In Proceedings of IEEE International Symposium on Requirements Engineering - RE97, pp. 226-235, (1997).

[12] Semmak F., Gnaho C., and Laleau R. Extended KAOS method to model variability in requirements. In: Maciaszek L.A., González-Pérez C., Jablonski S. (eds) Evaluation of Novel Approaches to Software Engineering. ENASE 2009. Communications in Computer and Information Science, vol. 69. Springer, Berlin, Heidelberg, (2010). https://doi.org/10.1007/978-3-642-14819-4_14

[13] Ponsard, C., Massonet, P., Molderez, J.F., and et al. Early verification and validation of mission-critical systems. Formal Methods in System Design 30, 233, (2007). https://doi.org/10.1007/s10703-006-0028-8

[14] Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., and Mylopoulos, J. On goal-based variability acquisition and analysis. 14th IEEE International Requirements Engineering Conference (RE'06), pp. 79-88, (2006). https://doi.org/10.1109/RE.2006.45

[15] Gonzales-Baixauli, B., Laguna, M.A., and Sampaio do Prado Leite, J.C. Using Goal-models to analyse variability. First International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2007, Limerick, Ireland, (2007).

[16] Griss, M. L., Favaro, J., and d'Alessandro, M. Integrating feature modeling with the RSEB. In Proceedings of Fifth International Conference on Software Reuse (Cat. No.98TB100203), pp. 76-85, (1998). https://doi.org/10.1109/ICSR.1998.685732

[17] Pohl, K., Bockle, G., and van der Linden, F. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Berlin, Heidelberg, (2005). https://doi.org/10.1007/3-540-28901-1

[18] Dubois, E., Yu E., and Petit, M. From early to late formal requirements: a process-control case study. In Proceedings of Ninth International Workshop on Software Specification and Design, pp. 34-42, (1998). https://doi.org/10.1109/IWSSD.1998.667917

[19] Fuxman, A., Liu, L., Pistore, M., and Mylopoulos, J. Specifying and analyzing early requirements: Some experimental results. In Proceedings of the 11th IEEE International Requirements Engineering Conference, Monterey Bay, CA, USA, pp. 105-114, (2003). https://doi.org/10.1109/ICRE.2003.1232742

[20] Ponsard, C., Ramon, V. and Deprez, J. Goal and Threat Modelling for Driving Automotive Cybersecurity Risk Analysis Conforming to ISO/SAE 21434. In Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021), pp. 833-838, (2021). https://doi.org/10.5220/0010603008330838

[21] Global Mass Transit, Information & analysis on the global mass transit industry. https://www.globalmasstransit.net/index.php Accessed 2021/07/10.

[22] The Railway Technical Website. A window on the world of railway systems, technologies, and operations. http://www.railway-technical.com/ Accessed 2021/07/10.

[23] Rakesh, L., and Kadakolmath, L. Modeling and formal verification of SMT rail interlocking system using PyNuSMV. 2018 4th International Conference on Recent Advances in Information Technology (RAIT), vol. 2, pp. 667-674, (2018).

[24] https://doi.org/10.1109/RAIT.2018.8388983

[25] UITP. Statistics Brief - World Metro Figures 2018. https://cms.uitp.org/wp/wp-content/uploads/2020/06/Statistics-Brief-World-metro-figures-2018V3_WEB.pdf,

[26] Accessed 2021/07/10.

[27] The Railway Technical Website. A window on the world of railway systems, technologies, and operations. http://www.railway-technical.com/ Accessed 2021/07/10.

[28] Babar, Z., Nalchigar, S., Lessard, L., and et al. Instructional experiences with modeling and analysis using the i* framework. In Proceedings of the 1st International istar Teaching Workshop Co-Located with the 27th International Conference on Advanced Information Systems Engineering, Stockholm, Sweden. pp. 31-36, (2015).

[29] Vahdat-ab. Textual modeling language for GRL. https://github.com/vahdat-ab/TGRL Accessed 2021/07/15.

[30] ITU-T. Z.151: User requirements notation (URN) - Language definition.

[31] https://www.itu.int/rec/T-REC-Z.151-201210-I/en Accessed 2021/07/15.

[32] Xtext. Language engineering for everyone. http://www.eclipse.org/Xtext/ Accessed 2021/07/15.

[33] Roy JF., Kealey J., and Amyot D. Towards integrated tool support for the user requirements notation. In: Gotzhein R., Reed R. (eds) System Analysis and Modeling: Language Profiles. SAM 2006. Lecture Notes in Computer Science, vol 4320. Springer, Berlin, Heidelberg, (2006). https://doi.org/10.1007/11951148_13

[34] Yu E., and Liu L. Modelling trust for system design using the i* strategic actors framework. In: Falcone R., Singh M., Tan YH. (eds) Trust in Cyber-societies. Lecture Notes in Computer Science, vol. 2246. Springer, Berlin, Heidelberg, (2001). https://doi.org/10.1007/3-540-45547-7_11

[35] Fuxman, A., Liu, L., Mylopoulos, J. et al. Specifying and analyzing early requirements in Tropos. Requirements Engineering 9, 132–150 (2004). https://doi.org/10.1007/s00766-004-0191-7

***