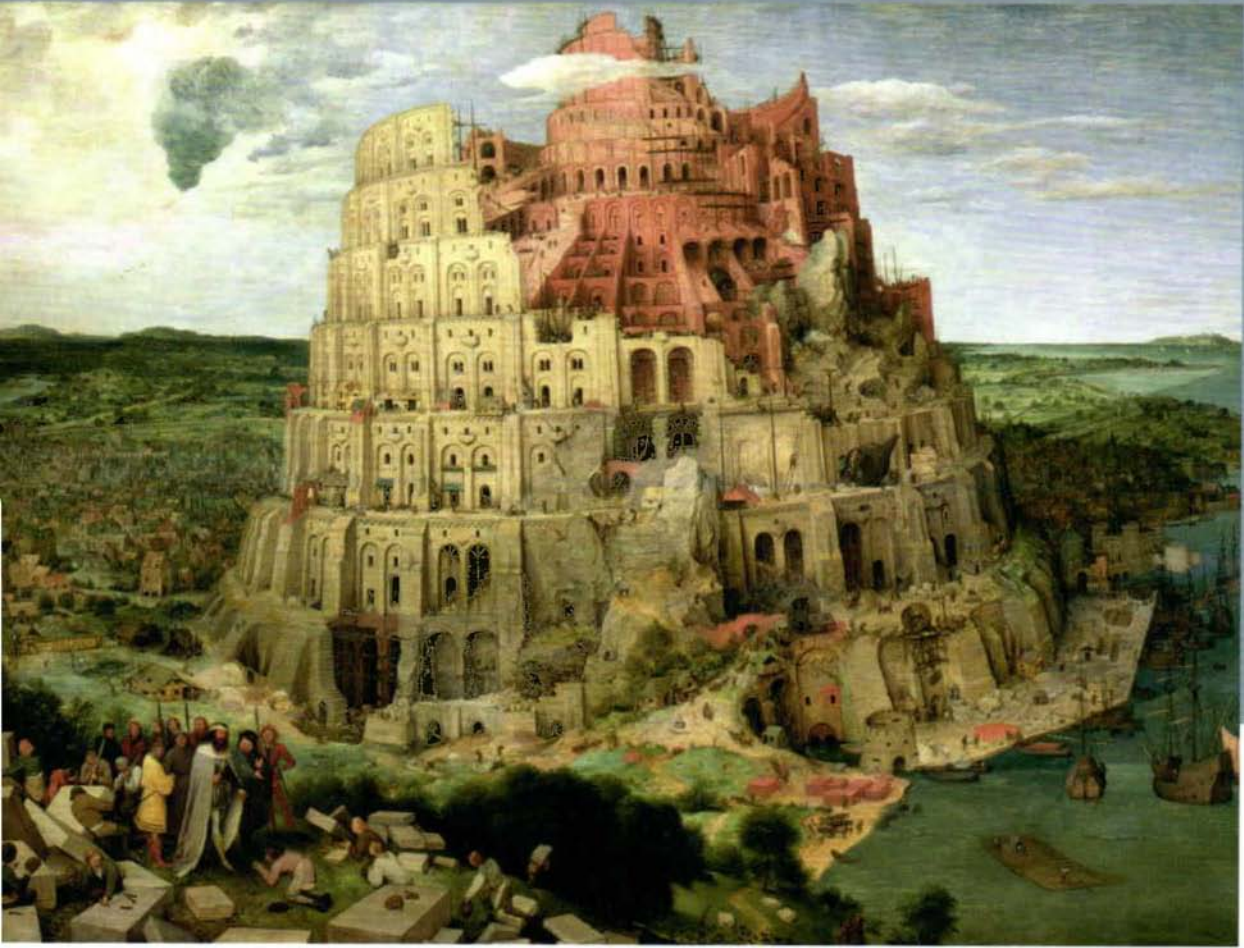


Axel van Lamsweerde



Requirements Engineering

From System Goals to UML Models to Software Specifications

7

Goal Orientation in Requirements Engineering

In our study of the various phases of the RE process, we have regularly found a common abstraction in the preceding chapters without really focusing on it. Chapter 1 introduced a variety of questions that the RE process must address, including *WHY*-questions about the objectives that the system-to-be should meet. As explained in Chapter 2, we need to understand the objectives of the system-as-is during domain analysis, and we need to discover the right reasons for a new system version during requirements elicitation. Chapter 3 discussed the early evaluation of elicited statements for potential conflicts, exposure to risks, selection of best alternatives and prioritization; the scope of this early evaluation included the objectives underlying requirements and environmental assumptions. The structured documentation templates in Chapter 4 allowed for specifying the rationale for specific requirements, unlike the semi-formal and formal specification languages reviewed there. Chapter 5 suggested the need for extending quality assurance techniques to enable earlier analysis, such as verification of the satisfaction of system objectives by operational requirements and assumptions. For evolution support, Chapter 6 confirmed the importance of satisfaction arguments and backward traceability to rationale.

This chapter focuses on system objectives as a core abstraction for requirements engineering. It introduces the concept of a goal to capture this abstraction and goal orientation as a basic paradigm for RE.

Section 7.1 defines this concept more precisely in relation to active system components called agents, and illustrates it through examples and counterexamples from our running case studies. Section 7.2 discusses the varying granularity of goals and clarifies their relationship to requirements and assumptions. Section 7.3 reviews the various types and categories of goals that we may need to elicit, evaluate, specify and consolidate while engineering requirements. Section 7.4 explains the key importance of goals by reviewing their multiple uses in the RE process. Section 7.5 discusses where and when goals can be found in this process. Section 7.6 finally relates goals to other requirements-related artefacts – such

as scenarios of interaction between the software and the environment, operational specifications, use cases and properties used for finite state verification. This section also compares goal orientation with other paradigms such as agent orientation, object orientation and top-down system analysis. The bibliographical notes at the end of the chapter provide further information and credits on this increasingly important area of RE research and practice.

This brief chapter acts as a transition between the state-of-the-art part of the book and the model-building and analysis parts. While relying on various techniques introduced in Part I, we will use goals as a driving force for building and analysing system models in Parts II-III. Thanks to goal orientation, we will be able to reason about system models in terms of the *WHY* and *WHO* dimensions of RE.

7.1 What are goals?

Goal-oriented RE refers to the use of goals for requirements elicitation, evaluation, negotiation, elaboration, structuring, documentation, analysis and evolution.

A *goal* is a prescriptive statement of intent that the system should satisfy through the cooperation of its agents. An *agent* is an active system component playing a specific role in goal satisfaction. To fulfil such a role, the agent needs to restrict its behaviour by adequate control of system items.

In this definition, the word 'system' may refer to the system-as-is or the system-to-be. Goals are prescriptive statements like requirements and unlike domain properties, as the latter are descriptive (see Section 1.1.4). As implied by the wording 'prescriptive statement', goals are declarative, unlike operational procedures for achieving them. A key point in the definition is that *goal satisfaction requires the cooperation of system agents*. As a consequence, the goal must be formulated in terms of phenomena shared among agents; such phenomena are controlled by some agents and monitored by others (see Section 1.1.4). The system agents define the *scope* of the system. Goal satisfaction may involve a variety of system agents:

- Human agents playing specific roles, such as organizational units, operators or end-users.
- Devices such as sensors, actuators, measurement instruments or communication media.
- Existing software components such as legacy, off-the-shelf or foreign components in an open system.
- New software components forming the software-to-be.

Examples Consider the following statement in our meeting scheduling system:

Meetings shall be scheduled so as to maximize the attendance of invited participants.

This statement is clearly prescriptive – unlike the property 'a participant cannot attend multiple meetings at the same time'. It states an intent to be satisfied by agents of the system. No matter who these exactly are – although we expect them to include the meeting scheduler software,

participants (who should attend) and initiators (who need to initiate meetings). The statement is thus a goal. Consider now the statement:

Borrowed copies of books shall be returned within the prescribed deadline.

This statement is a goal that at first sight might involve borrowers (to return copies), the library software (to track loans and issue reminders) and perhaps library staff, depending on which option is taken to meet the goal (we don't really care at this point). Consider now the following statement about our train control system:

The system shall guarantee safe transportation of all passengers.

We recognize a goal again that might involve various agents for its satisfaction, such as on-board train controllers (for control of safe acceleration), the station software (for issuing safe acceleration commands), the network communication infrastructure (for transmitting commands to train controllers rapidly), the tracking system (for keeping track of every train accurately), probably door actuators (for not opening doors if not commanded by the train controller), maybe passengers (for not forcing door opening during train moves) and so on. Again, no matter here who these agents exactly are; the point is that more or less complicated cooperation schemes are required to enforce this safety objective.

Counter-examples Consider the following statement for the meeting scheduler:

The system shall guarantee safe transportation of all participants to the meeting venue.

This statement looks like a prescriptive statement of intent. It cannot be considered as a goal of our meeting scheduling system, though; no cooperation among scheduler, participant and initiator agents can enforce it. Such an objective is out of the system's scope. Similarly, the prescriptive statement 'Make library users happy' is out of reach of the potential agents expected to form the library system-to-be. Consider now the following statement:

To initiate a meeting, the initiator needs to prompt the meeting scheduler, authenticate him- or herself, fill in the ad hoc form to provide the date range and information about invited participants, and then confirm the request.

This statement is not a prescriptive statement of intent; it is not a declarative statement. It rather captures an operational procedure for achieving implicit goals such as 'Initiated meetings must be given a date range and contact information about invited participants' or 'Meeting initiation shall be restricted to authorized personnel'.

7.2 The granularity of goals and their relationship to requirements and assumptions

We obviously need to clarify the commonalities and differences between this notion of goal and the notions of requirement and assumption defined in Section 1.1.4 and used in the previous chapters.

The granularity of goals

Goals can be stated at different levels of abstraction:

- At higher levels, there are coarser-grained goals stating strategic objectives related to the business or the organization, for example:

The system's transportation capacity shall be increased by 50%.

The library system shall provide effective access to the state of the art.

Meetings shall be scheduled so as to maximize the attendance of invited participants.

- At lower levels, there are finer-grained goals stating technical objectives related to system design options, for example:

Acceleration commands shall be sent to every train every 3 seconds.

Reminders for non-returned book copies shall be issued every week after expiration of the loan period.

When no date can be found outside all excluded dates returned by participants, the constraints should be weakened by dropping the constraints of low-priority participants.

This difference in goal level and granularity suggests a specification-structuring mechanism based on *contribution links* among goals. A coarser-grained goal may be *refined* into finer-grained goals contributing to it or, conversely, finer-grained-goals may be *abstracted* towards coarser-grained goals to which they contribute. For example, the lower-level goal:

Acceleration commands shall be sent to every train every 3 seconds

is expected to contribute to the higher-level goal:

The system shall guarantee safe transportation of all passengers

through intermediate higher-level goals along a contribution chain. We will come back to this briefly in Section 7.4 and, in much greater detail, in Chapter 8.

Goals, requirements and expectations

This notion of goal and the varying granularity of goals bring us to a next important point. *The finer-grained a goal is, the fewer agents are required to satisfy it.* For example, the previous safe transportation goal was seen to require some cooperation among on-board train controllers, the station software, the network communication infrastructure, the tracking system, doors actuators, passengers and so on. Consider now the following finer-grained sub-goal contributing to it:

All train doors shall always remain closed while a train is moving.

We expect this goal to involve much fewer agents; namely, the corresponding train controller, speed sensor and door actuators. Consider now the following finer-grained sub-goal contributing to the latter goal:

DoorsState shall always have the value 'closed' when *measuredSpeed* has a non-null value.

This goal now only requires a single agent to enforce it; namely, the train controller software.

In this setting, the notions of requirement and assumption defined in Section 1.1.4 can be related to the notion of goal as follows.

A *requirement* is a goal under the responsibility of a single agent of the software-to-be. For example, the preceding statement is a requirement. Similarly, the following statements are requirements:

Acceleration commands shall be sent to every train every 3 seconds.

A loan coupon shall be issued if there is a copy of the requested book in the library database.

The scheduled meeting date and location shall be sent to the e-mail address of every participant on the invitee list once they have been validated by the meeting initiator.

An *expectation* is a goal under the responsibility of a single agent in the environment of the software-to-be. For example, the following statements are expectations:

Passengers get out of the train when the train doors are open at their destination platform.

Acceleration commands sent to a train controller shall be received by it within X seconds at most.

A borrower will return borrowed book copies within a week after the second threatening reminder.

An invited participant will attend the meeting if the notified date and location fit the constraints he or she has returned.

The statement in the first example above is an assumption about the environment. It prescribes a specific behaviour for passenger agents. In the second example, it prescribes a behaviour for the communication infrastructure. (This assumption is required to achieve the higher-level goal of safe train acceleration based on real-time estimates of speed and position of the preceding train; otherwise the train might accelerate or decelerate based on outdated information about the preceding train.) By definition, expectations cannot be enforced by the software-to-be, unlike requirements.

Types of statement revisited in the presence of goals

Section 1.1.4 introduced various types of statement involved in the RE process, including system requirements, software requirements, domain properties and assumptions. The introduction of the goal concept results in a slight refinement of the terminology introduced there.

A *requirement* corresponds to what was called before a software requirement.

A *goal* might correspond to a system requirement or not, depending on which system agents are involved in its satisfaction; a system requirement as defined in Section 1.1.4 necessarily involved the software-to-be.

A *domain property* is still a descriptive statement about the environment, expected to hold invariably regardless of how the system behaves.

An assumption was defined in Section 1.1.4 as a statement to be satisfied by the environment. We observe now that some assumptions are prescriptive for environment agents. These are expectations as defined in this section. Other assumptions are descriptive, for example:

Train tracks are in good conditions except the track segment X under maintenance.

This statement does not prescribe any agent behaviour. It is descriptive, but it is not a domain property. The assumption states some factual information that is subject to change, in this example once the segment maintenance has been done. This is not the case for a domain property such as:

A train is moving if and only if its physical speed is non-null.

Environment assumptions will accordingly be specialized into expectations and hypotheses. An expectation as just defined is a prescriptive statement to be satisfied by a single environment agent; a *hypothesis* is a descriptive statement satisfied by the environment and subject to change. Here are a few more examples of hypotheses:

The university library has about 20,000 patrons.

Saturdays are excluded dates for meetings.

Figure 7.1 summarizes our refined typology of statements. In this typology, goals, requirements, expectations, domain properties and hypotheses are all formulated in terms of environment phenomena, in the vocabulary of stakeholders, with the restriction that requirements are now formulated only in terms of phenomena shared between the software and the environment. We will not talk about system requirements any further.

The importance of the distinction between descriptive and prescriptive statements should be recalled here. Goals may need to be refined into sub-goals, negotiated with stakeholders, assigned to agents responsible for them, weakened in the case of conflicts, or strengthened or abandoned in the case of unacceptable exposure to risks. Domain properties and hypotheses do not undergo such manipulation. Unlike prescriptive statements, they are not subject to decisions in the RE process. They are just there, for use in the RE process, and are pre-assigned to organizational components or Mother Nature.

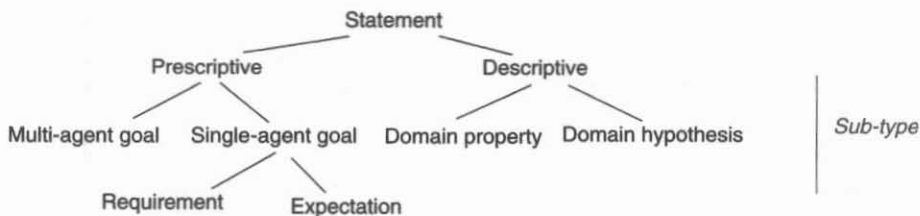


Figure 7.1 Statement typology with goals

7.3 Goal types and categories

Goals can be classified along two dimensions. On the one hand, a goal may be of one *type* or another depending on whether it prescribes intended system behaviours or preferences among alternative behaviours. On the other hand, goals allow us to capture functional and non-functional properties under a single abstraction; a goal may belong to one *category* or another depending on whether it prescribes a functionality or a quality constraint.

This section defines and illustrates the various types and categories of goals that we may find in the RE process. Chapter 8 will introduce diagrammatic notations for capturing them, together with techniques for building intentional system models through goal diagrams. Chapters 16 and 18 will detail further techniques for analysing such models.

7.3.1 Types of goal: behavioural goals vs soft goals

Figure 7.2 shows a taxonomy of goal types. The types there do not overlap; a goal is either a behavioural goal or a soft goal; a behavioural goal is either an *Achieve* goal or a *Maintain/Avoid* goal.

Behavioural goals

Such goals prescribe intended system behaviours declaratively. A *behavioural goal* implicitly defines a maximal set of admissible system behaviours.

A *system behaviour* is composed of parallel behaviours of the agents forming the system. An *agent behaviour* is captured by a sequence of state transitions for the items that the agent controls. These items are generally represented by *state variables*. A *state* of a variable x is a functional pair (x, v) where v denotes a particular value for x . A global system state is an aggregation of the state of all variables characterizing the system.

Figure 7.3 shows one admissible sequence of state transitions implicitly defined by the behavioural goal 'All train doors shall always remain closed while a train is moving'. There are implicitly two controlled state variables in this example: *trainMovement* and *trainDoors*, say. A possible state for the latter variable is the pair (*trainDoors*, 'closed'), denoted by *Closed* in Figure 7.3 for short. Each state there aggregates two sub-states: one for the controlled *trainMovement* and one for the controlled *trainDoors*.

A behavioural goal can be established in a clear-cut sense. We can always determine whether such a goal is established or not; that is, whether a specific behaviour is a member of the set of admissible behaviours implicitly defined by the goal. The phrase *goal satisfaction* thus makes sense for a behavioural goal, as we can observe that the goal is satisfied or that it is not.



Figure 7.2 A taxonomy of goal types

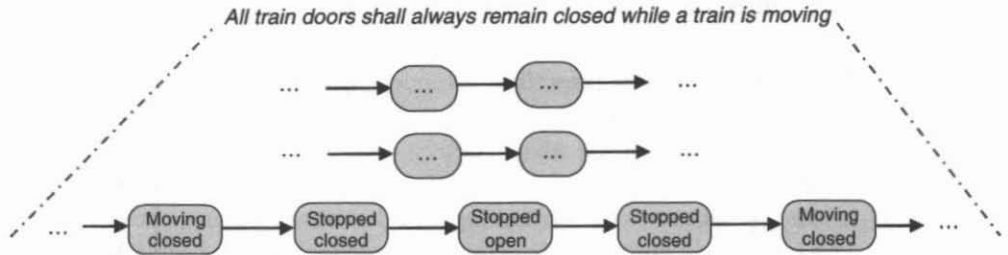


Figure 7.3 Behavioural goals prescribe intended system behaviours

As we will see in Part II of the book, behavioural goals may be used for building operational models of the target system such as UML use cases or state diagrams.

Behavioural goals can be further specialized into *Achieve* goals and *Maintain* goals (Dardenne *et al.*, 1993); see Figure 7.4.

Achieve goals An *Achieve goal* prescribes intended behaviours where a target condition must *sooner or later* hold whenever some other condition holds in the current system state. When we want to highlight the goal type, we will prefix the goal name with the corresponding keyword. For an *Achieve* goal we will write: *Achieve* [TargetCondition].

The corresponding specification has the following informal temporal pattern:

Achieve [TargetCondition]: [If CurrentCondition then] sooner-or-later TargetCondition.

In this specification pattern, the *CurrentCondition* prefix is optional (in other words, it can be true).

An *Achieve* goal is sometimes denoted by *Cease* [TargetCondition] for the dual variant of this pattern:

Cease [TargetCondition]: [If CurrentCondition then] sooner-or-later **not** TargetCondition.

- *Examples.* Here is a sample of *Achieve* goals, with the name of the goal followed by the pattern instantiation:

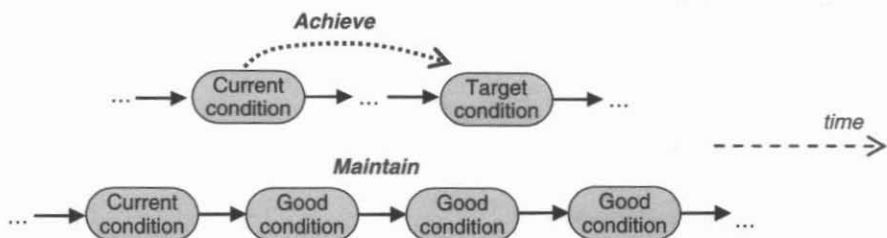


Figure 7.4 Behavioural goals: Achieve and Maintain goals

Achieve [MeetingRequestSatisfied]:

if a meeting is requested **then sooner-or-later** the meeting takes place and is attended by all important invited participants.

Achieve [BookRequestSatisfied]:

if a book is requested **then sooner-or-later** a copy of the book is borrowed by the requesting patron.

Achieve [TrainProgress]:

if a train is at some platform **then sooner-or-later** the train is at the next platform.

In many instances of this pattern the target condition has to be established *sooner* rather than later, of course. This must be made further precise in the actual goal specification – typically, by prescribing a time interval within which the target condition must become true, for example:

Achieve [BookRequestSatisfied]:

if a book is requested **then within a week** a copy of the book is borrowed by the requesting patron.

Achieve [TrainProgress]:

if a train is at a platform **then within 5 minutes** the train is at the next platform.

Maintain goals A *Maintain goal* prescribes intended behaviours where a ‘good’ condition must *always* hold (possibly under some other condition on the current state). Its specification takes the following informal temporal pattern:

Maintain [GoodCondition]:
in particular:

[if CurrentCondition then] always GoodCondition,
always (if someCondition then GoodCondition).

A *Maintain goal* may also be denoted by *Avoid [BadCondition]* for the dual variant of this pattern:

Avoid [BadCondition]:

[if CurrentCondition then] always not BadCondition.

- *Examples.* Here is a sample of *Maintain goals*:

Avoid [ParticipantConstraintsDisclosed]:

always not participant constraints disclosed to other invited participants.

Maintain [AccurateBookClassification]:

if a book is registered in the library directory **then always** its keyword-based classification is accurate.

Maintain [DoorsClosedWhileMoving]:

always (if a train is moving then its doors are closed).

Maintain [WorstCaseStoppingDistance]:

always (if a train follows another then its distance is sufficient to allow the other to stop suddenly).

There are instances of the *Maintain/Avoid* pattern where the good/bad condition always has to be maintained/avoided up to some point, for example where another condition becomes true; this must of course be made further precise. We will come back to more specific *Achieve/Maintain* patterns in Part II while discussing goal-specification techniques.

Note that an *Achieve* goal refers to *some* future state, from an arbitrary current one, whereas a *Maintain/Avoid* goal refers to *every* future state (possibly up to some point).

Soft goals

Unlike behavioural goals, *soft goals* prescribe preferences among alternative system behaviours. They are more fulfilled along some alternatives and less along others.

Consider the following goal in our meeting scheduling system, for example:

Interactions with invited participants should be limited as much as possible.

This goal prescribes that behaviours where there are fewer interactions (for example through e-agenda access) are to be preferred to behaviours where there are more interactions (for example through e-mail requests and reminders). Here is an additional sample of soft goals for our running case studies:

The stress underpinning the working conditions of train drivers should be reduced.

Passengers should be better informed of flight connections and airport facilities.

The meeting scheduler software should be easy to use by administrative staff.

The workload of library staff members should be reduced.

A soft goal cannot be established in a clear-cut sense. For example, we cannot say in a strict sense whether such or such a system behaviour in isolation satisfies the stress-reduction goal or not. We might, however, say that one system behaviour may reduce stress further than another. Put in more general terms, the phrase 'goal satisfaction' does not make too much sense for soft goals, as we cannot observe that they are satisfied by some behaviours and not satisfied by others. The phrase *goal satisficing* is sometimes used instead; a soft goal is more satisfied in one alternative than in another (Chung *et al.*, 2000).

Soft goals are therefore used as criteria for selecting one system option among multiple alternatives – in particular, one goal refinement among alternative ones, one risk countermeasure among alternative ones, one conflict resolution among alternative ones, or one goal responsibility assignment among alternative candidate agents. We will come back to this in Chapter 16.

In a way similar to *Achieve* and *Maintain* behavioural goals, we may prefix a soft goal name by a keyword indicating a corresponding pattern, such as:

Improve [TargetCondition],

Increase [TargetQuantity],

Maximize [ObjectiveFunction],

Reduce [TargetQuantity],

Minimize [ObjectiveFunction].

As already introduced in Section 4.2.1, the specification of a soft goal should in addition clearly state a fit criterion to ensure its measurability.

Soft goals are not the same as goals whose satisfaction is required in $X\%$ of cases. The latter are behavioural goals sometimes called *probabilistic goals*. Unlike soft goals, we can observe whether a probabilistic goal is satisfied or not; satisfaction is only required 'on some average' due to limited resources, possibility of uncontrollable failure and so on. For example:

The meeting scheduler should determine schedules that fit the diary constraints of all invited participants in at least 80% of cases.

The distance between two consecutive trains should always be greater than the worst-case stopping distance of the train in front, with a mean time between hazards of the order of 10^9 hours.

Thus, the phrases 'satisficing' and 'partial degree of satisfaction' do not mean the same. The current state of the art in evaluating, validating and verifying probabilistic goals is still fairly limited, although such goals are frequently found in software projects (Letier & van Lamsweerde, 2004).

7.3.2 Goal categories: Functional vs non-functional goals

In a manner quite similar to the requirements categories introduced in Section 1.1.5, *goal categories* establish a distinction between functional and non-functional goals together with their respective specializations:

- A *functional goal* states the intent underpinning a system service, for example schedule convenient meetings, control acceleration of trains for rapid transportation.
- A *non-functional goal* states a quality or constraint on service provision or development, for example minimum interaction with participants during meeting scheduling, safe distance maintained between trains.

Figure 7.5 shows a goal taxonomy corresponding to the requirements taxonomy discussed in Section 1.1.5 (see Figure 1.5):

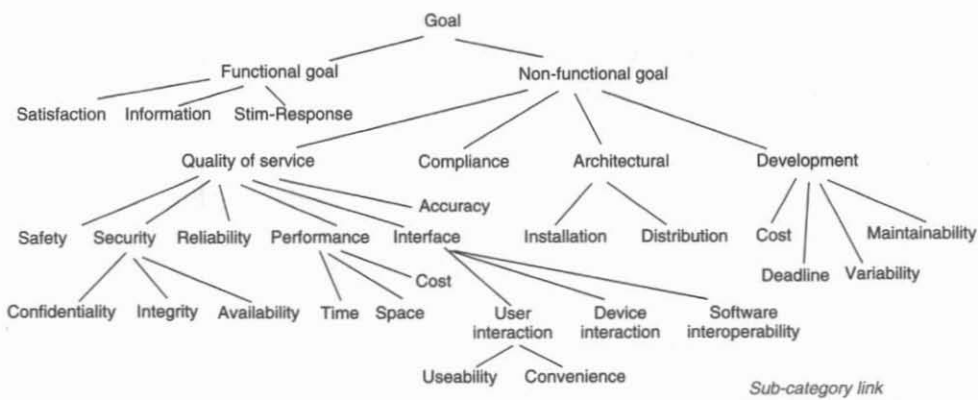


Figure 7.5 Goal categories

- *Satisfaction goals* are functional goals concerned with satisfying agent requests, for example 'Every relevant book request should be satisfied' or 'Every requested meeting should be scheduled if feasible'.
- *Information goals* are functional goals concerned with keeping agents informed about important system states, for example 'Passengers should be informed of what airlines are served at the next station' or 'Every invited participant should be notified as soon as possible of the scheduled meeting date and location'.
- *Stimulus-response goals* are functional goals concerned with providing appropriate responses to specific events, for example 'The on-board controller shall update the train's acceleration to the commanded one immediately on receipt of an acceleration command from the station computer'.
- *Accuracy goals* are non-functional goals requiring the state of variables controlled by the software to reflect the state of corresponding quantities controlled by environment agents accurately, for example 'The train's physical speed and commanded speed may never differ by more than X m.p.h.' or 'A copy of a book is stated as available if and only if it is physically on the shelves'.
- *Security goals* prescribe different types of protection of agent assets against unintended behaviours; such behaviours might be accidental or malicious, for example 'Date preferences of important participants may never be disclosed to anyone else than the meeting initiator' or 'Meeting initiation shall be restricted to authorized personnel'.

The definition of other goal categories in Figure 7.5 is obtained by transposing the corresponding definitions in Section 1.1.5.

For similar reasons to those discussed there, goal categories may overlap – unlike goal types. For example, consider the following goals:

Acceleration commands shall be sent to every train every 3 seconds
An ambulance shall be at the incident scene within 11 minutes

These goals are both functional and non-functional; they might be classified as satisfaction goals, safety goals and performance goals.

Using goal categories

In spite of their possible overlap, goal categories may prove useful in different RE contexts:

- *For eliciting or finding missing goals.* We may ask ourselves whether the RD covers all relevant goal categories for our system, for example:
 - Is there any accuracy goal for this critical system information?
 - Is there any confidentiality or privacy goal to protect this sensitive system information?

- *For detecting conflicts among goals.* We may reuse knowledge about negative interactions among the corresponding categories, for example:
 - Is there any conflict to be resolved between information and confidentiality goals about related information?
 - Is there any conflict to be resolved between accountability and confidentiality goals in the system?
 - Is there any conflict to be resolved between security and usability goals in the system?
- *For conflict resolution.* We may encode and reuse conflict-resolution heuristics, for example:
 - Safety goals have the highest priority in conflict resolution.
- *For specifying goals.* We may reuse heuristic rules guiding us in the specification process, for example:
 - Confidentiality goals are *Avoid* goals about knowledge that agents may have about each other.

Difference between goal types and goal categories

Types and categories provide independent ways of classifying the goals that we may consider in a system. The partition of goals into behavioural and soft goals is completely different from their classification into functional and non-functional goals. In the former case, the discriminating criterion is a semantic one; that is, whether a goal is satisfied by system behaviours in a clear-cut sense or not. In the latter case, the classification criterion is a pragmatic, more fuzzy one; that is, whether a goal prescribes intended system services or quality constraints on such services.

We should therefore not confuse soft goals with non-functional goals (as sometimes done in the RE literature). For example, goals such as:

Avoid constraints disclosure to other participants

Maintain doors closed while the train is moving

Participants should return their constraints within a week at most after request

would typically be classified as non-functional goals in the *Confidentiality*, *Safety* and *TimePerformance* category, respectively. All three are behavioural goals, though; for each of them, we can recognize whether a system behaviour satisfies it or violates it.

In practice, certain goals within a category are more frequently of a specific type. For example, many functional goals are expected to be *Achieve* goals. Many safety, security and accuracy goals are expected to be *Maintain* goals. User interaction, cost, development and some performance goals tend to be *soft goals*. In particular, ‘-ility’ goals stating system-specific objectives related to usability, flexibility, maintainability or reusability are often soft goals.

7.4 The central role of goals in the requirements engineering process

There are multiple reasons for goals being so important in the RE process:

- *Goal refinement provides a natural mechanism for structuring complex specifications at different levels of concern.* A goal may be refined in a set of sub-goals that jointly contribute to it. For example, the high-level goal of *safe train transportation* can be partially refined into three sub-goals shown in Figure 7.6; namely, (1) Avoid train collisions, (2) Maintain train speed under the speed limit of the block the train is on, and (3) Maintain doors closed while the train is moving. These three sub-goals jointly contribute to the parent, higher-level goal. We may refine each of them in turn into finer-grained goals until we reach requirements on the software and expectations on the environment. (Chapter 8 will discuss goal refinement in much greater detail.)
- *Goals provide the rationale for requirements.* When the reason for a specific requirement is unclear, we can clarify it by checking what goals the requirement contributes to (if any). We can also explain the requirements to stakeholders by browsing goal refinement graphs such as the one in Figure 7.6 to visualize the contributions of requirements to higher-level goals
- *Goals drive the identification of requirements to support them.* Conversely, when a goal appears during the elicitation or evaluation phases of the RE process, we can explore various ways of achieving it to obtain new requirements contributing to it. (Chapter 15 will detail a method for goal-based requirements elaboration.)
- *Goals provide a richer structure for satisfaction arguments.* As suggested in Section 6.3.1, contribution links in a refinement graph provide the basis for chains of satisfaction arguments to show that higher-level concerns are met by lower-level ones (see Figure 6.10). Each argument in such chains takes the following form:

$$\{SUBGOALS, ASM, DOM\} \models ParentGoal,$$

meaning: 'the goal *ParentGoal* is satisfied if its sub-goals in the set *SUBGOALS* are satisfied, assuming that the environment expectations in the set *ASM* and the domain properties and hypotheses in *DOM* are satisfied'. The bottom-end argument in such chains takes the form:

$$\{REQ, ASM, DOM\} \models ParentGoal,$$

where *REQ* is a set of requirements. (We will come back to refinement-based satisfaction arguments in Chapters 8, 12 and 18.)

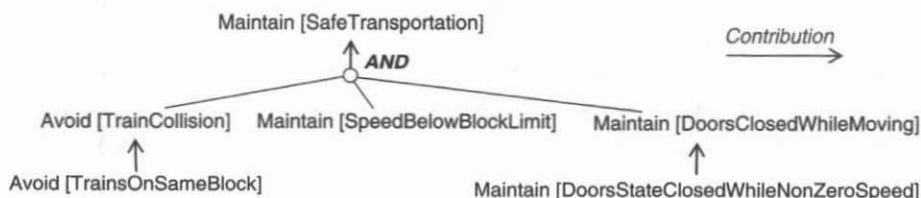


Figure 7.6 Goal AND-refinement and contributions

- *Goals provide a basis for showing the alignment of the system-to-be with the organization's strategic objectives.* Using such chains of satisfaction arguments, we can convince decision makers that the system-to-be will be in alignment with the organization's business goals at the top end of some chains – after all, this is what really matters to them.
- *Goals provide a precise criterion for requirements completeness.* A set of requirements is *complete* with respect to a set of goals if all the goals can be shown to be satisfied when the requirements are satisfied, assuming the environment assumptions and domain properties to hold. In other words, a set *REQ* is complete with respect to a set *GS* of goals, in view of known properties and hypotheses in *DOM* and expectations in *ASM*, if:

for all *G* in *GS*, we can provide a satisfaction argument $(REQ, ASM, DOM) \models G$

Achieving requirements completeness is a major concern of the RE process (see Section 1.1.7). Now we have a precise criterion we can check. A problem of course remains that the set of goals must be complete. Part III of the book will present techniques aimed at this, such as refinement checking, obstacle analysis, threat analysis and goal-oriented model animation.

- *Goals provide a precise criterion for requirements pertinence.* A requirement is *pertinent* with respect to a set of goals if it is used in the satisfaction argument of one goal at least. In other words, *r* in *REQ* is pertinent if:

for some goal *G*, *r* is needed in a satisfaction argument $(REQ, ASM, DOM) \models G$

Ensuring requirements pertinence is another important RE concern (see Section 1.1.7). Now we have a precise criterion we can check for this.

- *Goals provide a natural way of structuring the RD.* Goal AND-refinement structures may be used for organizing the RD, or portions of it, in a corresponding tree structure of sections and sub-sections. In fact, we can generate the RD semi-automatically from a goal-oriented model, as we will see in Chapter 16, Section 16.4.
- *Goals provide anchors for risk analysis.* A risk is usually defined as a loss of attainment of some corresponding objective (see Section 3.2). We can thus identify risks as goal negations, build risk trees as refinements of goal negations, and explore countermeasures as new goals to reduce, mitigate or reduce the identified risks. This can be done, in particular, for hazard analysis in safety-critical systems or for threat analysis in security-critical systems. (We will come back to goal-based risk analysis in Chapters 9, 16 and 18.)
- *Goals provide the roots for managing conflicts among requirements.* Conflicts among requirements are generally due to conflicts between the underlying goals to which the requirements contribute. The conflicting goals are owned by different stakeholders who have different viewpoints and concerns. For example, a goal of library staff members might be that:

whereas a borrower's goal might be that:

Borrowed book copies should be kept as long as they are needed.

To manage conflicts at goal level, we need to detect conflicts among goals; explore alternative conflict-resolution goals; select the best one based on soft goals; and propagate the selected resolution down to the requirements level. Conflict management at goal level proves much more effective. If we just handle apparent conflicts at the requirements level, we have no guarantee that a specific resolution among requirements would resolve the root conflict among the underlying goals; residual conflicts originating from the same root cause may be left unresolved elsewhere in the RD. (We will come back to goal-based conflict management in Chapters 16 and 18.)

- *Goals provide a criterion for delimiting the scope of the system.* The system is bounded by a set of goals to be satisfied through cooperation of 'good' agents. 'Bad' agents may obstruct some of these goals or threaten them. The set of 'good' and 'bad' agents defines the system's scope.
- *Goals support the analysis of dependencies among agents.* We can capture such dependencies in i^* diagrams that show agents depending on others for specific goals to be satisfied (Yu, 1993). Such diagrams can then be used for analysing strategic dependencies among organizational agents or for analysing dependency-based vulnerabilities in the system. (We come back to agent dependencies and i^* diagrams in Chapter 11.)
- *Goals provide a basis for reasoning about alternative options.* A system goal can be refined through alternative combinations of sub-goals. A fine-grained goal can be operationalized through alternative combinations of system services. The responsibility for a goal can be assigned to alternative agents. Incidental or malicious threats to a goal can be resolved through alternative countermeasure goals. A conflict among goals can be handled through alternative resolution goals. Those different types of alternatives are all captured at goal level, and generally result in fairly different system proposals. Goals thus provide the right RE abstraction for making alternative options explicit, exploring their respective implications and making effective decisions based on contributions to soft goals. For example, Figure 7.7 shows two alternative refinements of the goal *Avoid [TrainCollision]*. The first option consists of allowing one train at most on each block; the second option consists of allowing multiple trains to be on the same block provided that a worst-case stopping distance is maintained between successive trains. (Goal-based identification and

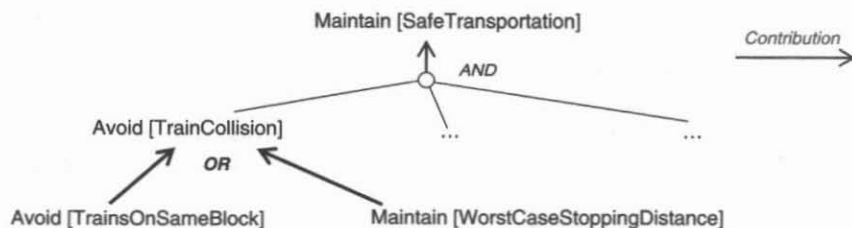


Figure 7.7 Goal OR-refinement, alternative options, and system versions

assessment of alternatives will be discussed in greater detail throughout Parts II and III of the book.)

- *Goals support traceability management.* As chains of satisfaction arguments are available from a goal-oriented RE process, there is no need to discover traceability links for evolution support – we get derivational traceability links for free, from low-level technical requirements to high-level strategic objectives (see Section 6.3.1).
- *Goals provide essential information for evolution support.* Section 6.2 discussed the identification of levels of stability and commonality as another important concern for managing requirements evolution. A requirement represents one particular way of achieving some underlying goal; the requirement is therefore more likely to evolve, towards another way of achieving that same goal, than is the parent goal itself. The higher level the goal, the higher its stability through successive revisions and the higher its commonality across variants of a system family. Multiple system revisions and variants generally share a common set of higher-level goals; they differ on the lower-level ones. In particular, the system-as-is and the system-to-be may correspond to alternative refinements of common parent goals. Their respective goals can be integrated into a single model, as we will see in Part II of the book. In Figure 7.7, the left-hand option contributing to the goal *Avoid [TrainCollision]* might correspond to the system-as-is, whereas the option on the right might correspond to the system-to-be, where more trains are allowed per block in order to increase train frequency and transportation capacity.

To sum up this discussion on the central role of goals in the RE process, we might say that requirements ‘implement’ goals much the same way as programs implement design specifications. Without a specification, we cannot develop the correct program that meets the specification. Without goals, we cannot engineer complete, consistent, pertinent and adequate requirements that meet them.

7.5 Where are goals coming from?

Goal identification is not necessarily an easy task. Sometimes goals are stated explicitly as system objectives during elicitation sessions. More often they are kept implicit and we need to ‘mine’ them from preliminary documents and from elicited material such as scenarios, workflow descriptions and other operational details provided by stakeholders.

The preliminary analysis of the system-as-is is an important source for goal identification. At higher levels, the business goals of the organization are generally apparent. Domain analysis also results in a list of complaints, deficiencies and other problems (see Chapter 2). *Avoiding* or *reducing* each of the listed problems yields an additional list of candidate goals to start from.

Another simple and quite effective means for goal identification consists of searching through preliminary documents and interview transcripts for intentional keywords, such as ‘in order to’ or ‘so that’ and keywords in the optative mood, such as ‘has to’ or ‘shall’. Chapter 8 will detail these heuristics and others.

When goals are explicitly stated as system objectives, we may ask *HOW* questions during elicitation to find out contributing sub-goals. When they are kept implicit in the scenarios and operational descriptions being elicited, we may ask *WHY* questions to make them explicit.

In view of the central role played by goals in the RE process, the sooner a goal is identified and validated, the better. This does not entail any sort of waterfall-like requirements elaboration process, though. There is some inevitable intertwining of goal identification and requirements elaboration. Some goals may be identified in the elicitation phase; other goals may result from the requirements evaluation phase where problems are found such as conflicts and overexposure to risks; yet other goals may appear in the requirements quality assurance phase where problems are found such as omissions or inadequacies.

A variety of more sophisticated techniques for goal identification, refinement and abstraction will be detailed throughout Parts II and III of the book while discussing techniques for building and analysing goal models.

7.6 The relationship of goals to other requirements-related products and processes

This section relates goals to other RE artefacts such as scenarios, operational specifications, use cases and properties used in finite-state verification. The aim is to highlight the differences and complementarities among these notions. Goal orientation is then briefly compared with other candidate paradigms for RE such as agent orientation, object orientation and top-down analysis.

7.6.1 Goals and scenarios

Goals prescribe admissible or preferred behaviours of the agents forming the system. They are related to each other through refinement/contribution links. As we will see in Part II, such links can be gathered in a goal model to support various forms of early, declarative and incremental reasoning for goal refinement and operationalization, completeness checking, conflict management, hazard analysis, threat analysis, goal-oriented animation and generation of the requirements document. On the other hand, goals are sometimes felt to be too abstract by stakeholders. They cover classes of intended behaviours, but such behaviours are left implicit. Goals may sometimes be hard to elicit and make fully precise in the first place.

Scenarios capture typical examples or counterexamples of intended system behaviour through sequences of interactions among agents (see Sections 2.2.5 and 4.3.6). Their narrative and concrete style of description make them appropriate for requirements elicitation and validation, as they are easily accessible to stakeholders. On the other hand, scenarios are inherently partial and cover few behaviours of specific instances. They leave intended system properties implicit, and may entail premature decisions about event sequencing and distribution of responsibilities among agents.

Goals and scenarios thus have complementary strengths and weaknesses. As Parts II and III will show in greater detail, they can be combined synergistically for building and analysing system models.

The semantic relationship between goals and scenarios should become clear at this point. A behavioural goal implicitly defines a maximal set of admissible system behaviours (see Section 7.3.1). Such behaviours are composed of parallel agent behaviours, each captured by a sequence of state transitions of items that the corresponding agent controls. We can therefore make these state transitions explicit along agent timelines through corresponding agent interactions. A behavioural goal then *covers* the set of all positive scenario examples that illustrate

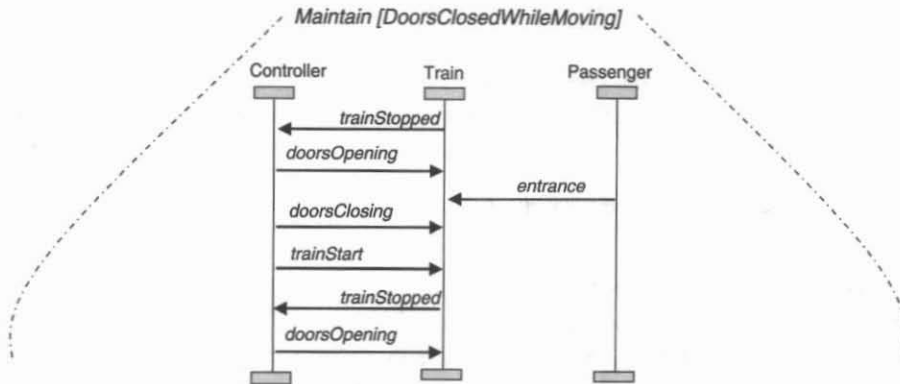


Figure 7.8 Scenario coverage by behavioural goals

it. Figure 7.8 depicts this coverage relationship between goals and scenarios by showing one member of the set of scenarios covered by the goal *Maintain[DoorsClosedWhileMoving]*.

7.6.2 Intentional and operational specifications

Goal specifications assert system intentions declaratively. Such intentional specifications allow us to reason early and incrementally on partial specifications, as we will see in Chapters 16–18.

On the other hand, most specification paradigms reviewed in Chapter 4 are operational; they refer to system operations, their data dependencies, their pre- and post-conditions, their laws of composition or the dynamics of their applications in terms of state transitions. Operational specifications support later forms of analysis on an entire specification, such as specification animation or model checking.

Intentional specifications leave the operations realizing them implicit, whereas operational specifications leave the intentions underlying them implicit. In terms of the various forms of analysis that they support, intentional and operational specifications complement each other. Chapter 12 will show how they can be integrated in a uniform framework, through a systematic mapping called goal *operationalization*, in order to support complementary types of analysis.

7.6.3 Goals and use cases

Use cases are a popular albeit fairly fuzzy form of operational specification (see Section 4.3.5). As their specification does not convey much, use cases are not really amenable to useful forms of analysis. However, they provide an outline view of the operations that an agent has to perform; such a view may prove useful for elicitation and communication with stakeholders. Like any operational specification, use cases leave their underlying functional and non-functional objectives implicit.

In Chapter 12 we will see how use cases can be easily generated from goal operationalizations.

7.6.4 Goals and model-checked properties

The properties that we can model check on state machine specifications can be descriptive or prescriptive (see Section 5.4.3). These properties are traditionally classified as safety or liveness

properties; a *safety property* states that a 'bad' state may never be reached, whereas a *liveness property* states that a 'good' state is eventually reached.

A goal can of course be model-checked on an operational specification, like any prescriptive property. We deliberately avoid the standard safety/liveness terminology in this book for specific reasons:

- To avoid confusion between safety properties and safety requirements found in RE projects.
- To avoid confusion between safety/liveness properties and *Maintain/Achieve* goals. Unlike liveness properties, *Achieve* goals need to be bounded for realizability by system agents – the latter could otherwise unboundedly defer goal achievement; see Chapter 11. Strictly speaking, realizable *Achieve* goals are safety properties.

The important difference between goals and model-checked properties lies in the context and structure into which they fit. Goals pertain to AND/OR graphs that capture their refinement/contribution links together with their assignments to system agents. These graphs can be built systematically and are amenable to various types of earlier analysis, as we will see in the next chapters. Properties in the traditional model-checking sense do not come from structures built in a systematic way.

7.6.5 Goal orientation and agent orientation

Goal orientation is sometimes contrasted with agent orientation. On the one hand, the RE process may be more driven by the goals to be fulfilled by the system, from which the agents and their relationships are identified and analysed. On the other hand, the RE process may be more driven by the agents in the system and their dependencies, from which the goals are identified and analysed.

There is no fundamental difference between those two views, though. In goal-oriented approaches, the goals are identified and analysed based on the agents that can satisfy them. In agent-oriented approaches, the agents are identified and analysed based on the goals that they need to satisfy. Goal analysis and agent analysis are thus highly intertwined; a goal-oriented approach to RE is agent oriented as well, and vice versa. The difference is more a pragmatic matter of which driver, goal or agent is more effective at which stage of the RE process. The answer to this may depend on the domain and the type of project.

7.6.6 Goal orientation and object orientation

Chapter 10 will show how a conceptual object model can be derived systematically from a goal refinement graph and the specifications annotating this graph. The object model structures domain concepts in an extended entity-relationship form represented by a UML class diagram.

This is not to say that a goal-oriented approach to RE is object oriented. Object orientation, as usually defined, does not make any sense while engineering requirements. We certainly need specialization with inheritance as a concept-structuring mechanism, but we don't need the other basic ingredient of object orientation; that is, data encapsulation. The latter is a design concern; clustering operations and hiding implementation choices is not relevant at the RE stage as we are not concerned with design and implementation issues at all.

7.6.7 Goal orientation and top-down analysis

Some people sometimes mistakenly believe that goal-oriented approaches are top-down approaches, where we start from very high-level goals and proceed by stepwise goal decomposition until the requirements are obtained. This is by no means the case, of course; see Section 7.5. The material obtained from elicitation sessions most often covers a mix of strategic concerns, operational procedures, scenario examples, complaints, low-level details and so forth. No RE process can be made linear. We need to backtrack on issues as risks, conflicts and other problems become apparent.

In order to build a complete, consistent and adequate picture of the system-to-be from this hybrid and ill-assorted material, a goal-oriented approach must necessarily combine top-down and bottom-up processes driven by *HOW* and *WHY* questions, respectively.

Summary

- Goals provide a basic abstraction for addressing the *WHY* dimension of requirements engineering. A goal is a prescriptive statement of intent that the system should satisfy through cooperation of its agents. An agent is an active system component that can restrict its behaviour by adequate control of system items so as to satisfy the goals for which it is responsible. Goal-oriented requirements engineering refers to the use of goals for requirements elicitation, evaluation, elaboration, structuring, documentation, analysis and evolution.
- Goals can be defined at different level of abstraction. A goal at one level can be refined into sub-goals at the next level. Higher-level goals are coarser grained. The finer grained a goal is, the fewer agents are required to satisfy it. A requirement is a goal assigned to a single agent of the software-to-be. An expectation is a goal assigned to a single agent in the environment of the software-to-be. Unlike domain properties and hypotheses, goals may be refined, negotiated, assigned as responsibilities to agents and transformed in case of conflict or overexposure to risks.
- There are different types of goals. A goal is either a behavioural goal or a soft goal. A behavioural goal prescribes intended system behaviours declaratively. It can be established in a clear-cut sense; a system behaviour satisfies it or not. A behavioural goal is an *Achieve* goal or a *Maintain* goal according to the pattern of temporal behaviour that it prescribes. *Achieve* goals prescribe behaviours where some target condition must sooner or later become true. *Maintain* goals prescribe behaviours where some 'good' condition must be maintained or some 'bad' condition must be avoided. Behavioural goals are used for building operational specifications of the system. A soft goal prescribes preferences among alternative system behaviours. It is more fulfilled along some alternatives and less along others. Soft goals cannot be established in a clear-cut sense. They are used as criteria for selecting system options among multiple alternatives.

- There are different categories of goals that may overlap. A functional goal states the intent underpinning a system service. Satisfaction goals and information goals are common sub-categories of functional goals. A non-functional goal states a quality or constraint on service provision or on system development. Accuracy, safety and security goals deserve special attention in mission-critical systems.
- Goal types and categories provide different, independent ways of guiding requirements elicitation, specification and analysis. Non-functional goals should not be confused with soft goals. Many safety, security and accuracy goals are behavioural *Maintain* goals.
- Goal refinement provides a natural mechanism for structuring complex specifications at different levels of concern. A goal refinement graph shows the refinement and contribution links among goals. Requirements and expectations appear as leaf nodes in this graph.
- Goals play a crucial role in the RE process. They provide the rationale for requirements, drive the identification of requirements to support them and bound the scope of the system. They yield a precise criterion for requirements completeness and pertinence. Goal contribution links yield chains of satisfaction arguments that can be used for showing the alignment of the system-to-be with the organization's strategic objectives. Goals also provide anchors for risk analysis, conflict management and comparison of alternative options. Goal refinement structures provide useful information for evolution support and traceability management.
- Goals can be identified in all phases of the RE process, in particular during requirements elicitation, evaluation and consolidation. They are generally found by top-down refinement of higher-level concerns *and* by bottom-up abstraction from lower-level material such as scenario examples and operational descriptions.
- The complementary strengths and benefits of goals, scenarios and operational specifications call for their combined use for more effective requirements elicitation, evaluation and quality assurance.

Notes and Further Reading

Many old software development methodologies considered organizational objectives and user goals for early forms of analysis – called context analysis (Ross & Schoman, 1977b), definition study (Hice *et al.*, 1974) or participative analysis (Munford, 1981). This natural practice has led requirements documentation standards such as IEEE-Std-830 to require a specific document section for the objectives that the system should meet.

Means–ends analysis through AND-decomposition of problems into sub-problems has been standard practice in other disciplines for a long time. In artificial intelligence,

AND/OR problem reduction is a standard paradigm for problem solving (Nilsson, 1971). In human-computer interaction, AND-decomposition is a common mechanism for hierarchical modelling of user tasks (Rasmussen, 1986; Dix *et al.*, 2003). Goals are sometimes used in workflow models too (Ellis & Wainer, 1994). Business goals form an integral part of enterprise models (Loucopoulos & Kavakli, 1995; Boman *et al.*, 1997). In management science, Ishikawa diagrams (sometimes called fishbone diagrams) have a goal refinement structure for linking causes to problem symptoms in industrial processes (Kondo, 1994).

Goal orientation has proved also quite useful in other areas of software engineering. In software metrics, the Goal-Question-Metrics approach to measurement has proved quite effective for some time; it is goal driven and makes the distinction between intentional and operational levels (Basili *et al.*, 1988).

Surprisingly enough, goals have been, and still are, largely ignored in the literature on software modelling, specification and object-oriented development. Notable exceptions are the methodology described in Berzins and Luqi (1991), where goal hierarchies are used in the preliminary stages of software development, and the OBA approach, which uses goals as initial drivers of object-oriented analysis (Rubin & Goldberg, 1992). UML advocates sometimes confess the need for goals as higher-level abstractions not supported by the UML (Fowler 1997a, p. 45). Along these lines, practitioners have recommended the use of goal levels and goal decompositions as a means for building and structuring use cases to address them (Cockburn, 1997a, 1997b). The prominent tendency in software modelling research, however, has been to abstract programming constructs up to requirements level, rather than propagate requirements abstractions down to programming level (Mylopoulos *et al.*, 1999).

Yue was the first to argue that the integration of explicit goal representations provides criteria for requirements pertinence and completeness (Yue, 1987). The notion of operationalization of intentional specifications was introduced in Mostow (1985). The case for handling conflicts at goal level, rather than at the operational level, was convincingly made in Robinson (1989).

Feather developed a composite system design approach where local goals on individual agents are obtained from global system goals (called 'constraints' there). Agents have a choice of behaviour; an agent is responsible for a goal if restricting its individual behaviour is sufficient for ensuring goal satisfaction (Feather, 1987). In this framework, operational agent specifications and intentional system specifications can be analysed together to detect inconsistencies and resolve them by application of specification transformation operators (Fickas & Helm, 1992).

Two goal-oriented RE frameworks emerged independently in the early 1990s: KAOS (Dardenne *et al.*, 1991, 1993) and NFR/i* (Mylopoulos *et al.*, 1992; Yu, 1993). While both frameworks addressed common targets such as goal refinement and conflicts, there were differences and complementarities in focus. In KAOS, the emphasis was more on semi-formal and formal reasoning about behavioural goals for derivation of goal refinements, goal operationalizations, goal-based risk analysis and conflict

management (van Lamsweerde, 2000b). In NFR/i*, the emphasis was more on qualitative reasoning on soft goals for analysis of goal contributions, evaluation of alternative goal refinements and reasoning about goal dependencies among organizational agents (Chung *et al.*, 2000; Mylopoulos *et al.*, 2001). KAOS was more oriented towards goal satisfaction in the composite system design framework, whereas NFR/I* was more oriented towards goal satisficing.

Other goal-oriented RE frameworks emerged in the sequel (van Lamsweerde, 2001). In GBRAM, the focus was on guidelines and heuristics for identifying goals from raw material (Anton & Potts, 1998; Anton *et al.*, 2001). Most frameworks exploit the synergies between goals and scenarios for requirements elicitation and validation (Leite *et al.*, 1997; Anton & Potts, 1998; Haumer *et al.*, 1998; Rolland *et al.*, 1998; Sutcliffe, 1998; Kaindl, 2000). For embedded control software, the need for intentional specifications upwards of operational ones was recognized too (Leveson, 2000).

Some experience in using goal refinements as argumentation structures for safety-critical and security-critical systems is reported in McDermid (1994), Kelly and Weaver (2004) and Moore *et al.* (1999), respectively.

Goal types, and the distinction between *Achieve* and *Maintain* goals, were introduced in Dardenne *et al.* (1991, 1993). Their use is illustrated in, for example, Anton *et al.* (1994), van Lamsweerde *et al.* (1995). Goal categories were introduced in Mylopoulos *et al.* (1992) and Dardenne *et al.* (1993). A rich taxonomy of goal categories is proposed in Chung *et al.* (2000). Other goal classifications were proposed, in particular a classification according to desired system states, such as positive, negative, alternative, feedback or exception repair, and according to goal level, such as policy level, functional level or domain level (Sutcliffe & Maiden, 1993). Distinctions were also made between objective goals, which refer to objects in the system, and adverbial goals, which refer to ways of achieving objective goals (Anton *et al.*, 1994). Keywords were proposed to identify or represent goals in specific categories, for example qualitative verbs such as *Improve*, *Increase*, *Reduce*, *Make* and so on (Anton & Potts, 1998). Experience in using such categories for better requirements coverage is reported in Anton *et al.* (2001).

A great deal of work has been done on using goal specifications for a variety of early analyses, from informal to semi-formal to formal, in particular for:

- identifying goals from scenarios (Rolland *et al.*, 1998; van Lamsweerde & Willemet, 1998)
- checking goal refinements for completeness and realizability (Darimont & van Lamsweerde, 1996; Letier & van Lamsweerde, 2002a)
- deriving correct goal operationalizations (Letier & van Lamsweerde, 2002a)
- model checking early requirements (Fuxman *et al.*, 2001)
- identifying goal failure as source of new requirements (Potts, 1995; Cockburn, 1997b; van Lamsweerde & Letier, 1998; Anton & Potts, 1998; van Lamsweerde & Letier, 2000)

- managing conflicting goals (van Lamsweerde *et al.*, 1998; Robinson *et al.*, 2003)
- negotiating goal-based requirements (Boehm *et al.*, 1995; Robinson, 1990)
- reusing goal taxonomies and specifications (Massonet & van Lamsweerde, 1997; Anton & Potts, 1998; Sutcliffe & Maiden, 1998)
- elaborating or analysing security requirements (Anton *et al.*, 2002; Liu *et al.*, 2003; van Lamsweerde, 2004a)
- reasoning about partial goal satisfaction (Letier & van Lamsweerde, 2004)
- assessing or deriving software architectures from goal-based requirements (Gross & Yu, 2001; van Lamsweerde, 2003)

Goals are often defined in relation to the capabilities or skills of the agents responsible for their satisfaction (Dardenne *et al.*, 1993; Hui *et al.*, 2003). In their work on personal RE, Fickas and Sutcliffe have introduced the notion of *deferred goals* for dynamically customizable systems. Goals are ranked at RE time by levels of skills required for their satisfaction. The goals implemented in the first system versions are only those requiring elementary skills. As the system is running, higher-skill goals that were deferred are progressively incorporated into the system based on runtime monitoring of skill evolution (Sutcliffe *et al.*, 2006).

General overviews on goal-oriented RE can be found in van Lamsweerde (2000b, 2001). Case studies on goal-oriented analysis are presented in Anton *et al.* (1994), which illustrates the higher stability of goals with respect to operational descriptions; van Lamsweerde *et al.* (1995), and Dubois *et al.* (1998). Experience with goal-oriented RE in industrial projects is reported in van Lamsweerde (2004b).

Exercises

- Classify the following statements about a bank ATM system as *goal*, *requirement*, *expectation*, *domain property*, *hypothesis* or *definition*. Discuss the reasons for your classification. In particular, explain why some of these statements are not goals. For goal statements, identify the goal type and list system agents that might be involved in their satisfaction.
 - a. ATM is an acronym for automatic teller machine.
 - b. Ubiquitous cash services shall be provided.
 - c. ATM passwords are 4-digit numbers.
 - d. The likelihood of forgetting ATM cards at ATM counters shall be reduced as much as possible.

- e. Users will take their ATM card back if the card is returned with a beep before cash delivery.
- f. Any ATM card shall be kept after 3 wrong password entries.
- g. Cash is delivered when the user password and the requested amount are valid.
- h. Cash is delivered in multiples of banknote units.
- i. ATM vaults contain single-currency banknotes.
- j. Cash is taken by the cardholder when returned by the ATM.
- k. To get cash the user must (i) enter his or her ATM card, (ii) type the correct password, (iii) select some amount below the acceptable limit, (iv) get his or her card back, (v) take the cash.

Explain why the distinction between *goal*, *requirement*, *expectation*, *domain property*, *hypothesis* and *definition* is important for engineering the requirements for such a system.

- Identify the type and the category of each of the following goals for an ambulance dispatching system. Discuss the reasons for your classification:
 - a. The red button on the ambulance panel shall be pushed when the ambulance arrives at the incident scene.
 - b. The stress inherent to working conditions of ambulance crews should be reduced.
 - c. An ambulance is displayed by the software as being available if and only if it is actually ready for intervention at its ambulance station.
 - d. The nearest available ambulance shall be mobilized for the incident.
 - e. An ambulance may not be mobilized for an incident if it has not been allocated to that incident.
 - f. Upon mobilization the selected ambulance shall receive a mobilization order on its mobile terminal.
- Formulate each behavioural goal in the two previous exercises as instantiations of *Achieve* or *Maintain/Avoid* patterns.
- Explain the rationale for the following statements about a traffic light system by making their underlying goals explicit:
 - a. The light switch for pedestrians will be set to 'green' within x seconds after the pedestrian button has been pressed.
 - b. The light switch for cars will be set to 'red' at least y seconds before the light switch for pedestrians is set to 'green'.
- Consider the following goal in a university library system:

Book copies borrowed by students shall be returned within two weeks.

- a. Identify a combination of sub-goals that would jointly contribute to this goal.
 - b. Find another combination of contributing sub-goals as an alternative system option.
- Build a preliminary refinement graph for your favourite bank ATM system, showing lower-level and higher-level goals together with their respective contribution links. Consider functional *and* non-functional goals by checking for pertinent instances of the various goal categories in Figure 7.5. Identify chains of satisfaction arguments from your graph.