PART 2: BUILDING SYSTEM MODELS FOR REQUIREMENTS ENGINEERING

Part 1 of this book was dedicated to the fundamentals of requirements engineering. Once the motivations, conceptual background, and terminology used in this discipline were introduced, we studied a comprehensive sample of state-of-the-art techniques for requirements elicitation, evaluation, specification, consolidation, and evolution.

As we noted it, those techniques are faced with a recurring problem of *focus* and *structure*. The elicitation techniques in Chapter 2 raised the problem of focussing and structuring elicitation sessions (such as background studies, interviews, observations, or group sessions), and elicitation artefacts (such as scenarios, prototypes, or reusable knowledge fragments). The evaluation techniques in Chapter 3 raised the problem of identifying and comparing items at some common level of abstraction and granularity for risk analysis, conflict management, option selection, or prioritization. The specification techniques in Chapter 4 offered mechanisms for structuring specifications but did not tell us much on how we should structure a complex specification through those mechanisms. For quality assurance, the inspection techniques in Chapter 5 are more effective if inspections can be focussed on structured specifications. The techniques for requirements validation and verification require the availability of structured specifications as well. Likewise, the evolution techniques in Chapter 6 are more effective when a rich structure is available for defining change units, granularities of traceable items, built-in derivation links, and satisfaction arguments to be replayed in case of change.

A model-based approach to RE addresses this recurring problem of focus and structure. A *model* is an abstract representation of the system, as-is or to-be, where key features are highlighted, specified, and inter-related to each other.

The role of models in requirements engineering

A "good" system model may act as a driving force for the RE process.

- It provides a comprehensive structure for what needs to be elicited, evaluated, specified, consolidated, and modified.
- It allows us to abstract from multiple details to focus elicitation, evaluation, specification, and evolution on key system aspects.
- It defines a common interface between those various RE activities, each acting as a producer or consumer of portions of it.
- It provides a basis for early detection and fix of errors in requirements, assumptions, and domain properties.
- It facilitates the understanding of complex systems and their explanation to stakeholders.
- It provides a basis for making decisions among multiple options and for documenting such decisions.
- It defines the core RE artefact from which the requirements document can be generated.

Requirements on RE models

To play such significant role in the RE process a good model should meet the following requirements.

- *Adequate:* The model should adequately represent the essence of the target system while abstracting from unnecessary details.
- *Complete and pertinent:* The model should capture all pertinent facets of this system along the WHY-, WHAT-, and WHO-dimensions introduced in Section 1.1.3.
- *Precise and analyzable:* The model should be accurate enough to support useful forms of analysis.
- *Multi-level:* The model should capture the system at different levels of abstraction and precision to enable stepwise elaboration and validation.
- *Open and evolvable:* The model should highlight important alternative options to be considered in the RE process together with the selected options.
- *Traceable:* The source, rationale, and impact of modeling decisions should be easy to retrieve from the model.
- C*omprehensible:* The model should be easy to understand by stakeholders for further elicitation, evaluation, validation, and modification.

A goal-oriented approach to model building

In the general setting established by Part 1 of the book, Part 2 presents a goal-oriented, model-based approach to RE aimed at producing models that meet those requirements. Such models may capture both the system-as-is and the system-to-be.

To address the *completeness* requirement, a model will integrate multiple complementary views of the target system that cover the WHY-, WHAT-, and WHO-dimensions.

- The *intentional* view captures the system objectives as functional and non-functional goals together with their mutual contribution links.
- The *structural* view captures the conceptual objects manipulated in the system, their structure, and their inter-relationships.
- The *responsibility* view captures the agents forming the system, their responsibilities with respect to system goals, and their interfaces with each other.
- The *functional* view captures the services the system should provide in order to operationalize its functional goals.
- The *behavioral* view captures the required behaviors for the system to satisfy its behavioral goals; interaction scenarios illustrate expected behaviors of specific agent instances whereas state machines prescribe classes of behaviors for any such instance.

To address the *traceability* requirement, those different views will be connected through traceability links showing how items in one view are explained by or derived from items in another view.

To address the *precision* requirement and support various forms of RE-specific analysis, the models will be expressed in terms of semi-formal notations augmented with structured annotations defining each model item precisely. The annotations can be made optionally formal, where and when needed, in order to support more sophisticated forms of analysis.

To address the *understandability* requirement, the models will be visualized through diagrammatic notations, in the style of those introduced in Section 4.3, but compliant with the UML standards wherever possible – that is, wherever the corresponding system view is supported by the UML under a simple, semantically well-defined, and widely used form (Booch et al, 1999; Dobing & Parsons, 2006).

The next chapters introduce modeling techniques for capturing those various system views and for integrating them into a comprehensive system model. For each view, we will briefly introduce the modeling notations used, provide a precise yet simple semantics for them, and illustrate their use in our running case studies. The main emphasis, however, will be on well-grounded heuristics for constructing "good" models in the previously defined sense – this is where the difficulty lies in practice.

Chapter 8 is dedicated to the intentional view of the system. It presents techniques for modeling the system's functional and non-functional goals in terms of individual features, such as their specification, elicitation source, or priority, and inter-relationships, such as their contributions to each other, their potential conflicts, and their alternative refinements into software requirements and environment assumptions. The resulting model is called *goal model*.

Chapter 9 complements the modeling techniques from the previous chapter with a goal-based form of risk analysis. The focus there is on what could go wrong with overideal versions of the goal model, typically obtained in the first stages of modeling. Obstacles are conditions for goal obstruction that are modeled as goal-anchored variants of the risk trees introduced in Section 3.2. The resulting model is called *obstacle model*. New goals for a more robust system are then added to the goal model as countermeasures to the modeled obstacles.

Chapter 10 is dedicated to the structural view of the system. It presents techniques for building UML class diagrams to characterize, structure, and inter-relate the conceptual objects manipulated in the system. The resulting model is called *object model*. This chapter provides precise criteria for deciding whether a concept we want to capture in an object model is to be defined as an entity, attribute, relationship, event, or agent. It discusses object structuring mechanisms such as aggregation and specialization with inheritance. It also shows how a complete and pertinent object model can be derived from the goal model.

Chapter 11 is dedicated to the responsibility view of the system. The agents forming the system are modelled as active objects whose instances can restrict their behavior so as to meet the goals they are responsible for. Responsibility assignments rely on agent capabilities. The latter are defined in terms of ability to monitor or control the objects involved in goal formulations. An agent can also be decomposed into finer-grained ones with finer-grained responsibilities. It may be related to other agents through dependency links. The resulting model is called *agent model*. We will also see how context diagrams can be derived from the goal model. (Context diagrams show the interfaces among agents, see Section 4.3.3.)

Chapter 12 is dedicated to the functional view of the system. The services to be provided are modelled as operations performed by system agents under specific conditions to operationalize the goals from the goal model. The techniques in this chapter will help us identify such conditions in a systematic way from goal specifications. The resulting model is called *operation model*. We will also see how UML use cases can be derived in a straightforward manner from the operation model.

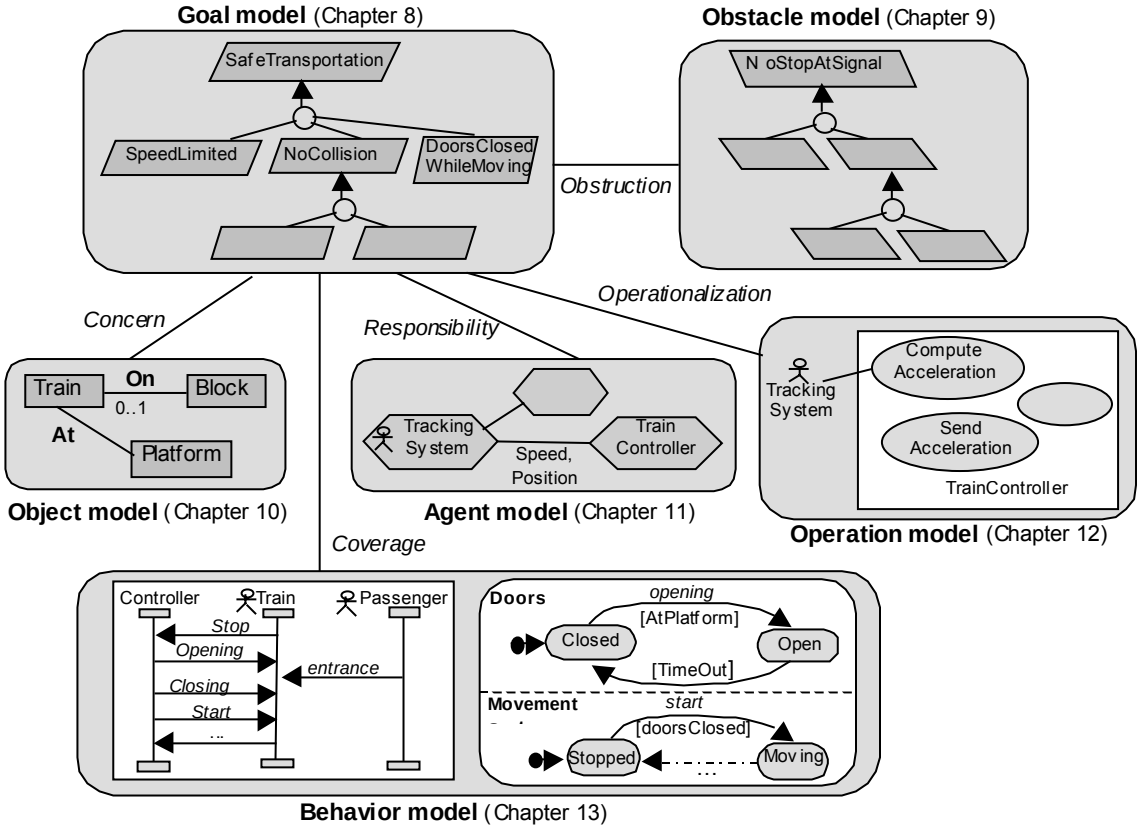
Chapter 13 is dedicated to the behavioral view of the system. A *behavior model* is composed of scenarios and state machines. At the instance level, UML sequence diagrams are used for capturing interaction scenarios among specific agent instances. This chapter will provide heuristics for building a comprehensive set of sequence diagrams to illustrate the identified goals and obstacles, or to identify goals and obstacles from them. At the class level, simple forms of UML state diagrams are used for representing state machines that capture the admissible behaviors of any agent instance. The

techniques discussed there will allow us to build state diagrams incrementally from interaction scenarios and from goal operationalizations.

Chapter 14 describes mechanisms for integrating the goal, object, agent, operation, and behavior models. These include the definition of a metamodel in terms of which the models can be defined and interrelated; the enforcement of inter-model consistency rules; and the grouping of related model fragments from different system views into cohesive model packages. The aggregation of the goal, object, agent, operation, and behavior models form the entire *system model*.

Chapter 15 concludes this second part of the book by presenting a constructive method for elaborating a full, robust, and consistent system model by incremental integration of the goal, object, agent, operation, and behavior sub-models. The requirements document is then produced in a systematic way by mapping the resulting model into some textual format annotated with figures. The produced document preserves the goal-oriented structure and content of the model, and fits prescribed documentation standards if required. The method will be shown in action there on a mine safety control system.

The figure hereafter oulines the complementary system views studied in Part 2. Their construction and use will be grounded on the general RE framework introduced in Part 1. Part 3 will then present a variety of techniques for reasoning about the models built in Part 2 in order to analyze, evaluate, and consolidate them.



Chapter 8 Modeling System Objectives with Goal Diagrams

This chapter is dedicated to the intentional view of the system we want to model. This view covers the *WHY*-dimension of requirements engineering. It is provided by a goal model.

A *goal* was defined in Chapter 7 as a prescriptive statement of intent the system should satisfy through cooperation of its agents. Such statement is formulated in terms of environment phenomena. The formulation is declarative – unlike operational procedures to "implement" it. We may need to evaluate, negotiate, weaken, or find alternatives to goals – unlike domain properties or hypotheses that are descriptive statements holding regardless of how system agents behave. Goals may refer to high-level, strategic, coarse-grained objectives the system should fulfill or to lower-level, technical, finer-grained prescriptions. The finer-grained a goal is, the fewer agents are required to satisfy it. A *requirement* was defined in Chapter 7 as a goal under responsibility of a single software agent; an *expectation* is a goal under responsibility of a single environment agent. The instances of the responsible agent are then the only ones required to restrict their behavior to satisfy the goal.

The *goal model* basically shows how the system's functional and non-functional goals contribute to each other through refinement links down to software requirements and environment assumptions. It may capture alternative ways of refining goals and potential conflicts among them. At its interface with other views of the system, the goal model captures inter-model relationships such as responsibility links between goals and system agents, obstruction links between goals and obstacles, reference links from goals to conceptual objects, or operationalization links between goals and system operations. The model also specifies individual features of each goal such as its name and precise specification, its type, category, priority, the elicitation source it comes from, and so forth. A formal specification of some goals may be optionally provided for further analysis.

Graphically, a goal model is represented by an AND/OR graph called *goal diagram*. As we will see, goal nodes in such diagrams are annotated by their features and connected through various types of edges. *Refinement* links indicate how a goal is AND-decomposed into conjoined subgoals. The same goal node can be the target of multiple such links; each of them indicates an alternative way of refining the goal into subgoals. Leaf goals along refinement branches represent software requirements or environment assumptions needed to enforce their parent goals. In a goal diagram, *conflict* links may interconnect goal nodes to capture potential conflicts among them. At the interface with other system views, a goal diagram may show alternative *responsibility assignments* connecting goal nodes to agents, *concern* links connecting goal nodes to the objects they refer to, *operationalization* links connecting goal nodes to the system operations ensuring them, and so forth.

Goal modeling makes it possible to capture the system *as-is* and *to-be* within the same model. In general, both system versions share high-level goals and differ along refinement branches of common parent goals.

The importance of the goal model derives from the central role played by goals in the RE process (see Section 7.4). In particular, we can derive other models in a systematic way from the goal model such as the object and operation models. Moreover, the goal model enables early forms of RE-specific analysis such as risk analysis, conflict analysis, threat analysis, or evaluation of alternative options (as Part 3 of the book will show).

Section 8.1 describes how goals are individually characterized within a goal model through various kinds of features annotating them. Section 8.2 discusses goal refinement as a basic mechanism for capturing goal contributions and for interrelating goals, domain properties, and hypotheses. Section 8.3 briefly introduces how conflicts among goals may be documented on the goal model for later

resolution. Section 8.4 introduces other link types to connect goal model items to corresponding items in the object, agent, operation, behavior, and obstacle models. Section 8.5 explains how alternative options can be captured in a goal model through alternative goal refinements and assignments. Section 8.6 puts all pieces together by showing how an entire goal model amounts to a special kind of AND/OR graph. Section 8.7 explains how a goal model can be further documented through features attached to goal refinements and assignments. Section 8.8 concludes this chapter by presenting a number of heuristics, tips, and patterns for guiding modelers in the difficult task of building goal models.

8.1 Goal features as model annotations

In a goal model, each goal is annotated by a number of features to characterize the goal individually. Some of these correspond to slots of the statement templates discussed in Section 4.2.1. To help visualize such annotations, Fig. 8.1 shows possible features of a variety of goals from our running case studies.

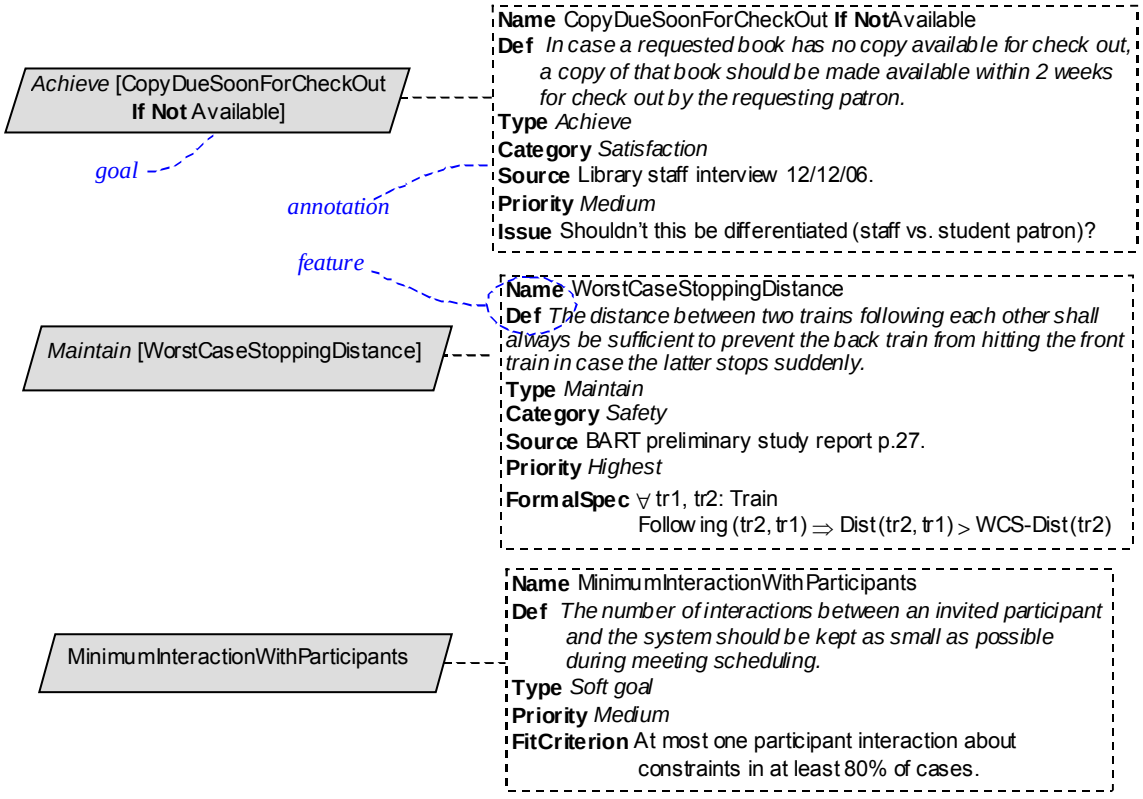


Figure 8.1 – Goal features as model annotations: examples

Each goal in a goal model is graphically represented by a parallelogram labelled by the goal's name, possibly prefixed by its type. The *Achieve* prefix on the first goal in Fig. 8.1 indicates that this goal is a behavioral goal having the specification pattern: ***[if** CurrentCondition **then] sooner-or-later** TargetCondition* (see Section 7.3.1). Similarly, the *Maintain* prefix on the second goal in Fig. 8.1 indicates that the goal is a behavioral one as well, but its specification pattern is: ***[if** CurrentCondition **then] always** GoodCondition*. The third goal is annotated as a soft goal, to be used for evaluating alternatives. The **Type** feature thus

indicates which class of prescribed or preferred behavior the goal refers to. Its range is {*Achieve, Maintain, SogtGoal*}.

Only two features are mandatory for any goal in a model: the goal's name and its specification. All others are optional.

- The **Name** feature must uniquely identify the goal throughout all views of the entire system model.
- The **Def** feature must precisely define, in natural language, what the goal prescribes in terms of phenomena that are monitorable and controllable in the system.

Goal names appearing in the goal model should be as suggestive as possible to enable their use as shortcuts for their complete definition. Names are just strings, however. For precise understanding, documentation, and analysis of goals, they may never replace a complete, adequate, and unambigous goal specification in the **Def** feature. Moreover, goal specifications are used as input for the model-based generation of the requirements document (see Section 16.4).

In addition to the **Type** feature discussed before, the optional features of a goal include the following.

- **Category**: This feature indicates which taxonomic categories the goal belongs to. For example, the first goal in Fig. 8.1 is concerned with satisfying agent requests whereas the second goal is concerned with keeping the system in safe states. The use of such information for heuristic analyses was discussed in Section 7.3.2.
- **Source**: This feature indicates where the goal is coming from – a stakeholder who brought it up during elicitation, a specific section in some preliminary documentl used during background studies, a standard regulation in the domain, and so forth. For example, the goal *Maintain[WorstCaseStoppingDistance]* in Fig. 8.1 appeared at some place in a study report issued at a preliminary stage of the project. The use of such information for traceability management was discussed in Section 6.3.
- **Priority**: This feature indicates a qualitative level of goal priority for comparison with competing goals. The use of such information for conflict resolution and prioritization was discussed in Section 3.1.3 and Section 3.4, repectively. For example, the goal *Maintain[WorstCaseStoppingDistance]* in Fig. 8.1 is of highest priority, like most safety goals. It should never be weakened in case of conflict with performance goals regarding train speed or frequency, for example.
- **Stability**: This feature indicates a qualitative level of estimated stability with respect to other comparable goals. The use of such information for change anticipation was discussed in Section 6.2. Lower-level goals are expected to be less stable than the higher-level goals they contribute to, as alternatives to the former might be considered to satisfy the latter.
- **FitCriterion**: This feature may annotate a *soft* goal to quantify the extent to which the goal should be met. It can be used for evaluating alternative options against it and for checking whether the goal is satisfactorily met by subgoals. Fit criteria were discussed and illustrated in Section 4.2.1. The aim of this feature is to make soft goal specifications measurable (see also Section 1.1.7). For example, the fit criterion annotating the soft goal *MinimumInteractionWithParticipants* in Fig. 8.1 provides a measurable threshold for what is meant by "as small as possible" in the goal specification.
- **FormalSpec**: This feature may annotate a behavioral goal to formalize its informal **Def** specification and thereby enable a variety of formal analyses. The use of this feature will be deferred until Chapters 17 and 18. A real-time temporal logic, similar to the one introduced in Section 4.4.2, will be used there to formalize *Achieve* and *Maintain/Avoid* goals. The **FormalSpec** feature allows goal-based models to be analyzed formally for adequacy, consistency, and completeness – in alignment

with the benefits of formal methods discussed in Section 4.4.7. Early formal analysis is especially relevant in the case of safety-critical or security-critical systems as suggested by the train system's goal *Maintain[WorstCaseStoppingDistance]* in Fig. 8.1.

- **Issue**: This process-level feature proves very useful in practice. It serves as a placeholder for recording questions raised about the goal during model elaboration. Such questions are to be addressed subsequently in the RE process. For example, the issue annotating the goal *Achieve[CopyDueSoonForCheckOut**IfNot**Available]* in Fig. 8.1 records a modeller's question that should still be addressed as different types of library users might deserve different service policies.

8.2 Goal refinement

The core of the goal model consists of a refinement graph showing how higher-level goals are refined into lower-level ones and, conversely, how lower-level goals contribute to higher-level ones. Refinement graphs were briefly suggested in Section 7.4. We look at them in full detail now.

In a refinement graph, an *AND-refinement* link relates a goal to a set of subgoals. This set is called *refinement* of the parent goal. Each subgoal in the refinement is said to *contribute* to the parent goal. The meaning of an *AND-refinement* is that *the parent goal can be satisfied by satisfying all subgoals in the refinement*.

For example, Fig. 8.2 shows a possible AND-refinement of the goal *BookRequestSatisfied* in our library system. Graphically, a refinement is represented by a small circle connecting the contributing subgoals to the refined goal; the latter is the target of the directed link. Semantically, this AND-refinement link expresses that the goal *BookRequestSatisfied* can be satisfied by satisfying the goal *CopyBorrowed**If**Available* and the goal *CopyDueSoonForCheckOut**IfNot**Available*. (The latter subgoal was characterized by an annotation in Fig. 8.1.)

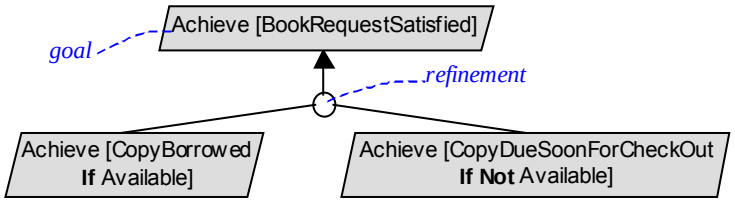


Figure 8.2 – AND-refinement

An AND-refinement of a goal G into subgoals $G_1, G_2, \ldots, G_n$ should ideally be complete, consistent, and minimal.

- *Complete refinement:* The satisfaction of all subgoals $G_1, G_2, \ldots, G_n$ should be sufficient for the satisfaction of the parent goal G in view of all known domain properties and hypotheses in *Dom*. In short:

  $$\{G_1, G_2, \ldots, G_n, Dom\} \models G$$

  where $S \models A$ means: "the statement A is always satisfied in any circumstance where all statements in S are satisfied". In other words, no subgoal is missing for the parent goal to be satisfied. The refinement is then called a *complete refinement*.

  A refinement which is arguably complete is graphically represented by a black circle. In Fig. 8.2, the refinement circle should be blackened as it can be argued to be complete – either a copy is available, and letting the requesting patron borrow it ensures that the book request is satisfied, or no copy is available, and making sure that a copy becomes available within two weeks for loan by the requesting patron ensures that the book request is satisfied. All possible cases are thus covered.

To provide such completeness arguments, properties and hypotheses about the domain may be needed as contextual information. We come back to this below, see the example in Fig. 8.4. Getting complete AND-refinements of behavioral goals is obviously essential for requirements completeness. A missing subgoal will result in missing requirements or assumptions to satisfy it. This is the most harmful type of RE error, as we saw it in Chapter 1. Completeness arguments should therefore be provided for mission-critical goals. Refinement patterns may be used informally to achieve this, see Section 8.8 below. Formal techniques for checking refinement completeness can be used as well, see Chapter 18.

- *Consistent refinement:* The subgoals, domain properties, and hypotheses may not contradict each other:

$$\{G_1, G_2, \ldots, G_n, Dom\} \not\models \textbf{false}$$

We clearly do not want the parent goal G to be trivially satisfied – as anything can be argued to hold in inconsistent situations.

- *Minimal refinement*: If one of the subgoals in the refinement is missing, G_j say, the satisfaction of the parent goal is no longer always guaranteed:

$$\{G_1, \ldots, G_{j-1}, G_{j+1}, \ldots, G_n, Dom\} \not\models G$$

In a minimal refinement, the contribution of each subgoal in the refinement is needed for the parent goal to be satisfied. We are not interested in imposing additional restrictions in the refinement that are not strictly required for the satisfaction of the parent goal. (Such additional restrictions might be needed for other reasons; they would then appear in refinements of parent goals capturing such other reasons.)

In an AND-refinement of a goal G, a subgoal may itself be AND-refined, and recursively. The parent goal G may thus be the root of an AND-refinement tree. Fig. 8.3 illustrates this for the AND-refinement in Fig. 8.2.

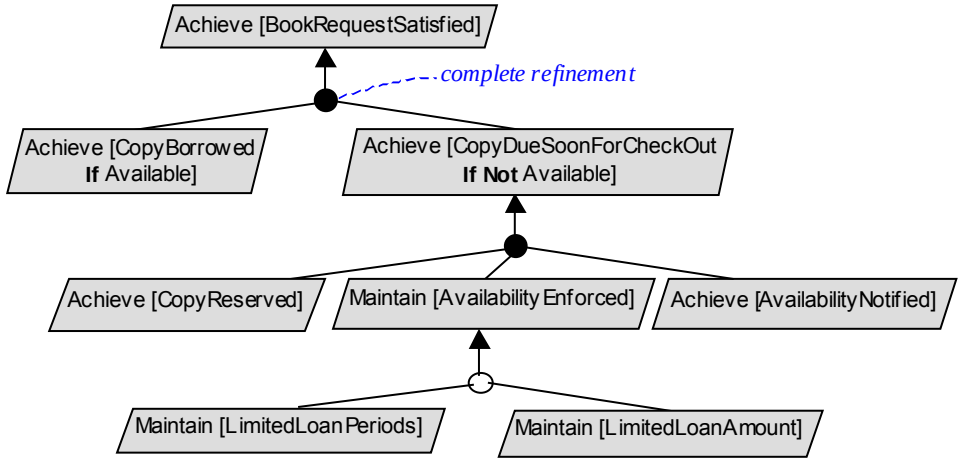


Figure 8.3 – AND-refinement tree

The leaf nodes in refinement trees are nodes that need not be refined further. These include goals whose responsibility can be assigned to single software agents (as requirements) or to single environment agents (as expectations), or else descriptive statements used in the refinement. The latter might be domain properties or hypotheses. (Those different types of statements were defined in Section 7.2, see Fig. 7.1.)

Fig. 8.4 shows leaf nodes in a refinement tree. Graphically, they may be differentiated by a bold border. The "home" shape is used for representing domain properties or hypotheses. Responsibility links are also shown; they are interface links with the agent model, see Section 8.4 below. Hexagons are used for representing system agents, with an inside "fellow" icon if the agent is an environment one.

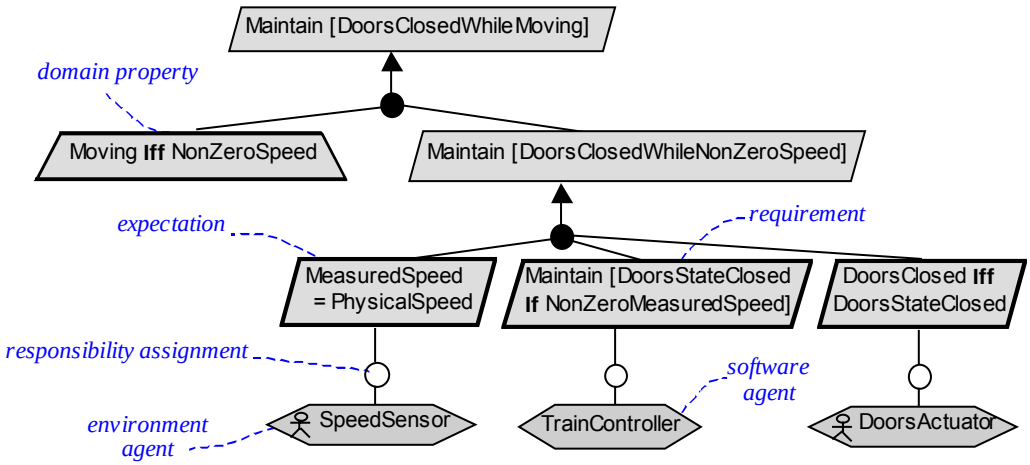


Figure 8.4 – Leaf nodes in an AND-refinement tree

Note that one single subgoal may sometimes be sufficient in a complete refinement (see the top refinement in Fig. 8.4).

Also note that the portion of the goal model showing AND-refinement links will in general be a directed acyclic *graph* rather than a tree. There might be multiple root goals, and a single goal node may contribute to multiple parent nodes. For example, the goal *HighFrequencyOfTrains* contributes to the goal *RapidTransportationOfPassengers* but also to the goal *TransportationCapacityIncreased* (see Fig. 8.5 below). Similarly, the goal *AccurateBookClassificationByTopic* in the library system contributes to multiple parent goals such as *AccurateAnswerToBiblioQuery* and *EasyCopyLocalizationInShelves*.

It is important to emphasize that refinement links are two-way links – one way showing goal decomposition and the other way showing goal contribution. We may thus identify refinement links *top-down*, asking ourselves how to satisfy a given goal by some AND-combination of subgoals; *bottom-up*, asking ourselves which parent goals a given goal contributes to; or in a hybrid way, proceeding bottom-up and then asking ourselves what other refinement links are required for the refinement of the identified parent goal to be a complete one. Goal modeling thus does not entail top-down decomposition, as mistakenly suggested sometimes in the literature on the subject. We come back to this important point in Section 8.8 while discussing heuristics for model building.

Goal refinement and satisfaction arguments. As suggested in Section 7.4, AND-refinement links in goal models support a rich structure of satisfaction arguments, each taking the form:

$$\{REFINEMENT, DOM\} \models ParentGoal$$

where *REFINEMENT* denotes a set of conjoined subgoals refining a goal *ParentGoal* possibly in conjunction with domain properties and hypotheses in *DOM*. By chaining such arguments bottom-up, we may show that a set of requirements and expectations ensure some parent goal, the latter ensuring its own parent goal together with others, and recursively, until some high-level goal of interest is thereby shown to be satisfied.

For example, consider the goal model fragment in Fig. 8.5 for our train control system. We might argue that the goals *SignalSafelyKeptToStop* and *TrainStoppedAtBlockEntry**If**StopSignal* together ensure the goal *Avoid[TrainsOnSameBlock]*; the latter ensures the goal *Avoid[TrainCollisions]*; the latter ensures the goal *SafeTransportation* together with others. At each step in such argumentation tree we should ask ourselves whether some subgoal, domain property, or hypothesis is missing for the refinement of the parent goal to be a complete one.

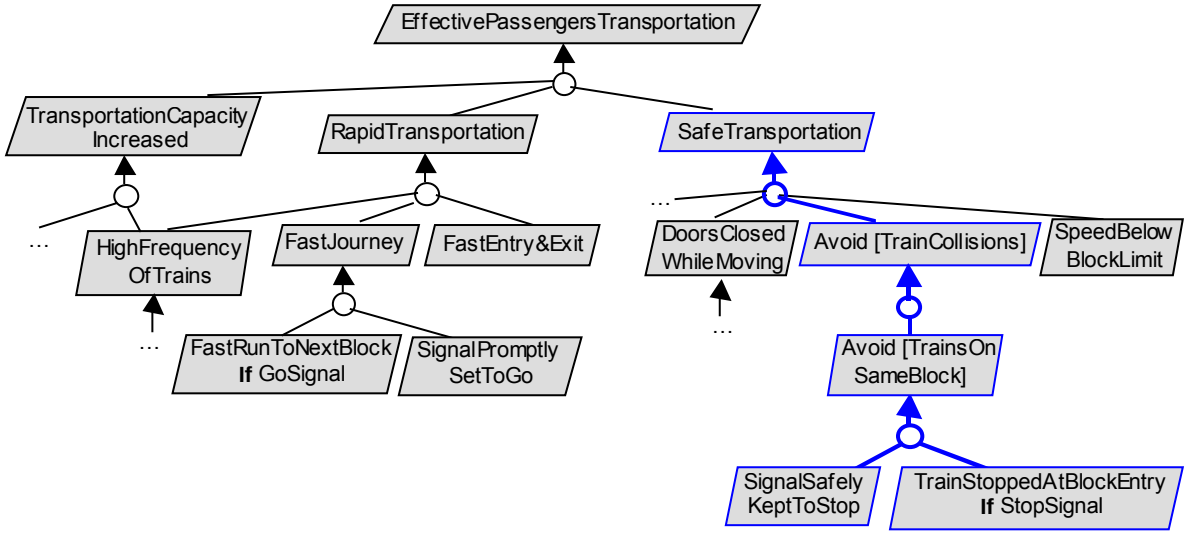


Figure 8.5 – Tree of satisfaction arguments on a goal model

When domain properties or hypotheses are used in satisfaction arguments, it is very important to check also whether these are *adequate*. Fatal RE errors may originate from wrong properties or unrealistic hypotheses. For example, Fig. 8.6 shows a correct goal refinement that involves an inadequate property about the domain (for better precision, we use simple propositions connected by implication/equivalence symbols instead of goal names). The wrong domain property there resulted in a major accident during landing on a rainy day at Warsaw airport, see Section 1.2.1. Chapter 18 will show how this error could have been detected using formal obstacle analysis.

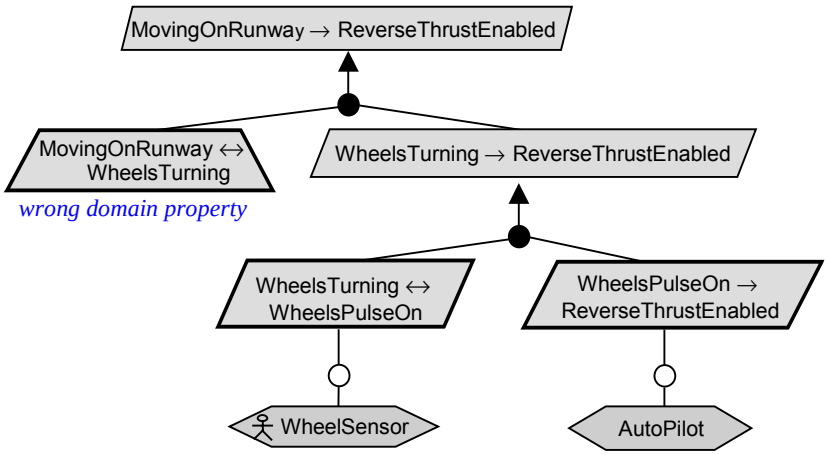


Figure 8.6 – Correct refinement based on wrong domain property for the airbus A320 braking logic

8.3 Representing conflicts among goals

Beside positive contributions among goals, there might be negative ones. Section 3.1.1 introduced the notion of divergence among statements emerging from the RE process. A divergence captures a *potential* conflict, where some statements become logically inconsistent if a boundary condition becomes true. Goals in a refinement graph might be potentially conflicting in that sense.

Roughly, the goals G_1, G_2, …, G_n are *divergent* in a domain *Dom* if we can find a feasible boundary condition *B* under which the goals cannot be satisfied together:

$$\{G_1, G_2, \ldots, G_n, B, Dom\} \models \textbf{false}$$

(Section 16.2.1 will be provide a more complete definition of divergence.)

Such potential conflicts among goals in a model often occur when the goals originate from multiple sources or viewpoints. For example, the goal *LimitedLoanPeriods* in Fig. 8.3 may have been formulated by library staff stakeholders whereas somewhere else in the model we might find the goal *CopiesBorrowedAsLongAsNeeded* that comes from patron interviews.

We must of course detect such situations and resolve them in some appropriate way. As discussed in Section 3.1.3, resolution should not take place too early in the RE process; conflicting statements might be a source for further elicitation of useful information. It may however be worth capturing potential conflicts in the goal model, when they are suspected, in order to get back to them subsequently for deeper analysis (such analysis will be discussed in Chapters 16 and 18).

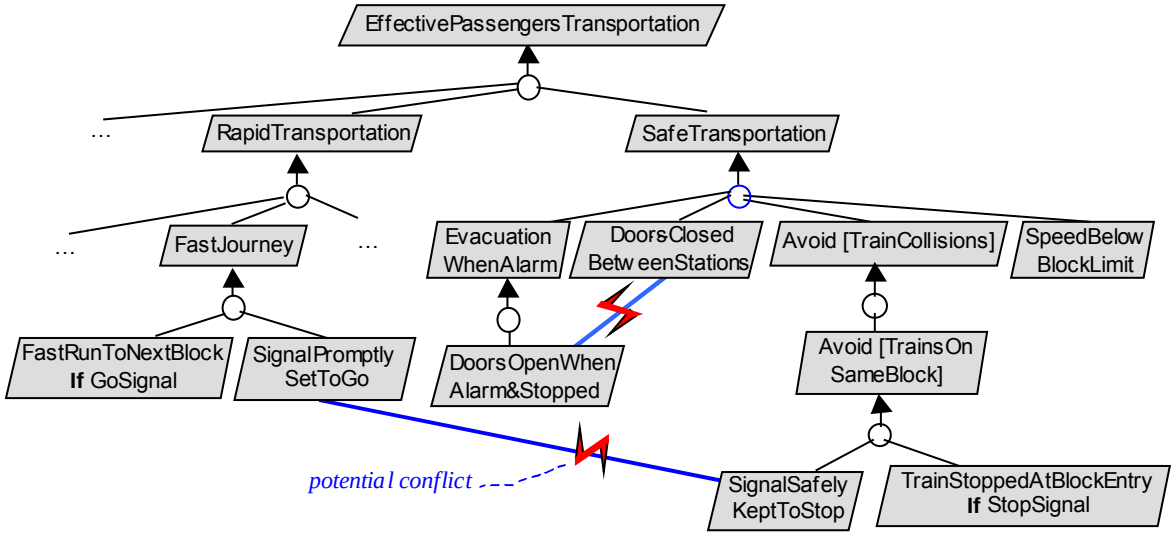


Figure 8.7 – Capturing potential conficts in a goal model

Graphically, a potential conflict among goals is represented by a "flash" icon on a link connecting them. Fig. 8.7 illustrates some possible conflicts in a goal model for the train control system. The goals *DoorsClosedBetweenStations* and *DoorsOpenWhenAlarm&Stopped* are potentially conflicting there; the former prescribes train doors to remain always closed between two successive stations whereas the latter prescribes doors to be open in case an alarm is raised and the train is stopped. A feasible boundary condition for conflict consists of a train being stopped between two stations with an alarm raised; under this condition the two goals cannot be satisfied together. (In Chapter 18 we will see how such boundary condition can be derived formally from the goal specifications.) Similarly, the goals *SignalPromptlySetToGo* and *SignalSafelyKeptToStop* are potentially conflicting; a too prompt "go" signal

guarding the block a train is waiting to enter might result in an unsafe situation where two trains are on that block (as the previous train did not have enough time to leave it); this would violate the goal *SignalSafelyKeptToStop*.

Soft goals often prescribe some target quantity to be increased or reduced. A frequent situation of potential conflict arises when one soft goal prescribes some quantity to be increased whereas another soft goal prescribes a related quantity to be reduced. For example, the quantity to be increased may refer to some functionality or quality whereas the related quantity to be reduced refers to cost –e.g., *JournalCoverageIncreased* vs. *OperationalCostsReduced* in the library system, or *MoreControllersHired* vs. *RunningCostsDecreased* in an air traffic control system. In such situations, it is important to annotate the soft goal with a **FitCriterion** feature so that feasible boundary conditions for conflict can be determined and acceptable thresholds can be found for conflict resolution.

8.4 Connecting the goal model with other system views

The core of a goal model consists of an annotated refinement graph where potential conflicts may be indicated. In addition to refinement and conflict links, the goal diagram representing the goal model generally shows interface links between the goal model and the other submodels of the system model.

- The ***responsibility*** link type is defined at the interface between the goal model and the agent model. A responsibility assignment of a goal to an agent means that the agent is the only one required to restrict its/her behavior in the system so as to satisfy the goal.

  Responsibility links were already illustrated in Fig. 8.4 and Fig. 8.6. They will be further discussed when the agent model will be described, see Chapter 11.

- The ***obstruction*** link type is defined at the interface between the goal model and the obstacle model introduced for risk analysis. A goal is obstructed by an obstacle if the satisfaction of the obstacle prevents the goal from being satisfied.

  For example, the goal *TrainStoppedAtBlockSignal**If**StopSignal* in Fig. 8.5 is obstructed by the obstacle *SignalNotVisible* or the obstacle *TrainDriverUnresponsive*; the condition of an unresponsive driver failing to react to the "stop" command issued by the train controller does indeed prevent that goal from being satisfied.

  Obstruction links will be detailed Chapter 9 when the obstacle model will be described.

- The ***concern*** link type is defined at the interface between the goal model and the object model. A goal concerns a conceptual object if its specification refers to this object.

  For example, the goal *Avoid[TrainsOnSameBlock]* concerns the conceptual entities *Train* and *Block*, and the conceptual association *On* between *Train* and *Block*.

  Concern links will be detailed in Chapter 10 when the object model will be described.

- The ***operationalization*** link type is defined at the interface between the goal model and the operation model. A leaf goal is operationalized by a set of operations if the specification of these operations ensures that the goal is satisfied.

  For example, consider the leaf goal *DoorsStateClosed**If**NonZeroMeasuredSpeed*, defined by

  $$MeasuredSpeed \neq 0 \rightarrow DoorsState = 'closed'.$$

  This goal is operationalized by operations such as *StartTrain* and *OpenDoors* through preconditions that restrict their applicability so as to satisfy the goal – e.g., "$MeasuredSpeed = 0$" is a precondition on the operation *OpenDoors* for the satisfaction of that goal.

  Operationalization links will be detailed in Chapter 12 when the operation model will be described.

- The ***coverage*** link type is defined at the interface between the goal model and the behavior model. A behavioral goal *covers* a scenario or a state machine if the sequences of state transitions expressed by the scenario or state machine capture some of the behaviors prescribed by the goal.

  Section 7.6.1 already introduced this link type and illustrated it for the goal *Maintain[DoorsClosedWhileMoving]*, see Fig. 7.8. Coverage links will be further discussed in Chapter 13 when the behavior model will be described.

8.5 Modeling alternative options

As mentioned in the introduction to Part 2, it is important for a RE model to capture multiple options that should be considered while engineering the requirements. When they are made explicit in the model, such options can be discussed with stakeholders to assess their pros and cons; they can be evaluated against soft goals; they can be negotiated with decision makers to reach agreement on most satisfactory ones. Multiple alternatives in a model may also capture multiple versions of the modeled system (variants or revisions, see Chapter 6).

A goal model makes it possible to capture two kinds of alternative options.

- A goal might be refinable in different ways; each will result in a different system, satisfying a different set of subgoals. Alternative goal refinements are discussed in Section 8.5.1.
- A leaf goal might be assignable to different agents in the system; each alternative assignment will result in a different system, with a different distribution of responsibilities. Alternative responsibility assignments are discussed in Section 8.5.2.

8.5.1 Alternative goal refinements

In a refinement graph, a goal node can be the target of multiple AND-refinements. Each refinement is then called *alternative* for achieving the parent goal. An alternative refinement is thus one set of subgoals, among others, that AND-refines the parent goal (see the definition of refinement in Section 8.2). The meaning of multiple alternative refinements is that *the parent goal can be satisfied by satisfying all subgoals from any of the alternative refinements*.

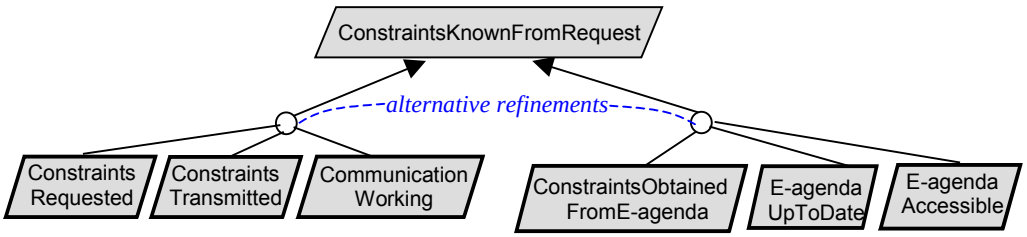


Figure 8.8 – Alternative goal refinements in the meeting scheduling system

For example, Fig. 8.8 shows two alternative refinements of the goal *ConstraintsKnownFromRequest* in the meeting scheduling system. This goal states that "*the time/location constraints of the invited participants listed in the meeting request shall be known to the meeting scheduler*". The two refinements connected to the goal *ConstraintsKnownFromRequest* in Fig. 8.8 express that this goal can be satisfied by satisfying the subgoals *ConstraintsRequested*, *ConstraintsTransmitted*, and *CommunicationWorking*, or, alternatively, by satisfying the subgoals *ConstraintsObtainedFromE-agenda*, *E-agendaUpToDate*, and *E-agendaAccessible*. (In the second alternative, the goal *E-agendaUpToDate* is an expectation on participants, in the category of accuracy goals.)

Fig. 8.9 illustrates alternative goal refinements in our other case studies. In the train control system, the goal *Avoid[TrainCollisions]* can be satisfied by making sure that a block will always contain one train at most (left alternative in Fig. 8.9) or, alternatively, by allowing multiple trains to be on the same block provided a safe stopping distance is maintained between successive trains (right alternative in Fig. 8.9, see the characterization of this goal in Fig. 8.1). In the library system, the goal *ComprehensiveLibraryCoverage* can be satisfied by satisfying the goal *EffectiveBookSupply* (left alternative in Fig. 8.9) or, alternatively, by satisfying the goal *AccessToForeignDigitalLibrary* (right alternative in Fig. 8.9).

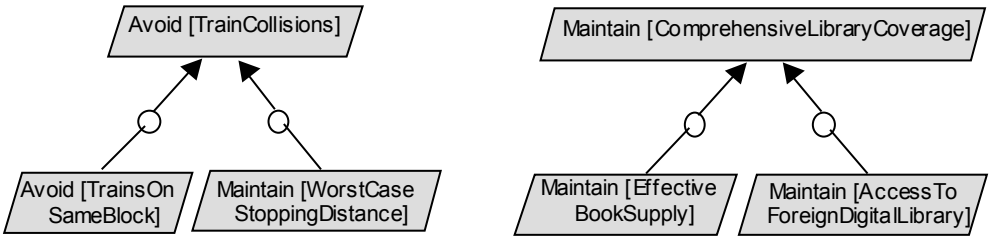


Figure 8.9 – Alternative refinements in the train control and library systems

Alternative goal refinements generally result in different system designs that will produce different versions of the system. For example, the left alternative in Fig. 8.8 will result in a meeting scheduler with interaction through email whereas the right, e-agenda alternative will result in a meeting scheduler with no interaction with participants for getting their constraints. Similarly, the left alternative for the train system in Fig. 8.9 corresponds to a system design incorporating signal management at every block whereas the right alternative corresponds to a system design without block signals but with dedicated management of acceleration commands to trains. Likewise, purchasing books or subscribing to digital libraries correspond to fairly different system designs for achieving comprehensive coverage.

A system might of course combine multiple alternatives to be considered under different conditions within the same version. In our meeting scheduling example, a hybrid of the two previous designs, with e-mail or e-agenda interaction depending on the type of participant, should certainly be worth considering. This would however correpond to a third alternative refinement in Fig. 8.8, producing a third possible version of the system.

Each alternative goal refinement may have its pros and cons, to be evaluated and compared for selection of a best option (see Section 3.3). The pros should correspond to soft goals in the model. If this is not the case, they should be added to the model as new soft goals, possibly to be refined. Similarly, the cons should be conflicting with soft goals; the latter should be added and refined if they are not found in the model. Alternative options are thus a source of further goal elicitation. We come back to this in Section 8.8. Section 16.3 will present techniques for evaluating alternative options based on their contribution to soft goals.

8.5.2 Alternative responsibility assignments

As introduced in Chapter 1, the WHO-dimension of requirements engineering concerns the distribution of responsibilities among system agents. We need to carefully analyze who is going to be responsible for what.

At the interface between the goal model and the agent model, *responsibility* links connect software requirements and environment expectations to corresponding agents. The meaning of assigning a leaf

goal to an agent is that the instances of this agent will be the ones (and the only ones) required to restrict their behavior so as to satisfy the leaf goal.

In general, multiple agents might be candidate for restricting their behavior to ensure some leaf goal. The responsibility for that goal is then assignable to any of them. We capture this in the model through multiple alternative responsibility links between the leaf goal and the candidate agents. Each alternative link is called an *assignment*.

Alternative goal assignments result in different system versions. Such versions differ by the distribution of responsibilities among the agents forming the system. From one version to the other, more or less functionality might be automated.

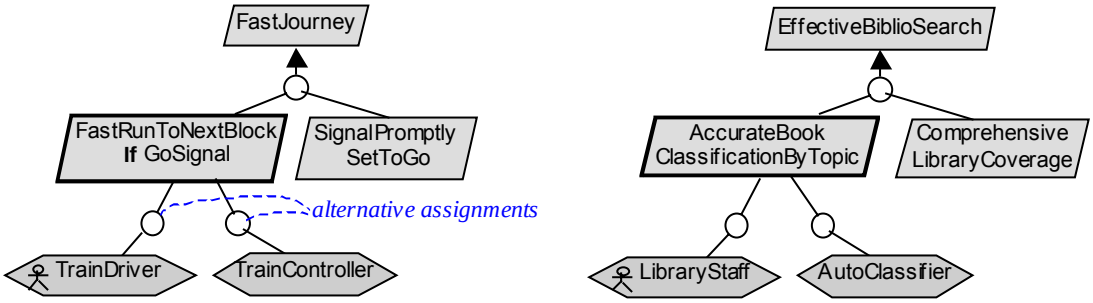


Figure 8.10 – Alternative responsibility assignments

For example, two alternative assignments might be considered for the goal *FastRunToNextBlock**If**GoSignal* in our train system (see Fig. 8.10). This goal might be under the responsibility of the *TrainDriver* agent or under the responsibility the *TrainController* agent in a driverless alternative with increased automation. In the library system, the goal *AccurateBookClassificationByTopic* is assignable to the *LibraryStaff* agent or, alternatively, to a software component that would retrieve relevant keywords from electronic abstracts supplied by publishers and classify books accordingly (this component is named *AutoClassifier* in Fig. 8.10).

Like for alternative goal refinements, alternative assignments have their respective pros and cons that should be made explicit as soft goals in the goal model. For example, the assignment of the goal *AccurateBookClassificationbyTopic* to the *AutoClassifier* agent might increase development costs and sometimes produce bizarre classifications; the alternative assignment of this goal to the *LibraryStaff* agent might result in increased load of library staff and in delayed accessibility of books waiting for classification. In the train system, the goal *DoorsClosedWhileNonZeroMeasuredSpeed* was assigned to the *TrainController* agent in Fig. 8.4. Alternative assignments might connect this goal to the *TrainDriver* agent or to the *Passenger* agent. The responsibility assignment to the *TrainDriver* agent results in transportation delays and in driver overload; those cons are conflicting with the soft goals of rapid transportation and reduced driver load, respectively. The responsibility assignment to the *Passenger* agent results in passengers getting control on door opening; this cons is conflicting with safe transportation goals.

The pros and cons of alternative assignments need to be evaluated and compared for selection of best options. (Section 16.3 will present techniques for evaluating alternatives based on their contribution to soft goals.) As seen from the preceding examples, the goal assignments that will be selected among the considered alternatives determine what will be automated in the system-to-be and what will not – and, correspondingly, what are going to be the requirements on the sofware-to-be and the expectations on the environment. As a result, the *boundary* between the software-to-be and its environment will be established.

Note that alternative goal refinements generally entail different distributions of responsibilities, as illustrated at the bottom of Fig. 8.11 for the meeting scheduling system.

8.6 Goal diagrams as AND/OR graphs

The previous sections showed piecewise what goal diagrams look like. Roughly, we have seen graphs where goal nodes are connected together through refinement and conflict links, and connected to items from other models through interface links such as *responsibility* or *operationalization* links. Looking more closely at such graphs, we have seen other types of nodes, namely, refinement nodes, "home" nodes capturing domain properties or hypotheses, and nodes from other models that are connected to goals through such interface links. We have also seen that multiple refinements or assignments of the same goal node capture alternative options.

Fig. 8.11 shows a larger fragment of a goal diagram for our meeting scheduling system. It contains high-level goals, requirements, expectations, refinement/contribution links, and some possible responsibility assignments.

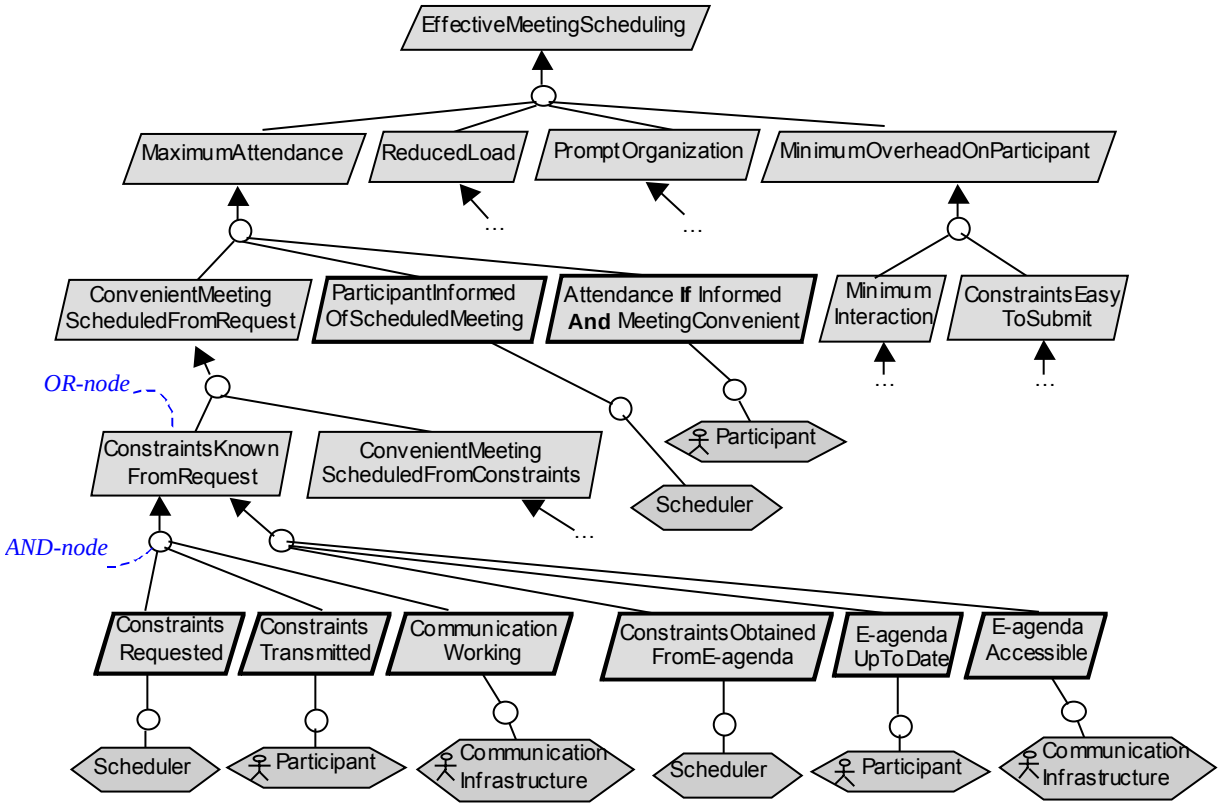


Figure 8.11 – Goal model fragment for the meeting scheduling system: AND-nodes and OR nodes

The core of a goal diagram shows alternative AND-refinements of the system's goals. It amounts to a special kind of graph known as *AND/OR graph*. Non-leaf *goal* nodes are OR-nodes whereas *refinement* nodes are AND-nodes (see Fig. 8.11).

- An *OR-node* is satisfied provided one of its successor nodes is satisfied.
- An *AND-node* is satisfied provided all its successor nodes are satisfied.

(Note that some nodes might have one successor node only.) This semantics of *AND* and *OR* nodes is to be taken in a strict, clear-cut sense for behavioral goals. For soft goals, the wording "satisfied" is to be understood as "satisficed" or "more or less satisfied" (see Section 7.3.1).

8.7 Documenting goal refinements and assignments with annotations

The same way as we further characterize goals through annotations, we can attach useful features to refinements and assignments for better documentation of the goal model. These features are all optional.

- **Name**: This feature can be attached to a refinement or to an assignment for unambiguous reference when dealing with alternatives, e.g., "the ConstraintsThroughE-agenda alternative" for the alternative goal refinement on the right in Fig. 8.8 or "the DriverlessStart alternative" for the alternative assignment to the TrainController agent in Fig. 8.12.
- **SysRef**: The *system reference* feature can be attached to a refinement or to an assignment in order to indicate which alternative is taken in which version of the system. For example, the alternative assignment DriverlessStart and the alternative refinement AccelerationControl in Fig. 8.12 are *both* taken in the system version named SystemToBe.
- **Status**: This graphical feature can be attached to a refinement to indicate whether the refinement is arguably complete for satisfying the parent goal. The value "complete" is represented by blackening the refinement circle as introduced before (see Figs. 8.4 or 8.12).
- **Tactic**: This feature can be attached to a refinement to document the tactic used for producing it. For example, "Goal decomposition by cases" is a tactic that was used for refining the goal *BookRequestSatisfied* in Fig. 8.2. The tactic "Guard introduction" is attached to the refinement of the goal FastJourney in Fig. 8.12 to explain how this refinement was found.

  A number of tactics are available as refinement patterns to help modelers refine goals. Section 8.8 hereafter will discuss them in detail. Documenting a model with the tactics used for building it makes it much easier to understand.

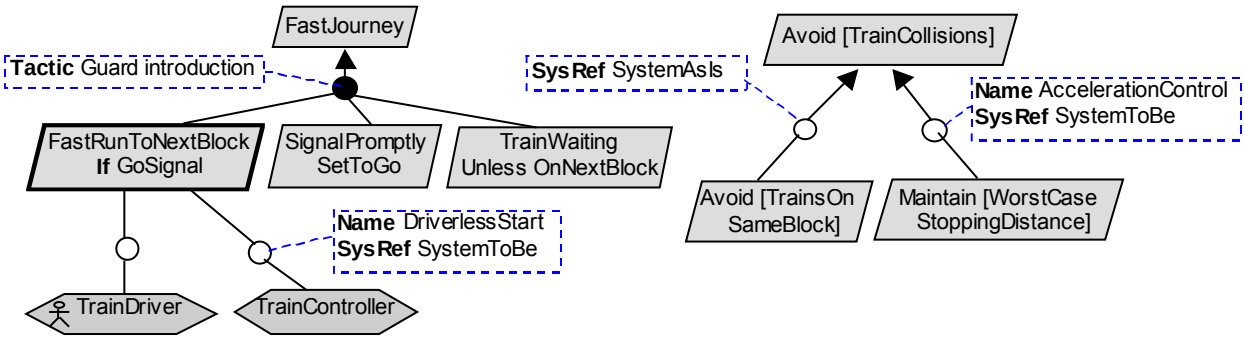


Figure 8.12 – Annotating refinements and assignments

The annotation mechanism can be applied to other link types used for modeling the system. In particular, the *conflict* links introduced in Section 8.3 may be annotated with the boundary condition making the corresponding goals conflicting.

8.8 Building goal models: heuristic rules and reusable patterns

Building a complete, consistent, minimal, and adequate goal model for a complex system turns to be surprisingly difficult in practice. This task is especially critical in view of the central role of goals in the RE process (see Section 7.4).

This section describes a variety of heuristic rules, tips, bad smells, and common refinement patterns to support the model building process. These may guide us in the elaboration of a first draft of the goal model. Such draft will subsequently be enriched from the building of the obstacle, object, agent, operation, and behavior models, as the next chapters will show. The resulting model needs to be consolidated through various forms of analysis, discussed in Chapters 16-18.

We first consider the elicitation of preliminary goals to start with (Section 8.8.1). Rules for enriching the resulting sketches and for scoping them are provided next (Sections 8.8.2 and 8.8.3). Some common pitfalls are then reviewed (Section 8.8.4). Finally, Section 8.8.5 describes a number of reusable refinement patterns that encode common tactics for goal refinement.

8.8.1 Eliciting preliminary goals

The building of a preliminary draft of the goal model should proceed from the early stages of domain analysis and requirements elicitation. Several heuristics are available for eliciting individual goals to start with. We can analyze the current objectives and problems in the system-as-is, search for intentional keywords in elicitation material, and browse goal taxonomies to explore relevant instantiations.

(H1) Analyze the current objectives and problems in the system-as-is. As we saw it in Section 1.1.6, the early phases of the RE process are partly concerned with the identification of various types of objectives.

- There are strategic objectives of the organization, business goals, and policies that are pervasive through any version of the system. When they appear during elicitation, they should be considered as candidate *high-level goals* for our goal model, to be connected later on to finer-grained contributing goals. In some cases policies might already be connected to underlying business goals through contribution links.

  For example, "*Effective access to state-of-the-art knowledge*" is a strategic objective of a university that should be found in any version of a library system. Privacy policies regarding staff and students might be relevant as well. In our train system, we might elicit strategic objectives such as "*Serve more passengers*"or "*Reduce operational costs*".

- There are domain-specific objectives that should be preserved from the system-as-is to the system-to-be. Each of these should be found in our goal model.

  For example, "*Satisfy book requests*" or "*Accurate classification of books*" are domain-specific concerns that might emerge from domain analysis and are likely to be found in any version of a library system.

- Requirements elicitation is partly concerned with the analysis of problems with the system-as-is in the light of opportunities to be exploited. Each such problem might raise the candidate goal of addressing it in the system-to-be. Multiple candidates might be considered as alternatives in the goal model, namely, to *avoid*, *reduce*, or *mitigate* the problem or its causes by exploiting technology opportunities.

  For example, one of the problems with the library system-as-is is that "*bibliographical search is restricted to library opening hours*" (see the list of stakeholder complaints in Section 1.1.2). The goal

"*bibliographical search accessible from anywhere at anytime*" might be introduced in the goal model to counter this complaint. For the meeting scheduling system, one of the problems emerging from domain analysis is that "*people are unnecessarily inconvenienced by scheduling messages they are not concerned with*". The soft goal "*avoid unnecessary interactions with invited participants*" might be identified as a result – to be subsequently refined so as to make it precise what "*unnecessary interactions*" means.

(H2) Search for goal-related keywords in elicitation material. Another simple, cheap, and quite effective technique for identifying preliminary goals for the system-to-be consists of applying some standard text search engine to find prescriptive, intentional, or amelioration keywords in interview transcripts and other preliminary documents involved in the elicitation process. The keywords are predefined in a table driving the search. The returned phrases containing such keywords are considered for goal formulations. The underlying justification is that a goal by definition is a prescriptive statement of intent. Many goals moreover prescribe some amelioration with respect to the system-as-is. Table 8.1 shows a table of goal-related keywords that might drive goal search in elicitation material.

Prescriptive	*shall, should, must, has to, to be, may never, may not, should never, should not, …*
Intentional	*in order to, so as to, so that, objective, aim, purpose, achieve, maintain, avoid, ensure, guarantee, want, wish, motivate, expected to, …*
Amelioration	*improve, increase, decrease, reduce, enhance, enable, support, provide, make, …*

Table 8.1 - Keywords for goal search in elicitation material

For example, a returned phrase such as "*passengers at a station should be informed in time about train arrivals*" may call for the introduction of a corresponding goal in the goal model.

Although simple and useful, such blind search has of course limitations. False positives may be obtained. The technique is applied to raw material that may contain many of the defects discussed in Section 1.1.7. It is also sensitive to specific formulations. Each returned phrase must therefore be analyzed in its context for adequacy and precision.

The intentional keywords listed first in Table 8.1 are however fairly accurate in capturing intents. In particular, keywords such as "*in order to*", "*so as to*", or "*so that*" are especially useful. When they are found, phrases like:

"In order to *X,* … has to *Y*", "… shall *Y* to *X*", "*Y* so as to *X*", "*Y* so that *X*"

yield both a goal *X* and a refinement link from *X* to a contributing subgoal *Y*. For example, consider the following phrase from the case study description in Section 1.1.2:

The distance between two trains following each other shall always be sufficient to prevent the back train from hitting the front train in case the latter stops suddenly.

This phrase suggests a goal *Avoid [TrainCollision]* and a subgoal *Maintain [WorstCaseStoppingDistance]* contributing to it.

(H3) Instantiate goal categories. Section 7.3.2 described a taxonomy of functional and non-functional goal categories. We can browse such taxonomy and, for each leaf node in the classification tree, look for system-specific instantiations of it.

For example, we might ask questions about the train system such as: "is there any *information* goal concerning passengers?". As a result, we might elicit the goal "*passengers at a station should be informed*

about train arrivals" mentioned previously. For the meeting scheduling system, we might look for instantiations of *interoperability* goals in view of foreign services the meeting scheduler will have to interact with. We might also look for *confidentiality* goals concerning participant information. In the library system, we might look for instantiation of *accuracy* goals relating physical books (and book copies) to their software "image".

8.8.2 Identifying goals along refinement branches

When a goal is found and its features are made precise, we should look for subgoals contributing to it and for parent goals the goal contributes to. Several heuristics are available to support this task.

(H4) Ask HOW and WHY questions. Let *G* be a goal already identified. We can systematically identify subgoals and parent goals of *G* by asking two kinds of questions about this goal.

- *Goal refinement through HOW questions*. Subgoals of *G* are found by asking questions such as:
  - "how can *G* be satisfied?",
  - "is this subgoal sufficient or is there any other subgoal needed for satisfying *G*?"
- *Goal abstraction through WHY questions*. Parent goals of *G* are found by asking questions such as:
  - "why should *G* be satisfied by the system?",
  - "is there any other parent goal that *G* contributes to?"

A frequent goal acquisition pattern consists of asking a *WHY* question about goal *G*, directly followed by a *HOW* question about the parent goal just found, in order to find "brothers" of *G* that might be missing for the parent goal to be satisfied.

For example, a *WHY* question about the goal *TrainStoppedAtBlockSignal**If**StopSignal* in Fig. 8.5 would result in the identification of the goal *Avoid[TrainsOnSameBlock]*. A *HOW* question about the latter would then lead to the goal *SignalSafelyKeptToStop* that might have been overlooked.

Fig. 8.13 illustrates such HOW/WHY acquisition process on the library system. During interviews, library staff might have expressed the concern of limiting loan periods in time – hence the goal Maintain[LimitedLoanPeriods] at the bottom of Fig. 8.13. By asking a *WHY* question about this goal we might identify the upper goal Maintain[AvailabilityEnforced]. A new *WHY* question about the latter goal might lead to the identification of the upper goal Achieve[CopyDueSoonForCheckOut**IfNot**Available]. Back to the previous goal Maintain[AvailabilityEnforced], we might ask a *HOW* question about it to find the missing subgoal Maintain[LimitedLoanAmount] that contributes to it as well.

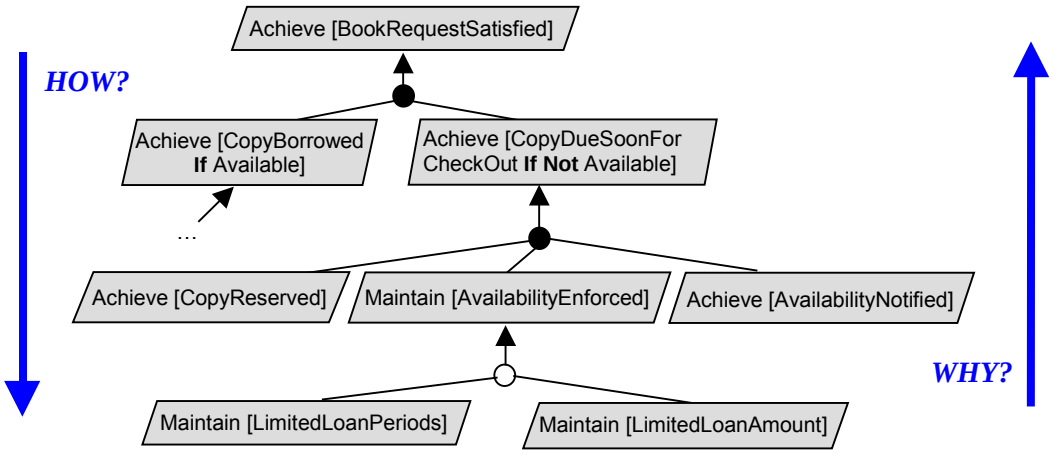


Figure 8.13 – Top-down and bottom-up identification of goals along refinement branches

Every time we ask a *HOW* question about a goal we can ask a *HOW ELSE* question next, in order to explore alternative options for refining that goal – see Figs. 8.8 and 8.9 for examples.

WHY questions can also be asked about more operational material in order to identify the underlying goals. This applies particularly to *scenarios* that frequently arise in elicitation sessions (see Section 2.2.5).

Fig. 8.14 illustrates this point for the meeting scheduling system. The goals ConstraintsKnownFromRequest and ParticipantsInformed, appearing in Fig. 8.11, might have been elicited that way from episodes illustrating them in the scenario shown in Fig. 8.14.

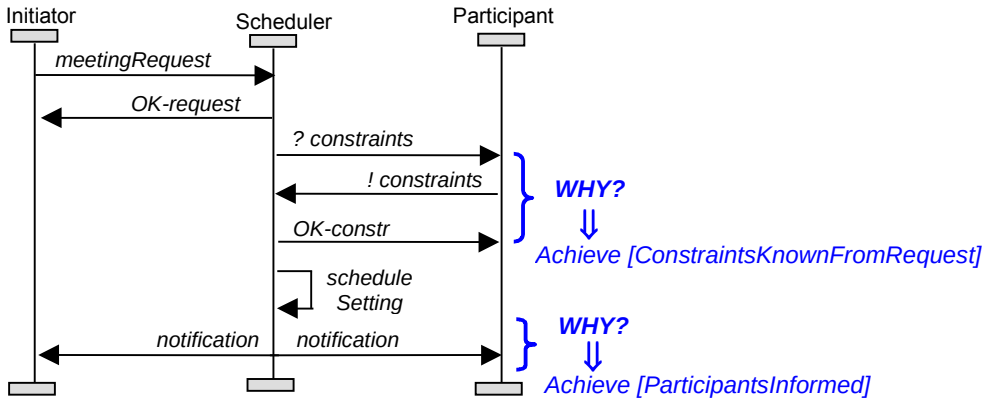


Figure 8.14 – Goal identification from *WHY* questions on scenario episodes

(H5) Split responsibilities. We may constrain *HOW* questions by requiring the resulting subgoals to involve fewer potential agents in their satisfaction, when such agents are easily identifiable. The set of potential agents that need to cooperate for the satisfaction of the parent goal is thereby decomposed in subsets associated with corresponding subgoals. The use of this heuristic is important for ensuring that the refinement process makes progress towards a stage where all finer-grained goals are assignable as requirements on software agents or expectations on environment agents.

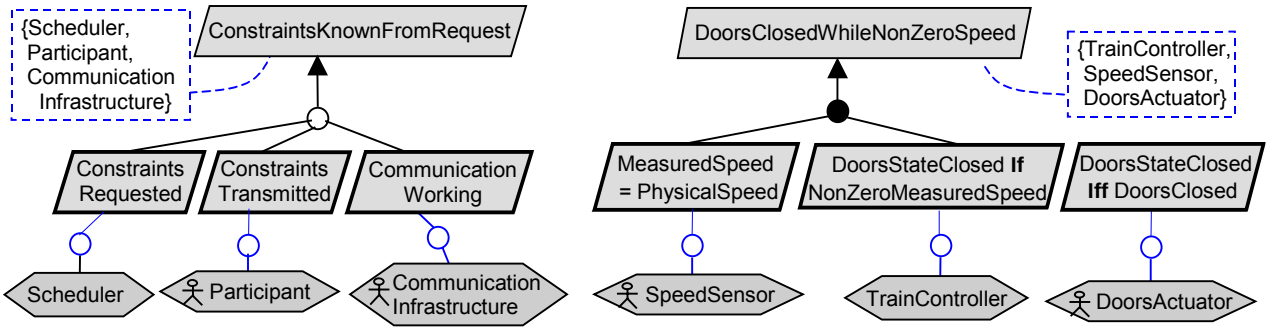


Figure 8.15 – Goal refinement by splitting responsibilities

Fig. 8.15 shows how this heuristic was applied in two refinement examples that appeared previously in this chapter. For example, the goal ConstraintsKnownFromRequest should involve the agents Scheduler, which must know the constraints of participants; Participant, who needs to provide such constraints; and CommunicationInfrastructure, which must communicate the constraints. The goal refinement was driven by a *HOW* question constrained by the splitting of responsibilities among those agents.

(H6) Identify soft goals by analyzing the pros and cons of alternative refinements. As already noted in Section 8.5, a pros may suggest a soft goal that might not have been identified yet. If so, the corresponding refinement should be connected to this soft goal through a contribution link. In a dual way, a cons may suggest a negated soft goal that might not have been identified yet. If so, the corresponding refinement should be connected to this soft goal via a conflict link.

For example, the two alternative refinements for the meeting scheduler to know the constraints of invited participants have respective merits (see Fig. 8.16). The ConstraintsThroughEmail alternative is likely to yield accurate constraints, as people know exactly what their actual constraints are. This alternative also results in invited people being pre-informed that the meeting will take place in the specified date range. On the other hand, the ConstraintsThroughE-agenda alternative contributes to ensuring minimum interaction and inconvenience with invited participants. For each such merit, we may question whether it should be considered as a soft goal for the system. On the other hand, the downside of the ConstraintsThroughE-agenda alternative is that it puts extra requirements on the participant's side, namely, to be equipped with an e-agenda and to make it accessible from outside. Such simple analysis may bring the goal MinimalRequirementsOnParticipants to light, which the ConstraintsThroughE-agenda alternative is conflicting with.

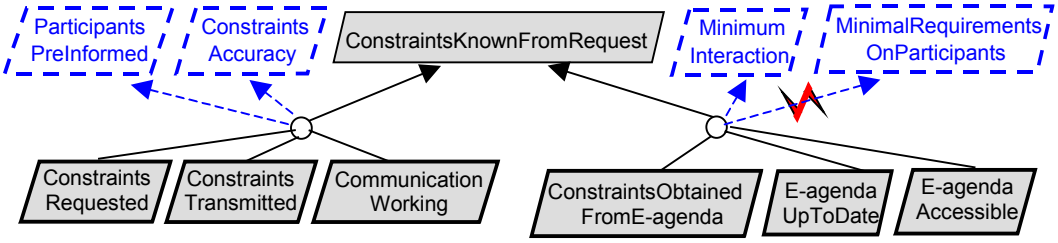


Figure 8.16 – Soft goals identifiable from *pros* and *cons* of alternative options

(H7) Identify agent wishes. Another heuristics for further identification of goals consists of reviewing the list of system agents already identified in the current stage of the elicitation process. For each of them, we may ask stakeholders playing the corresponding role at the process level what goals this agent might wish the new system to satisfy.

In the meeting scheduling system, for example, goals such as MinimumInteraction or MinimalRequirementsOnParticipants are wished by Participant agents and might be identified that way as well. The goal SchedulingLoadReduced might appear as a goal wished by the Initiator agent. In the library system, the goal BorrowedCopyReturnedOnTime might appear as being wished by LibraryStaff agents wheras the conflicting goal CopyBorrowedAsLongAsNeeded might appear as being wished by Patron agents.

(H8) Analyze obstacles, threats, and conflicts. Various forms of analysis of the goal model provide another important source for identifying further goals in a consolidated model. In particular, the identification of obstacles to goal satisfaction raises the exploration of new goals to overcome them. The analysis of threats to security goals results in the introduction of countermeasures as new security goals in the goal model. The detection of conflicts among goals may result in new goals to resolve them. These analyses will be detailed in Chapters 9, 16, and 18.

(H9) Check the converse of Achieve goals. It is often the case that an *Achieve* goal of form:

if preCondition **then sooner-or-later** TargetCondition

has a converse *Maintain* goal associated with it taking the form:

always (**if** Target Condition **then** preCondition).

The latter goal is typically a safety or security goal that may have been overlooked if we were primarily concerned by functional goals so far. If so, we need to integrate this new goal, possibly together with its parents and children, at some appropriate place in the goal model.

For example, let us suppose that we have found the functional *Achieve* goal for the A320 braking logic:

if WheelsPulseOn **then sooner-or-later** ReverseThrustEnabled

Once this goal has been identified we should question whether the converse prescription makes sense, that is,

always (**if** ReverseThrustEnabled **then** WheelsPulseOn).

This converse is indeed an essential safety goal in this case as enabling reverse thrust before the plane is on ground would result in a disaster. As another example, consider the following standard *Achieve* goal in an e-shopping system:

if ItemPaid **then sooner-or-later** ItemSent.

Using our heuristic, we should question whether the converse prescription makes sense, that is,

always (**if** ItemSent **then** ItemPaid),

which indeed is an essential security goal in any e-shopping system.

(H10) Check the complementary case of conditional Achieve goals. Logically speaking, this heuristic is a variant of the previous one. Once we have identified an *Achieve* goal of form:

if preCondition **then sooner-or-later** TargetCondition,

we should ask a *WHAT IF* question about it to check whether a complementary goal should be prescribed, taking the form:

if not preCondition **then** ??

For example, we might have identified the functional *Achieve* goal for our train control system:

if block signal set to 'go' **then sooner-or-later** the approaching train is on this block.

The *WHAT IF* question prompted by this heuristic would then lead us to elicit a companion goal of form:

if block signal set to 'stop' **then** ??

which in this case is an essential prescription to be made precise and integrated in the goal model (see Fig. 8.5).

8.8.3 Delimiting the scope of the goal model

While expanding the goal model top-down and bottom-up along refinement branches, we need to know when the goal refinement/abstraction process should stop. The answer is quite simple; it derives from the definition of the concepts of goal, requirement, and expectation (see Chapter 7).

(H11) Refine goals until they are assignable to single agents. The process of asking *HOW* questions along a refinement path in the goal model terminates when we reach a fine-grained goal assignable as a requirement on some software agent or as an expectation on some environment agent. The corresponding leaf goal will be operationalized into software operations or environment tasks depending on the type of responsible agent (see Chapter 12).

For example, the goal ConstraintRequested in Fig. 8.15 should not be refined futher as it can be assigned to the Scheduler agent. Similarly, the goal DoorsStateClosed**If**NonZeroMeasuredSpeed is a leaf goal in the goal model as it is assignable as a requirement on the TrainController agent.

As we will see it in Chapter 11, agents may have varying granulatities during model elaboration. An agent can be modelled as an aggregation of finer-grained agents. For example, an organizational

department with specific goal responsibilities, e.g., LibraryDepartment, might be decomposed at some stage of the environment modeling process into several operational units with corresponding finer-grained roles and responsibilities, e.g., LibraryStaff, SubscriptionService, BookAcquisitionService, etc. Likewise, a software agent with associated requirements, e.g., ConstraintHandler in the meeting scheduling system, might be decomposed during architectural design into finer-grained components with corresponding requirements, e.g., constraintRequestor and constraintCollector. In such cases, the goals under responsibility of the coarser-grained agent must be refined accordingly, with the finer-grained subgoals being assigned to the finer-grained subagents (see Section 11.4).

(H12) Abstract goals until the system's boundary is reached. The process of asking *WHY* questions along a refinement path in the goal model terminates when we reach a high-level goal whose parent goals cannot be satisfied through cooperation of the system's agents only. Such goals fall ouside the scope of the system.

For example, the goal "*eliminate greenhouse effect*" is not within the scope of the train control system as it requires the cooperation of additional parties that are not part of the system. The subgoal of meeting transport regulations contributing to it might lie in the system's scope though. Likewise, goals such as "*make library users cultivated*" or "*make meeting participants happy*" require the cooperation of additional parties and therefore fall outside the scope of the system. They are not specifically relevant to the problem world of managing a library or scheduling meetings.

8.8.4 Avoiding common pitfalls

There are a number of problems, confusions, and bad smells frequently made by novice modellers that we should be aware of and avoid.

(H13) Do not confuse goals with operations. A goal is conceptually different from a particular action taken to satisfy that goal. For example, the goal of keeping doors closed while a train is moving is different from the action of opening doors under safe conditions to meet that goal.

A goal captures an objective the system should satisfy; it is specified declaratively. An operation captures a functional service the system should provide to satisfy such objective;. it is specified by conditions characterizing its applicability and effect (see Chapter 12).

Semantically speaking, a behavioral goal constrains entire sequences of system state transitions; an operation constrains single state transitions within such sequences. For example, the goal Achieve[CopyBorrowed**If**Available] in Figs. 8.3 and 8.17 constrains sequences of system state transitions so that a requested book copy if available is eventually borrowed within some acceptable amount of time. On the other hand, the operation BorrowCopy constrains transitions from a state where the book copy is not borrowed to a state where it is borrowed; in the input state the book must be requested and the copy must be available.

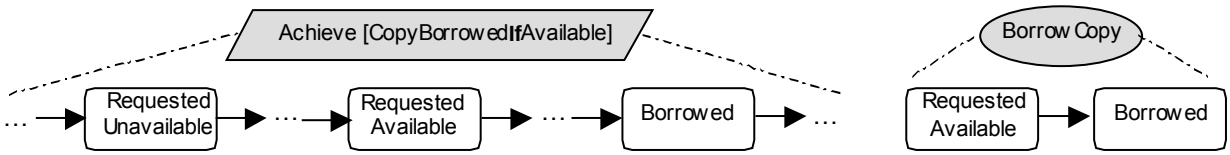


Figure 8.17 – Behavioral goals vs. operations: semantic difference

As we will see it in Chapter 12, operations from the operation model *operationalize* leaf goals from the goal model – somewhat the same way as programs implement their specification. A goal is in general operationalized through multiple operations ensuring it. On the other hand, an operation may operationalize multiple goals underpinning it. For example, the goal *DoorsStateClosed**If***

NonZeroMeasuredSpeed will be operationalized through operations such as OpenDoors, to be applied only when the train is stopped, and StartTrain, to be applied only when the train has its doors closed. On the other hand, the operation OpenDoors might itself operationalize other goals, such as Avoid[PassengersDelayed], through other restrictions – in this case, the additional restriction of being applied as soon as the train is stopped at a platform.

Furthermore, some goals may not have an operation counterpart. This is in particular the case of soft goals used for evaluating alternative options (see Section 16.3).

Terminology tip. To avoid possible confusions between a goal and an operation, use the past participle of some suggestive verb for the goal's name (e.g., "CopyBorrowed"), to refer to the target condition the goal prescribes, and the infinitive tense for the name of a corresponding operation (e.g., "BorrowCopy"), to refer to single transitions to output states satisfying that target condition.

(H14) Do not confuse AND-refinements into multiple cases with OR-refinements into multiple alternatives. Through case analysis, we may refine a goal into multiple subgoals where each subgoal specializes the parent goal to some distinct case. This must be captured by an AND-refinement into such subgoals; the refinement is complete if all possible cases that can occur in the target system are covered. The left diagram in Fig. 8.18 illustrates such AND-refinement into multiple cases –here, the case where a copy of the requested book is available and the case where such copy is not available.

On the other hand, we may refine a goal in different, alternative ways. In such OR-refinement, each alternative entails the parent goal according to some distinct option. Different options result in different system proposals; one single option is taken in the target system. The right diagram in Fig. 8.18 illustrates such OR-refinement into multiple alternatives.

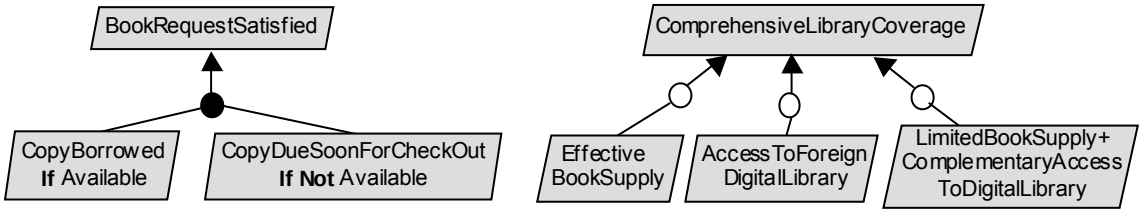


Figure 8.18 – AND-refinement into subgoals vs. OR-refinement into alternatives

Those two mechanisms for goal refinement are different and should not be confused with each other. An AND-refinement into cases introduces complementary subgoals within the same system. An OR-refinement into alternatives introduces subgoals for different systems. In Fig. 18, it would be a modeling error to replace the AND-refinement in the left diagram by alternative refinements like in the right diagram – one for the case *Available*, another for the case **Not**Available. It would make no sense to consider one system proposal where requested books always have a copy available and another alternative system proposal where requested books never have a copy available.

Such error corresponds to the frequent confusion between the "*and*" and "*or*" connectives in case analysis. As pointed out in Section 4.1, the correct decomposition of the statement:

If (Case1 **OR** Case2) **then** <Statement>

is the statement:

If Case1 **then** <Statement> **AND if** Case2 **then** <Statement>.

(H15) Avoid ambiguities in goal specifications. The same goal might be understood in different ways by different people. To avoid this, the goal name should be chosen as a suggestive shortcut for what it prescribes. More importantly, what such name exactly designates must be made fully precise in the

goal specification provided in the goal's ***Def*** annotation (see Section 8.1). This specification must be adequate, complete, formulated precisely in terms of measurable system phenomena, and agreed upon by all parties involved in the RE process.

Consider the goal named *DoorClosedWhileMoving*, for example. What is it really meant to prescribe? Do we mean that train doors should remain closed in any state where the train is moving? Or do we mean "between stations" instead? What if a train is not moving, e.g., between two successive stations? The answer to all such questions should be crystal clear and unambiguous from the ***Def*** annotation. For example, the specification "*train doors shall be closed in any state where the train is moving*" imposes no restriction on states where the train is not moving; doors may be either open or closed then. Another goal should be introduced then to prescribe some appropriate behavior in case of emergency stop between stations – this goal can be derived from further analysis, as Part 3 of the book will show.

8.8.5 Reusing refinement patterns

The reuse of "pre-canned" patterns provides quite effective guidance in the model elaboration process. Such patterns suggest generic refinements/abstractions for instantiation to the specifics of the modelled system. The technique works as follows.

1. Establishing a catalogue of refinement patterns. Common patterns for refining a goal into subgoals are predefined in a pattern catalogue. Each pattern encodes a specific tactic for goal decomposition.

A pattern is characterized by a *name*, an *applicability condition*, a two-level *AND-refinement tree* showing how a generic behavioral goal can be refined into generic subgoals, a number of possible *variants*, and *examples* of use. The goal specifications refer to parameters representing conditions. For example, the refinement pattern shown as left diagram in Fig. 8.19 hereafter has three parameters: TargetCondition, MilestoneCondition, and CurrentCondition.

The completeness of each generic AND-refinement in the catalogue is established once for all – through a formal proof, but the user of the pattern has no need to see any formal specification or proof.

2. Retrieving reusable refinement patterns. At system modeling time, we may browse the catalogue to retrieve applicable patterns.

- A generic parent goal in some pattern might match the goal currently considered. An AND-refinement is then obtained by instantiating the parameters in the generic subgoals to corresponding system-specific conditions, including the parameter instantiations from the matching parent goal. For example, the AND-refinement shown as right diagram in Fig. 8.19 is obtained by instantiating the pattern on the left; the parameters TargetCondition, MilestoneCondition, and CurrentCondition are replaced by ConvenientMeetingScheduled, ConstraintsKnown, and Request, respectively.
- Conversely, a set of generic subgoals in some pattern might match the goals currently considered. The root of a corresponding AND-refinement is then obtained by instantiating the parameters in the generic parent goal to the system-specific parameter instantiations in the matching subgoals. The pattern is thereby used as an *abstraction pattern*. For example, the parent goal Achieve[ConvenientMeetingScheduledFromRequest] in the right diagram in Fig. 8.19 might have been obtained by such reverse application of the pattern in the left diagram.
- More frequently, a generic parent goal together with some subgoals in a pattern might match the partial AND-refinement tree we are currently building. The refinement is then *completed* by adding the pattern subgoals that were missing in our current AND-refinement, and instantiating their parameters according to the match.

- Several candidate patterns might match a current modeling situation. This produces *alternative refinements* to be considered for inclusion in the goal model and for evaluation against soft goals from the model.

3. Adapting the reused patterns. The produced refinement, abstraction, or completion may need to be adapted for adequacy with the specifics of the modeled system. To document the refinement and help model users understand it, the resulting refinement/abstraction may be annotated with the name of the pattern that was used (see the "**Tactic**" annotation in Section 8.7).

A catalogue of common goal refinement patterns

The following patterns encode frequently used refinement tactics. As mentioned before, each pattern is described by a name, an applicability condition, a generic AND-refinement tree, variants, and examples of use from our running case studies.

The milestone-driven refinement pattern. This pattern is *applicable* to behavioral *Achieve* goals where an intermediate condition can be identified as a necessary milestone for reaching the target condition prescribed by the goal. More precisely, any behavior prescribed by the goal must necessarily reach a state satisfying the milestone condition before reaching a state satisfying the target condition.

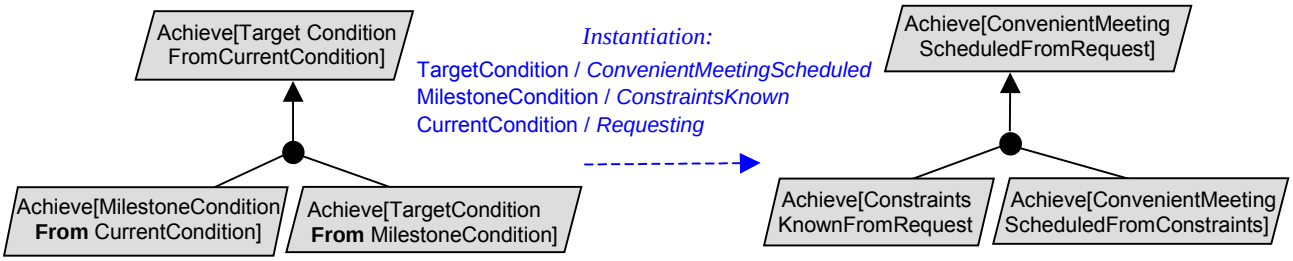


Figure 8.19 – The milestone-driven refinement pattern

Refinement tree. The left diagram in Fig. 8.19 shows the generic AND-refinement tree for this pattern. Remember that the specification of *Achieve* goals has the following form (see Section 7.3.1):

Achieve [TargetCondition]: **[if** CurrentCondition **then] sooner-or-later** TargetCondition ,

The first subgoal in the left diagram in Fig. 8.19 is an *Achieve* goal with the milestone condition as target condition; it states that sooner or later the milestone condition must hold if the condition *CurrentCondition* from the parent goal holds in the current state. The second subgoal is an *Achieve* goal as well; it states that sooner or later the target condition of the parent goal must hold if the milestone condition holds in the current state. The completeness of the AND-refinement can be established from the definition of what a milestone condition is about.

Example of use. The right diagram in Fig. 8.19 shows an application of the milestone-driven refinement pattern for building the model fragment in Fig. 8.11. The milestone condition expresses that the constraints of invited participants are known to the scheduler. This condition is indeed a prerequisite for determining a convenient date and location for the meeting. The first *Achieve* subgoal in the instantiated refinement states that those constraints should be known to the scheduler within some reasonable time; the second subgoal states that a convenient meeting date and location should be determined within some reasonable time once all participant constraints are known.

Variants. A first variant consists of considering several successive milestones to be reached on any path to a target state. For *n* milestones we obtain *n+1* subgoals *Achieve[NextMilestoneConditionFromPreviousMilestoneCondition]*, where NextMilestoneCondition is

TargetCondition for the last subgoal, and PreviousMilestoneCondition is CurrentCondition for the first subgoal ($n \geq 1$).

Another variant consists of taking a *Maintain* goal as parent goal in the pattern. The GoodCondition to be maintained acts as TargetCondition. The *Achieve* subgoals, if any, must guarantee that the milestone conditions are established sufficiently fast for the GoodCondition to be preserved. Fig. 8.20 shows the use of both variants in our train control system. The past participles in subgoal names are written there in bold to suggest corresponding milestone conditions.

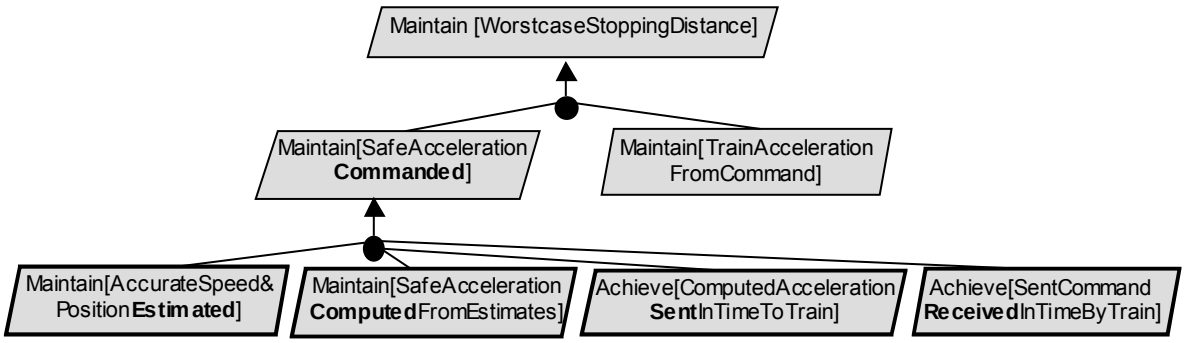


Figure 8.20 – Variants of the base pattern for milestone-driven refinement

Milestone-driven refinements and scenarios. A milestone-driven goal refinement can often be mapped to a scenario illustrating it. Conversely, an elicited scenario can often be mapped to a milestone-driven goal refinement generalizing it. Each milestone condition then corresponds to a state assertion holding after some interaction along the scenario timeline. For example, the meeting scheduling scenario in Fig. 8.14 corresponds to a milestone-driven refinement of the goal Achieve[MaximumAttendance] into subgoals Achieve[ConstraintsKnownFromRequest], Achieve[ConvenientMeetingScheduledFromConstraints], and. Achieve[ParticipantsInformedOfScheduledmeeting] (see Fig. 8.11).

Case-driven refinement patterns. These patterns introduce case conditions that guide the refinement of the parent goal. Two frequently used such patterns are the decomposition-by-cases pattern and the guard-introduction pattern.

The decomposition-by-case pattern. This pattern is *applicable* to behavioral *Achieve* goals where different cases can be identified for reaching the target condition. The cases must be disjoint and cover the entire state space.

Refinement tree. Fig. 8.21 shows the generic AND-refinement tree for this pattern. In each case a specific target condition must be reached. The refinement requires two domain properties, one stating the disjointness and coverage property of the case conditions, the other stating that the disjunction of the specific target conditions must imply the target condition of the parent goal. The completeness of the AND-refinement is derivable by use of those two domain properties.

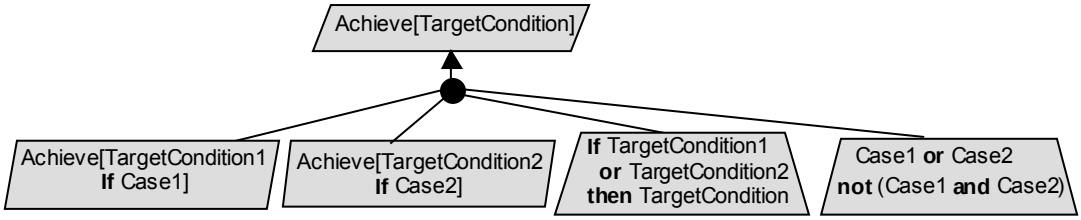


Figure 8.21 – Decomposition-by-cases pattern

Examples of use. The decomposition-by-case pattern was applied in Fig. 8.2 for refining the goal Achieve[BookRequestSatisfied] into subgoals Achieve[CopyBorrowed**If**Available] and Achieve[CopyCopyDueSoonForCheckOut**If****Not**Available]. The domain properties were left implicit there; the first states that if a copy of the requested book is borrowed or due soon for check out by the requesting patron, then the book request is satisfied. The second domain property tautologically states that a book either has an available copy or it has not. Note that the decomposition-by-case pattern could be used for refining the same goal according to different partitions into cases. For example, the goal Achieve[BookRequestSatisfied] could also be refined into subgoals Achieve[BookRequest**ByStaff**Satisfied] and Achieve[BookRequest**ByOrdinaryPatron**Satisfied].

Variants. The pattern in Fig. 8.21 can be trivially generalized to *n* disjoint cases covering the entire state space, with specific target conditions associated with each of them ($n \geq 2$). It can also be particularized to the situation where the target conditions in the various cases all amount to the target condition of the parent goal. Fig. 8.22 illustrates the use of the latter variant for refining the goal ConvenientMeetingScheduledFromConstraints that appeared in Fig. 8.11.

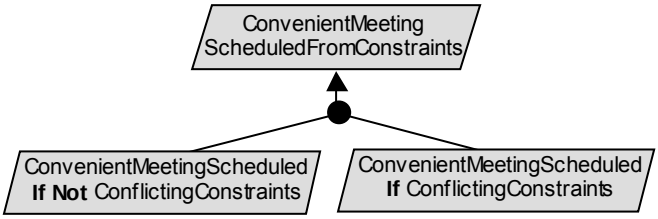


Figure 8.22 – Instantiating the decomposition-by-cases pattern to meeting scheduling

Another variant concerns the decomposition of *Maintain/Avoid* goals by cases. The corresponding pattern is defined by replacing "*Achieve*" by "*Maintain*" and "*TargetCondition*" by "*GoodCondition*" in Fig. 8.21.

The guard-introduction pattern. This pattern is a case-driven refinement pattern *applicable* to behavioral *Achieve* goals where a guard condition must necessarily be set for reaching the target condition.

Refinement tree. Fig. 8.23 shows the generic AND-refinement tree for this pattern. The first subgoal of the *Achieve* goal states that the target condition must be reached from a current condition where in addition the guard condition must hold. The second subgoal states that the guard condition must be reached as a target from that current condition. The third subgoal states that the current condition must always remain true unless the target condition of the parent goal is reached.

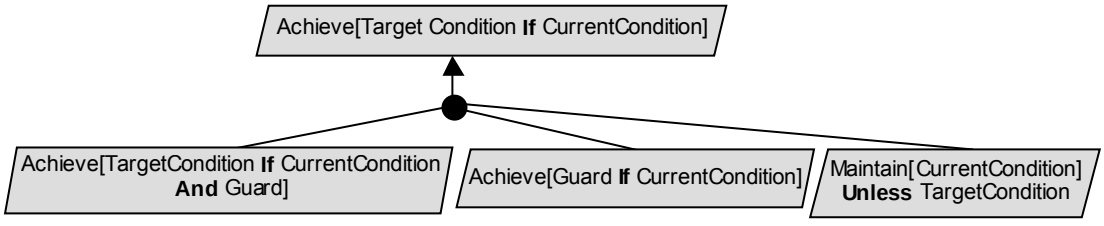


Figure 8.23 – Guard-introduction pattern

Example of use. The guard-introduction pattern was applied for refining the train system goal Achieve[FastJourney] in Fig. 8.12. In this example, the guard for reaching the next block is the condition GoSignal. The subgoals obtained by pattern instantiation are

Achieve[FastRunToNextBlock**If**GoSignal], Achieve[SignalPromptlySetToGo], and Maintain[TrainWaiting] **Unless** OnNextBlock.

Matching this pattern with the refinement of the same parent goal in Fig. 8.7, we conclude that the refinement there is incomplete due to the missing subgoal Maintain[TrainWaiting] **Unless** OnNextBlock. The completeness of the guard-introduction pattern can be established formally in temporal logic, and such completeness check can be automated, as we will see it Chapter 18.

The divide-and-conquer pattern. This pattern is *applicable* to *Maintain* goals where the GoodCondition to be preserved is composed of two conjoined conditions.

Refinement tree. Fig. 8.24 shows the generic AND-refinement tree for this pattern. The parent goal states that the conjoined good conditions must always be preserved unless some EndCondition becomes true. The refinement consists of splitting the parent goal according to the splitting of its good conditions.

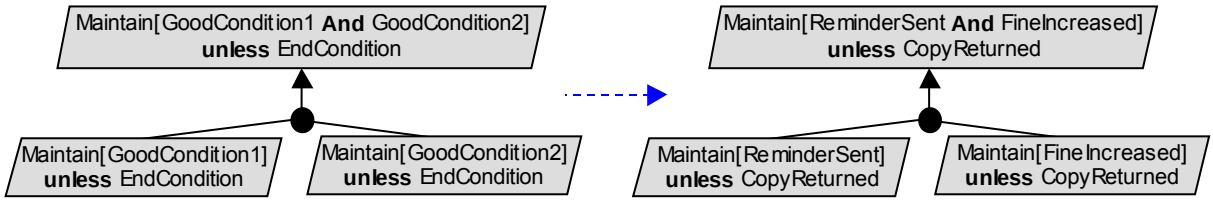


Figure 8.24 – Divide-and-conquer pattern

Example of use. Fig. 8.24 also shows an application in the goal model for the library system. The goal of periodically sending reminders while increasing fines, unless the corresponding book copy is returned, is thereby split in two subgoals.

A systematic use of the divide-and-conquer pattern improves the cohesion of goal models as finer-grained goal specifications refer to single concerns only.

Variants. The GoodCondition to be preserved may be composed of n conjoined conditions ($n \geq 2$). Another variant is the particular case where the EndCondition reduces to **true**, which results in dropping the "**unless**" part in the goal/subgoal specifications.

Unrealizability-driven refinement patterns. These patterns support the heuristic of splitting responsibilities among agents (see *(H5)* above). They are applicable to *Achieve* or *Maintain/Avoid* goals refering to conditions that cannot be monitored or controlled by the agents expected to take responsibility over them.

The general idea is to introduce monitorable or controllable counterparts of such conditions together with additional assertions mapping the original conditions to their monitorable/controllable counterpart. These assertions are accuracy goals or domain properties dependent on whether they are prescriptive or descriptive. (The notion of monitorable/controllable phenomena was introduced in Section 1.1.4. Accuracy goals were defined Section 7.3.2. The notion of goal realizability by an agent will be detailed in Chapter 11.)

The unmonitorability-driven refinement pattern. This pattern is *applicable* when some condition in a goal formulation, to be monitored by the agent expected to be in charge of the goal, cannot be monitored by the agent. The pattern is aimed at resolving such unmonitorability.

Refinement tree. The left diagram in Fig. 8.25 shows the AND-refinement tree for this pattern.

Examples of use. In the right diagram in Fig. 8.25, the agent expected to be responsible for the goal Maintain[DoorsClosedWhileNonZeroSpeed] is the software train controller. This agent cannot monitor the

condition NonZeroSpeed though; there is no way the software controller could evaluate the train's physical speed. The pattern is therefore instantiated by introduction of a condition NonZero*Measured*Speed, monitorable by the train controller, and a corresponding accuracy goal to be assigned as expectation to a speed sensor agent. Fig. 8.6 provides another example where the unmonitorability-driven refinement pattern was used twice. The first application to the top goal resulted in the introduction of a domain property rather than a goal (like the first application to the top goal in Fig. 8.4).

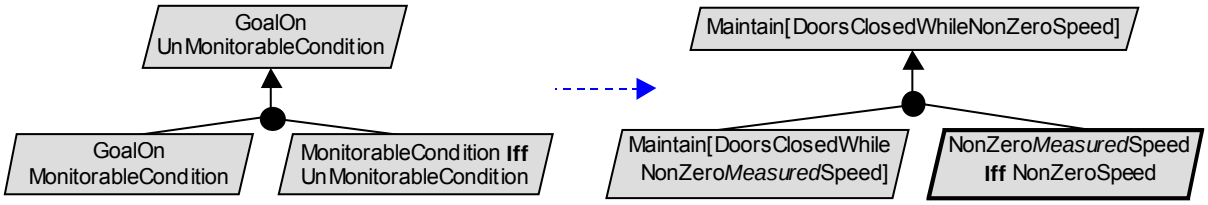


Figure 8.25 – Unmonitorability-driven refinement pattern

The uncontrollability-driven refinement pattern. This pattern is *applicable* when some condition in a goal formulation, to be controlled by the agent expected to be in charge of the goal, cannot be controlled by the agent. The pattern is aimed at resolving such uncontrollability.

Refinement tree. The left diagram in Fig. 8.26 shows the AND-refinement tree for this pattern.

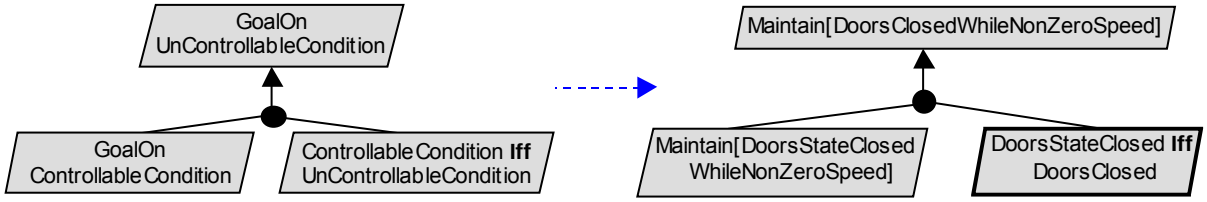


Figure 8.26 – Uncontrollability-driven refinement pattern

Example of use. In the right diagram in Fig. 8.26, the train controller cannot control the condition DoorsClosed as it cannot physically close train doors. The pattern is therefore instantiated by introduction of a condition DoorsStateClosed, controllable by the train controller, and a corresponding accuracy goal to be assigned as expectation to the doors actuator agent.

8.8.6 Reusing refinement trees associated with goal categories

The preceding refinement patterns encode general tactics commonly used for problem decomposition. They make no assumption on the goal category or on the domain in which they will be reused. (Goal categories were discussed in Section 7.3.2.)

As a complement, we may encode typical ways of refining goals within a specific goal category. Such encoding yields larger refinement trees on category-specific concepts and conditions.

Model reuse may help us in the goal elicitation process, as discussed in Section 2.2.7. When a goal is introduced in the goal model, instead of starting a refinement from scratch, we may check refinement trees in the corresponding category, instantiate reusable ones in system-specific terms, adapt them as necessary, or simply reject them if they are felt inadequate. We may also compare them with the refinements we are elaborating to check for missing subgoals or alternative refinements.

Fig. 8.27 shows a reusable refinement tree in the category of *satisfaction* goals (see Fig. 7.5). The goal RequestIssued is under the responsibility of the agent requesting the service. The service request is

satisfied if there is a response to the request by some appointed server that meets the request. The server must remain appointed until the service is fully delivered.

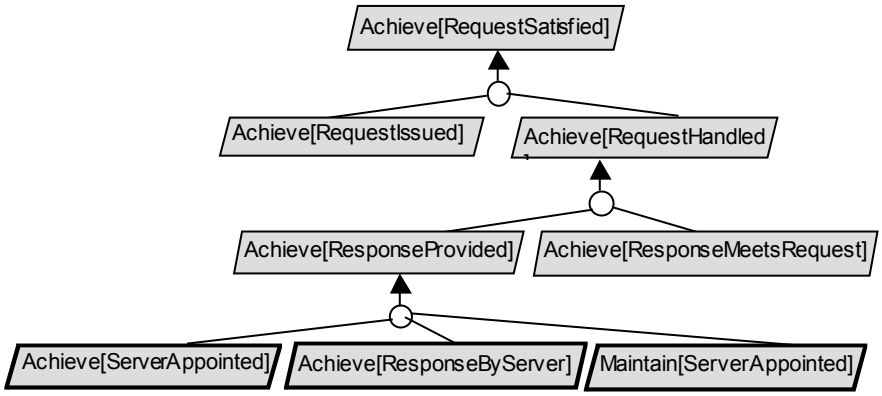


Figure 8.27 – A generic refinement tree for satisfaction goals

The comparison between the reusable refinement tree in Fig. 8.27 and the partial goal model for the meeting scheduling system in Fig. 8.11 might raise questions for further goal elicitation, such as the absence of goals related to the issuing of requests by meeting initiators or the absence of goals related to scheduler appointment – e.g., do we want multiple schedulers specialized to different types of meetings or distributed among multiple sites?

Fig. 8.28 shows a reusable refinement tree in the category of *safety* or *security* goals. Forbidden states are avoided there by anticipating dangerous states that might lead to them. Such dangerous states raise alarms or warnings to which responses must be provided by some appointed guard so as to clear the potentially dangerous situation. The goal ResponseByGuard is not necessarily a leaf goal as it may require multiple agents to cooperate. For instance, a response can itself be a stimulus sent to other agents; stimuli can be chained, forwarded, broadcasted, acknowledged, and so on.

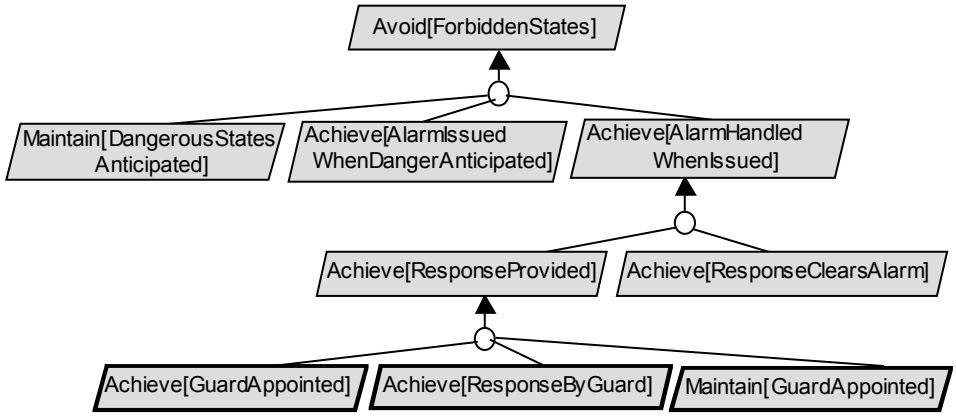


Figure 8.28 – A generic refinement tree for safety or security goals

Summary

- A model-based approach to RE provides focus on and structure for what needs to be elicited, evaluated, specified, consolidated, explained, and modified during the RE process. A model

abstracts from multiple details to capture the essence of the system *as-is* and *to-be*. A good model should be adequate, complete, pertinent, traceable, and comprehensible. It should support multiple levels of precision and highlight alternative options. To be complete, a model should cover the multiple facets of the system. These include the intentional facet (goals), the structural facet (objects), the responsibility facet (agents), the functional facet (operations), and the behavioral facet (scenarios and state machines).

- A goal model basically shows how the system's functional and non-functional goals contribute to each other through refinement links down to software requirements and environment assumptions. It may also capture potential conflicts among goals and alternative ways of refining goals. At its interface with other views of the system, the goal model captures responsibility links between goals and system agents, obstruction links between goals and obstacles, reference links from goals to conceptual objects, operationalization links between goals and system operations, and coverage links between goals and scenarios or state machines.
- A goal model is graphically represented by a goal diagram. Its core is an AND/OR graph showing AND-refinements of goals into conjoined subgoals and OR-refinements of goals into alternative AND-refinements. A goal AND-refinement means that the goal can be satisfied by satisfying all its subgoals. A goal diagram can be built and browsed top-down, bottom-up, or, most frequently, in both directions. Leaf goals correspond to requirements on the software-to-be or expectations on environment agents.
- Every goal in a goal diagram must be individually characterized through annotations. These include a precise goal specification and optional features such as the goal's type, category, priority, elicitation source, or issue still to be addressed. Soft goals may be annotated with a fit criterion. Behavioral goals may optionally be annotated with a formal specification for more sophisticated forms of analysis.
- Goal refinements should be consistent with known domain properties and hypotheses. They should be complete, to avoid missing requirements, and minimal, to avoid unnecessary requirements. Those properties should be established, at least informally, through chains of satisfaction arguments along refinement paths in the goal model. Domain properties and hypotheses are often involved in such arguments. They must be checked for adequacy with respect to real-world phenomena.
- In addtion to alternative goal refinements, a goal diagram may capture alternative responsibility assignments. Both kinds of alternative options result in different system proposals, distributions of responsibilities, and boundaries between the software-to-be and its environment. These define system versions. Multiple options can be annotated by the system version they refer to.
- To start building a goal model, we may obtain preliminary goals by analyzing the strategic business objectives of the system-as-is, by identifying the domain-specific objectives to be preserved across system versions, and by addressing the reported problems and complaints about the system-as-is in the light of new opportunities. We may also search for prescriptive, intentional, or amelioration keywords in interview transcripts and other preliminary documents involved in the elicitation process. In addition, we may browse the various goal categories to look for system-specific instantiations in each category.
- Once preliminary goals are obtained, we may build refinement and abstraction paths in a goal diagram by recursively asking HOW and WHY questions about available goals, respectively. HOW ELSE questions may help identifying alternative refinements. A simple qualitative analysis of the pros and cons of alternatives may help identify corresponding soft goals that might be

missing. *HOW* questions may be constrained by requiring the resulting subgoals to involve fewer potential agents. Goals are to be refined until they are assignable to single agents. They should be abstracted until high-level goals are reached whose parent goals cannot be satisfied through cooperation of the system's agents only.

- Goals should not be confused with functional services that operationalize them. AND-refinements into multiple cases are different from OR-refinements into multiple alternatives. Annotating each goal in a model with an adequate, complete, precise, and agreed definition of the goal in terms of measurable system phenomena is essential for avoiding ambiguities and misunderstandings.
- Refinement patterns provide effective guidance in the model elaboration process by encoding common tactics for goal decomposition. By instantiating a pattern in matching situations, we may produce a complete refinement, find some underlying parent goal, or complete a partial refinement. Pattern instantiations must be checked for adequacy and adapted if necessary. Multiple matching patterns produce alternative refinements. The most commonly used patterns include the milestone-driven, decomposition-by-cases, guard-introduction, divide-and-conquer, unmonitorability-driven, and uncontrollability-driven refinement patterns.

Notes and Further Reading

The integration of multiple views for model completeness is discussed in (Nuseibeh et al, 1994). Earlier modeling frameworks were built on this principle. For requirements modeling, SADT and RML combine structural and functional views (Ross & Schoman, 1977; Greenspan et al, 1986); SA combines functional and contextual views (DeMarco, 1978); KAOS adds the intentional dimension to such views (Dardenne et al, 1993). For modeling software designs, OMT and UML integrate structural, functional, and behavioral views (Rumbaugh et al, 1991; Rumbaugh et al, 1999; Jacobson et al, 1999). A comparative evaluation of modeling frameworks can be found in (Mylopoulos, 1998).

Goal AND/OR graphs for RE were introduced in (Dardenne et al, 1991; Mylopoulos et al, 1992). Such graphs are commonly used for problem reduction in artificial intelligence (Nilsson, 1971). The NFR modeling framework is more focussed on capturing positive/negative contributions among soft goals (Mylopoulos et al, 1992). The KAOS framework was originally more oriented towards behavioral goals and their refinement/conflict links (Dardenne et al, 1991; Dardenne et al, 1993). It also covered alternative responsibility assignments and operationalization links. The latter were inspired from the concept of operationalization in (Mostow, 1983). The completeness, minimality, and domain consistency of refinements is discussed in (Darimont & van Lamsweerde, 1996).

Many modeling frameworks integrate goals, scenarios, and goal-scenario coverage links (Anton et al, 1994; van Lamsweerde et al, 1995; Leite et al, 1997; Anton et al, 1998; Haumer et al, 1998; Rolland et al, 1998; Sutcliffe, 1998; Kaindl, 2000). Links between goals and failure scenarios were introduced in (Fickas & Helm, 1992).

The characterization of a goal by features such as its priority or feasibility was suggested first in (Robinson, 1989). Fit criteria as soft goal features are borrowed from (Robertson & Robertson, 1999). More sophisticated frameworks for capturing issues in models can be found in (Potts & Bruns, 1988; Jintae Lee, 1991). Formal specifications are introduced as goal features in (Dardenne et al, 1993).

Heuristics for goal identification are discussed in (Dardenne et al, 1993), including the elicitation of agent wishes and the splitting of responsibilities among agents. *HOW* and *WHY* questions for identifying goals along refinement links were introduced in (van Lamsweerde et al, 1995). They are used in (Anton & Potts, 1998) as well. The effectiveness of keyword-based identification of goals in documents is discussed in (Anton & Potts, 1998; van Lamsweerde, 2000c; Anton et al, 2001).

The problem of scoping goals within the system's subject matter is addressed in (Zave & Jackson, 1997). This paper also emphasizes the need for precise definition of modelled concepts for unambiguous interpretation in terms of domain phenomena.

Refinement patterns can be seen as a RE counterpart of design patterns (Gamma et al, 1995; Buschmann et al, 1996). They were introduced in (Darimont & van Lamsweerde, 1996). A more comprehensive sample of patterns can be found in Darimont's thesis together with larger refinement trees for certain goal categories and domains (Darimont, 1995). The milestone-driven pattern is inspired from a widely used heuristics in AI problem reduction (Nilson, 1971). The reverse use of refinement patterns for abstraction from scenarios is discussed in (van Lamsweerde & Willemet, 1998). The unmonitorability-driven and uncontrollability-driven patterns were introduced in (Letier & van Lamsweerde, 2002a). They are discussed in greater detail in Letier's thesis (Letier, 2001).

Numerous efforts were made to encode domain expertise in reusable models (Reubenstein & Waters, 1991; Sutcliffe & Maiden, 1998). These include structural models (Ryan & Mathews, 1993; Fowler, 1997b; Konrad et al, 2004), behavioral models (Konrad et al, 2004), and contextual models showing agent interfaces (Jackson, 2001). Patterns also emerged in workflow modeling (van der Aalst et al, 2003).

Experience with goal models in industrial projects is reported in (van Lamsweerde, 2004b; Fairbanks et al, 2006; Darimont & Lemoine, 2007).

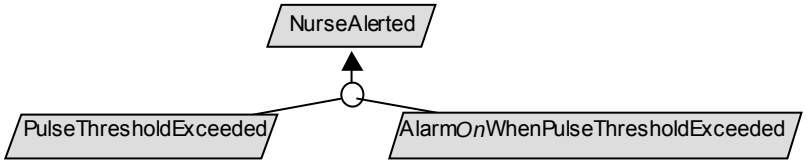
Exercises

- Consider the following integrity goal in the library system:

  Book copy return shall be encoded correctly and by library staff only.

  Elicit parent goals of this goal through WHY questions.

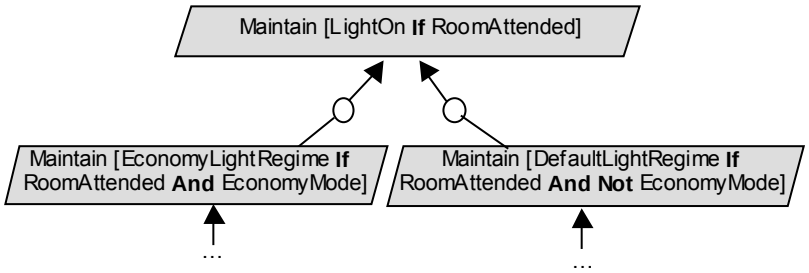
- In a simple patient monitoring system, the goal NurseIntervention**If**PulseThresholdExceeded states that a nurse shall promptly assist a patient whose pulse rate is beyond some critical threshold.
  - Consider the following subtree found somewhere in the refinement of this goal by a novice modeller (goal names there are sufficiently suggestive for what this modeller had in mind):



    In view of what goals and goal refinement refer to, explain why this proposal makes no sense.

  - A MonitoringSoftware agent is expected to help in the satisfaction of the goal NurseIntervention**If**PulseThresholdExceeded. Since this agent cannot monitor the condition PulseThresholdExceeded nor control the condition NurseIntervention, apply the monitorability-driven and controllability-driven refinement patterns to split responsibilities and derive requirements on the software and expectations on environment agents. Your goal diagram should show individual responsibility assignments.

- Consider the following portion of a goal diagram for a light contol system proposed by a novice modeller:



Explain what is wrong with this diagram and fix the error.

- Consider an ambulance dispatching system with the following goal specifications:

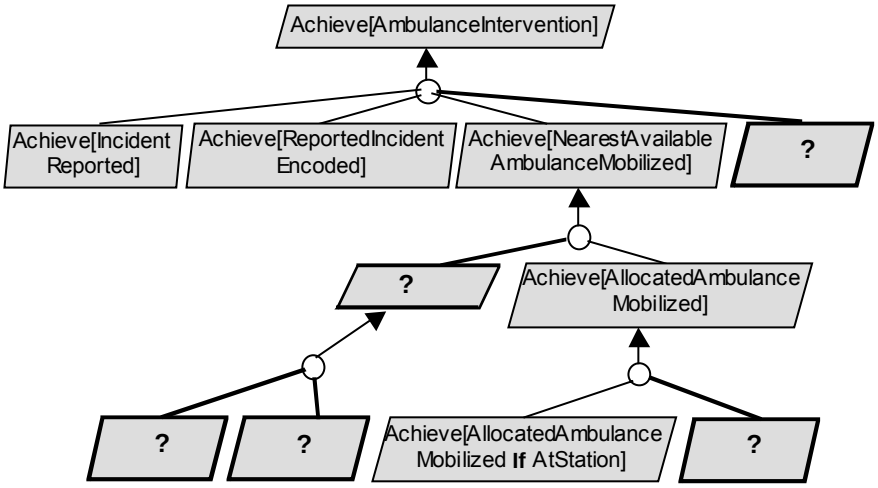
  Goal AmbulanceIntervention

  Def: *For every urgent call reporting an incident, a first ambulance shall arrive at the incident scene within 11 minutes.*

  Goal NearestAvailableAmbulanceMobilized

  Def: *For every encoded incident, the nearest available ambulance shall be mobilized within 3 minutes.*

  - Complete the following goal diagram based on those specifications and using applicable refinement patterns. For each "question mark" goal, provide a suggestive name and a precise specification.
  - In your completed diagram, list the goals needing further refinement and explain why.
  - Identify possible responsibility assignments in your diagram.



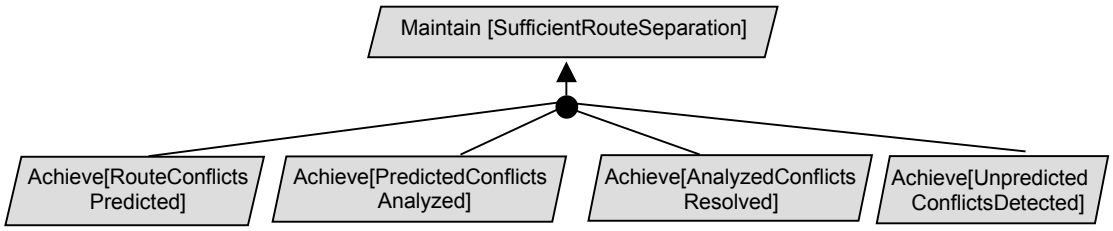
- Consider the leaf goals in the refinement tree in Fig. 8.20. Can these be assigned to agents or do they need further refinement? In the former case, suggest possible responsibility assignments.
- Investigate alternative refinements and responsibility assignments for the goal Avoid[BookCopiesStolen] in the library system. Identify soft goals from the pros and cons of each alternative assignment.
- Consider the following goal model fragment for an air traffic control system. The parent goal roughly states that aircraft routes should be sufficiently separated. The refinement is intended to

produce subgoals for continually separating routes based on different kinds of conflicts among routes.

(a) Identify the parent goal of the goal Maintain [SufficientRouteSeparation].

(b) Restructure this model fragment by use of the decomposition-by-case and milestone-driven patterns so as to make the use of those two patterns explicit and separated.

(c) Doing so, find an incompleteness in this model fragment and fix the anomaly.



- Extend the diagram in Fig. 8.9 so as to capture a meeting scheduling system version where participant constraints are obtained, for important participants, from their e-agenda, and for ordinary participants, through e-mail communication.
- Consider the case study description for the library management system in Section 1.1.2.
  - Based on the list of stakeholder complaints about the library system-*as-is*, reported in this case study description, apply the heuristics in Section 8.8.1 to identify some preliminary goals for a goal model of the library system-*to-be*.
  - Apply the keyword-based search heuristics to the case study description in order to complete this preliminary list.
  - From this list and the full case study description, use *WHY*/*HOW* questions, refinement patterns, and other heuristics in Section 8.8 to build a goal model for the entire system-to-be. You may want to integrate some of the model fragments shown in this chapter.
- Consider the case study description for the meeting scheduling system. in Section 1.1.2.
  - Based on the reported list of usual problems and difficulties in scheduling meetings, apply the heuristics in Section 8.8.1 to identify some preliminary goals for a goal model of the system-to-be.
  - Apply the keyword-based search heuristics to the case study description to complete this preliminary list.
  - From this list and the full case study description, use *WHY*/*HOW* questions, refinement patterns, and other heuristics from Section 8.8 to build a goal model for the entire system-to-be. You may want to integrate some of the model fragments shown in this chapter. The model should explicitly capture the *system variants* suggested in Section 1.1.2.