3

THE NFR FRAMEWORK: FOUNDATIONS — SOFTGOAL CONTRIBUTION GRAPH

??? Some part in Chapter 1 ???

Fundamental issues of software engineering include better software, lower cost, shorter production time, and making happier not only software engineers who produce software but ultimately customers for whom software is produced.

These issues of "better, cheaper, faster, happier" should be dealt with throughout software life-cycle, from the initial conception of a software system, all the way through planning, risk analysis, development, customer evaluation, release, operation and reengineering, to its graceful retirement.

These issues have been recognized as a vital determinant to the success of just about any software project, and led to proliferation of terms, be they organizational, managerial or technical. While also referred to as "desirable attributes", "design constraints", "system interface requirements", "user interface requirements", "hardware characteristics", etc., or more colloquially known as "–ilities" and "–ities", such issues are termed *non-functional requirements* in this book with the intent to distinguish them from functional requirements.

In this chapter, we present the NFR Framework which helps the developer to deal with non-functional requirements, namely, to express them explicitly, to deal with them systematically, and to use them to drive the entire software development process (or any phase of it) rationally.

Unlike goals in traditional problem-solving and planning frameworks, non-functional requirements can rarely be said to be "accomplished" or "satisfied" in a clear-cut sense. Instead, different design decisions contribute positively or negatively, and with a different degree, towards a particular non-functional requirement. Accordingly, we will speak of *softgoal* and softgoal *satisficing* to suggest that generated software is expected to satisfy within acceptable limits, rather than absolutely, non-functional requirements.

Formally, the NFR framework consists of five major components:

1. *softgoals* for representing non-functional requirements (NFRs) to be satisficed, design techniques which satisfice NFRs, and arguments which help justify development decisions;
2. *contributions* for stating relationships between softgoals, as well as between and among softgoals and contributions;
3. *methods* for refining softgoals or contributions into other softgoals or contributions;
4. *correlation rules* for inferring potential interactions, both positive and negative, between softgoals; and finally,
5. *evaluation procedure* which determines the degree, via a label, to which any given non-functional requirement is being addressed by a set of design decisions.

During the process of software development, softgoals and contributions, along with their labels, are organized into a *softgoal contribution graph*. It is a record of the developer's intentions, represented as softgoals, design alternatives, again represented as softgoals, design tradeoffs, represented as positive or negative contributions of design techniques towards the satisficing of NFRs, and design rationale, largely captured by arguments.

Of the remaining components, in contrast, methods and correlation rules help generate a softgoal contribution graph, by allowing for knowledge about ways of softgoal refinements and their interactions to be captured and reused. The evaluation procedure also helps generate a softgoal contribution graph, by determining, via label propagation, the impact of design decisions upon the satisficing of softgoals and contributions.

This chapter presents three of the five components: softgoals, contributions and the evaluation procedure. The next chapter will present the two remaining components, namely, methods and correlation rules.

The examples throughout this chapter continue with those in Chapter 2, and concentrate on accuracy and to a lesser extent operating cost and security requirements for credit card account management information systems.

3.1 SOFTGOALS AND CONTRIBUTIONS

??? Some in Chapter 1 ???

Non-functional requirements are often *subjective*. in the sense they can be viewed and evaluated differently by different (groups) of people. Non-functional requirements are also often *relative*, in the sense that, if there are ways to achieve them, there can be even better ways to achieve them. Furthermore, non-functional requirements often tend to be *interacting*, in that attempts to achieve one of them can hurt, or enhance, one or several other ones.

To explicate these tendencies, let us consider a simple scenario.

Suppose you want to buy a cup of coffee, say on a very cold day. You may or may not be able to buy a cup of coffee, depending on whether you have enough money, you can find a coffee shop, you have the time and strength to walk over there, etc. Now suppose you want to buy a *good* cup of coffee. You can go to the same coffee shop and buy the same kind of coffee. But is it good? Perhaps, it is to you as it is hot and strong, and you like it that way especially on a very cold day and when you are sleepy. But then it may not be good enough for your friend sitting next to you, perhaps because of the flavor or because it's too strong for her as she had another cup of coffee just half and hour earlier. Now, without even getting your friend involved, would the same cup of coffee still be good for you, say on a very hot day? It may still be if you like it that way also on such a sizzling day, but then it may not. If not, you might try to get a *better* cup of coffee, *but* by paying more or by walking a distance to another coffee shop to get one at the same price.

The notion of "goal" has been used in a variety of settings. One prevalent use of this notion is in traditional problem-solving and planning frameworks. Here, goals (e.g., "buy a cup of coffee") are considered as obligations which should be absolutely accomplished. If goals can indeed be absolutely accomplished (e.g., "a cup of coffee can be bought"), they are said to be solvable or satisfiable; if not, they are said to be unsolvable or unsatisfiable. Put differently, a goal should be either satisfied or unsatisfied, and nothing else, in every possible imaginable situation (e.g., "The same cup of coffee is good on a cold day, on a hot day, after five cups of coffee in a row, to you, to your friend, etc."). With this two-valued logic, then, a goal evaluates to true, if it is satisfied, and false, otherwise. Absolutely nothing in-between.

In this context, a set of "subgoals" are introduced to satisfy a given goal, where the relationship between the subgoals and goal is either *AND/OR*. When the relationship is AND, the goal is satisfied if all of its subgoals are; when the relationship is OR, the goal is satisfied if any of its subgoals is. This process of "subgoaling" continues until no goal, or subgoal, can be further reduced into more detailed ones.

Now, what is the appropriate notion of "goal" when we want to deal with NFRs? We need something different than the notion of "goal" in traditional problem-solving and planning frameworks. This is because non-functional requirements (e.g., "buy a *good* cup of coffee") can rarely be said to be "accomplished" or "satisfied" in a clear-cut sense. Due to their subjective tendency, solutions to non-functional requirements may be considered accomplished by some people, while not by some other people (e.g., "The same cup of coffee is good for you but nor for your friend."). Due to their relative tendency, non-functional requirements may be poorly accomplished, accomplished, well accomplished, etc. (e.g., "a cup of coffee", "a good cup of coffee", "an excellent cup of coffee", etc."). Furthermore, due to their interacting nature, design decisions can contribute positively or negatively towards a particular non-functional requirement (e.g., "a better cup of coffee by paying more or by walking a distance to another coffee shop to get one at the same price").

In this discussion, an interesting observation can be made about the way non-functional requirements are considered fulfilled or not fulfilled. It involves some kind of qualifications (e.g., "you like it (a cup of coffee) that way especially on a very cold day and when you are sleepy.", "it may not be good enough for your friend sitting next to you, perhaps because of the flavour or because it's too strong as she had another cup of coffee just half an hour earlier."). Such qualifications may in turn be explained by other qualifications (e.g., "You are sleepy because you stayed up all night.", "Several cups of coffee in a row is bad.").

This kind of iterative qualifications or reasoning, called *dialectical style of reasoning*, allows one to state arguments for, or against other statements. In contrast to reasoning with dualism (e.g., true and false), dialectical style of reasoning accommodates more easily the subjective, relative and interacting nature of non-functional requirements.

In order to accommodate these tendencies, we forgo that part of the notion of "goal" on absolutivism, but retain only the objective (or purpose) sense of it. Accordingly, non-functional requirements are treated as *softgoals*.

Now what is the appropriate notion of "relationship between softgoals"? Here, the strict AND/OR goal reduction does not quite work, as softgoals contribute only partially, and sometimes negatively, towards the achievement of other softgoals. Accordingly, we use the concept of *contributions* to reflect softgoal *satisficing* which conveys the semantics that generated software is expected to satisfy non-functional requirements with their subjectivism, relativism and interact-ivism taken into consideration.

With the above semantic distinctions in place, then, we can use the term "goal" rather freely, for example, to refer to non-functional requirements.

In dealing with non-functional requirements, expressing them as goals amounts to putting concerns about NFRs foremost in the developer's mind. Using the NFR framework, then, the developer can strive for NFRs systematically, rather than in an *ad hoc* manner, and at the same time use them to drive the overall development process, instead of merely evaluating how well a given product meets them.

This way, consideration of design alternatives, analysis of design tradeoffs and rationalization of design decisions are all carried out in relation to the stated goals.

3.2 TYPES OF GOALS

As hinted at in the above discussion, the space of softgoals includes three mutually exclusive classes, namely, *non-functional requirements* (e.g., "buy a good cup of coffee"), *satisficing techniques* (e.g., "paying more"), and *arguments* (e.g., "You are sleepy because you stayed up all night.").

In general, each goal will have an associated *sort* and zero or more *parameters* whose nature depends on the softgoal type. In the phrase "the security of an airplane", for example, "security" is the sort and "an airplane" is the parameter. When the sort changes (e.g., "the cost of an airplane"), so does the meaning of the phrase. Similarly, when the parameter changes, so does the meaning. After all, the security of an airplane is quite different from the security of a house, the security of a person, etc.

In the context of a credit card management system, an operating cost requirement, an important type of non-functional requirements in software engineering, might have as sort operating cost and as parameter a desired upper bound (say $100 000) on the annual operating costs of the system under development. This non-functional requirement as goals (semantically softgoals) might be represented by

Softgoal our-operating-cost-limit
    sort OperatingCost
    parameter $100 000,

or in a more compact form, without the keywords sort and parameter but with the parameter surrounded by square brackets ([ ])

Softgoal our-operating-cost-limit OperatingCost [$1 000 000].

The keyword Softgoal or a softgoal identifier, such as our-operating-cost-limit, can be omitted. Thus, with these omissions, the above can be written in an even more compact form:

OperatingCost [$1 000 000].

Throughout the book, a compact form of goal expression will be used mostly in text and figures. Once a softgoal is introduced with its identifier, the identifier alone can be used later on to designate the softgoal.

In addition to a sort and a list of parameters, each softgoal has a label which indicates the degree to which the softgoal is satisficed (details of this in Section 3.4). Each softgoal can also have other attributes, such as criticality, author, and time of creation.

For example, the aforementioned operating cost goal might be extended to:

Softgoal our-operating-cost-limit

sort OperatingCost

parameter $100 000

label S

criticality non-critical

author Jane Doe

creation time 08/15/97

With the addition of criticality alone, our-operating-cost-limit can be expressed in a compact form:

our-operating-cost[$100 000]:non-critical

Depending on the the value of the sort, the keyword Softgoal can be specialized into one of the three more specific softgoal types, namely, NFR, SatisficingTechnique (or SatisficingTechnique) or Argument, which will refer respectively to the set of all possible softgoals, non-functional requirements as goals, satisficing techniques as goals, and arguments as goals.

As will be detailed in what follows, non-functional requirements act as overall constraints on the system, and satisficed by satisficing techniques which involve design or implementation components. Unlike non-functional requirements or satisficing techniques, arguments provide rationale for development decisions.

As a consequence, only satisficing techniques appear in one form or another in the target design or implementation, whereas neither non-functional requirements nor arguments do. However, all the three types of softgoals appear in a softgoal contribution graph.

3.2.1 NFR Goals

These softgoals range over the different categories of non-functional requirements about better software, lower cost, shorter production time, and happier software engineers and customers. The sorts for NFR Goals include those for:

- *better software:* accuracy, adaptability, completeness, comprehensibility, configurability, flexibility, inter-operability, learnability, maintainability, modularity, performance, portability, reliability, reusability, safety, security, testability, traceability, user-friendliness, usability, etc.;
- *cheaper software:* development cost, operating cost, maintenance cost, reengineering cost, retirement cost, hardware cost, communication cost, etc.;
- *faster production:* project stability, planning time, development time, testing time, customer evaluation time, etc.; and
- *happier people:* employee loyalty, customer loyalty, supportability, trainability, etc.

Of course, one sort can be related to another sort, both within the same category and across different categories. A comprehensive, but not necessarily complete, list of possible sorts was also shown earlier in Table 1.2 in Chapter 1.

As specializations of softgoals, NFR goals are represented as such. Suppose, for our credit card account management system, that it is expected of the system under development to maintain accurately account data. Such a softgoal might be represented by

NFR accuracy-of-account-information
    sort Accuracy
    parameter Account,

or in a more compact form

Accuracy[Account].

In a NFR goal expression, its parameter can refer to more specific information than, for example, Account. The following shows how Account might be replaced:

Accuracy[GoldAccount.highSpending]

where the parameter GoldAccount.highSpending evaluates to the set of all spendings over a pre-set amount associated with the data class Gold Account, one type of Account. The interpretation of this softgoal is that values of all the high spending data ought to be maintained accurately in the system's database.

As another example, it may be expected that the system under development make minimal demands on manpower with an upper bound on the cost. This can also be treated as an operating cost requirement, but with more than one parameter. Since there are several contributing factors to operating costs (manpower, maintenance, etc.), this requirement might have manpower as one of its parameters and $100 000 as another and be represented as:

OperatingCost[manpower, $100 000].

NFR Goals, such as Accuracy[Account] and OperatingCost[manpower, $100 000], act as global constraints on the system which need to be satisficed by satisficing techniques, i.e., solutions involving design or implementation components, or operations performed on the components. Figure 3.1 illustrates how NFRs as goals are depicted pictorially in a softgoal contribution graph. In the graphical notation, NFRs as goals are denoted by by thin circles.

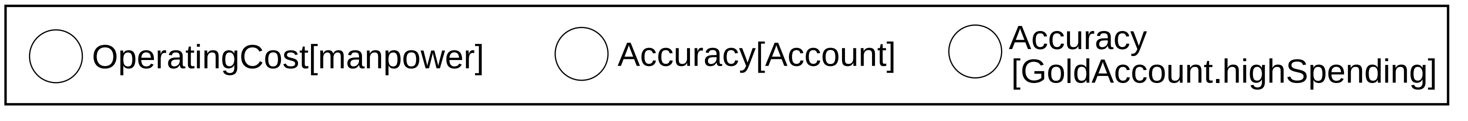

Figure 3.1 Pictorial representation of NFRs as goals.

3.2.2 Satisficing Technique Goals

When the developer has satisfactorily clarified the initial NFRs, she can use certain *techniques* to satisfice one or more non-functional requirements. Such techniques range over different categories of design techniques, during the design phase, or implementation techniques, during the implementation phase.

At any phase of system development, there may be more than one way to achieve a technique. Furthermore, each such alternative usually make only a partial contribution towards achieving the technique in question. Thus, like

NFRs, these techniques are also treated as goals (or more formally, softgoals) to satisfice.

As such, they are also sorted and parameterized, where the parameters associated with each sort depend on the nature of the corresponding satisficing technique.

For instance, one way to satisfice the accuracy goal mentioned earlier might be to validate all account data entered into the system. This can be represented as a satisficing technique:

SatisficingTechnique validation-of-account-information
  sort Validation
  parameter GoldAccount.highSpending

or in a more compact form

Validation[GoldAccount.highSpending],

where Validation is the softgoal sort and GoldAccount.highSpending is as before. This softgoal, in turn, might be refined into another satisficing technique:

ValidatedBy[GoldAccount.highSpending, class-I-secretary],

representing the situation that class I secretaries will be doing the validation. The softgoal sort is ValidatedBy, while the parameters consist of GoldAccount.highSpending and class-I-secretary.

This goal might be further refined, this time in terms of several satisficing techniques:

Available[GoldAccount.highSpending],
Available[policy-on-spending-pattern],
Available[class-I-secretary],

representing the requirements that, in order to perform the validation task, information about GoldAccount.highSpending, certain policy which states predetermined standards on permissible spending patterns, and class I secretaries should be available.

Figure 3.2 illustrates design techniques as goals are depicted pictorially in a softgoal contribution graph. In a graphical notation, satisficing techniques

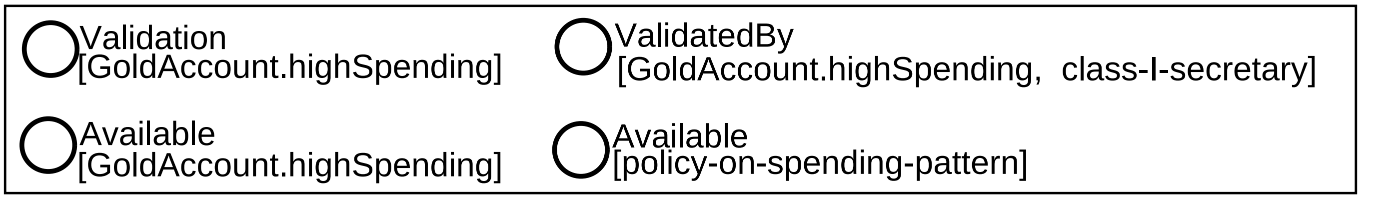

Figure 3.2 Pictorial representation of satisficing techniques as goals.

are denoted by thick circles.

3.2.3 Argument Goals

Unlike NFRs or satisficing techniques, arguments make it possible for domain characteristics to be considered and properly reflected into the decision making. They serve as justification in support or denial of the way softgoals are prioritized, the way softgoals are refined and the way target components are selected.

During this justification process, arguments may be considered weak, counter-intuitive or even wrong. For this reason, even arguments themselves sometimes need to be supported or denied, hence acting as evidences or counter-evidences for other arguments. Thus, like NFRs and satisficing techniques, arguments are also treated as goals.

This way, arguments provide rationale for development decisions, hence facilitating later review, justification and evolution of the system, as well as enhancing traceability.

These softgoals always have the sort Claim, with subsorts FormalClaim and InformalClaim, representing respectively formally or informally stated evidence or counter-evidence for other softgoals or contributions.

Consider:

Argument validation-policy
    sort FormalClaim
    parameter exists (*employee status*) where
        ValidatedBy[GoldAccount.highSpending, *employee*] and
        EmployeeStatus (*employee, status*) and
        HigherJobClassification (*status*, class-II-secretary)

This argument supports the refinement from the softgoal of validating high spendings data to the one assigning a class I secretary (class-I-secretary) to the task (instead of a class II secretary) or below), by claiming that the validation should be performed by an employee whose job is classified as higher than class II secretary.

In contrast,

Argument validation-policy-justification
    sort InformalClaim
    parameter Rigorous examination is recommended for high spendings
        by employees.

is an informally-stated argument supporting the previous argument by pointing out why accounts with high spendings should be validated by an employee whose job is classified as higher than class II secretary.

Figure 3.3 illustrates design techniques as goals are depicted pictorially in a softgoal contribution graph. In a graphical notation, arguments are denoted by by dotted circles.

Reiterating the presentation in this section, the three types of concepts, namely, NFRs, satisficing techniques and arguments are all treated as goals uniformly. This treatment helps the developer deal with design constraints, alternatives, tradeoffs, decisions and rationale systematically, rather than merely evaluate how good the generated product is.

Figure 3.4 illustrates how all the three types of softgoals are depicted pictorially in a softgoal contribution graph. As before, softgoals are denoted by

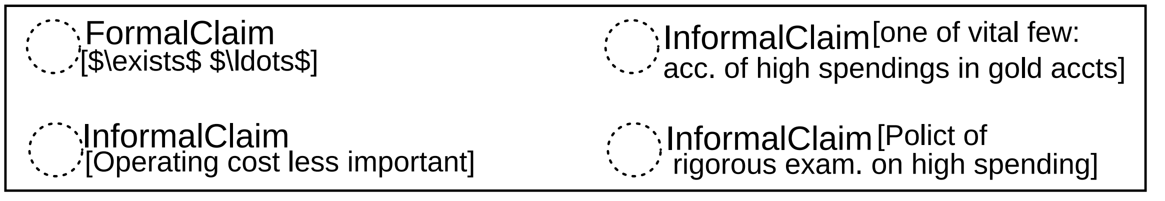

Figure 3.3 Pictorial representation of arguments as goals.

circles: NFR goals by thin circles, satisficing techniques by thick circles, and arguments by dotted ones.

Unclear in the figure, however, are relationships between goals, namely to what extent offspring goals contribute to the satisficing of parent goal. This is the topic of the next section.

3.3 TYPES OF CONTRIBUTIONS

As illustrated in Chapter 2, design proceeds by refining one or more times each softgoal, the parent, into a set of other softgoals, the offspring. In such a refinement, the offspring can contribute partially, and positively or negatively, towards the satisficing of the parent, instead of (fully and only positively) satisfying the parent.

Recall that we speak of softgoal *satisficing* to suggest that generated software is expected to satisfy within acceptable limits, rather than absolutely, non-functional requirements. Accordingly, in the NFR framework, there can be several different types of relationships or *contribution types* describing how the satisficing of the offspring (or failure thereof) relates to the satisficing of the parent.

A *contribution* then is the relationship, between the parent and its offspring which results from such a refinement, plus its contribution type.

For example, AND is one of the contribution types which roughly means that if the offspring are all satisficed, so will be their parent. A more formal definition will be described shortly. Now if Accuracy[Account] has, via AND contribution, three offspring Accuracy[RegularAccount], Accuracy[GoldAccount]

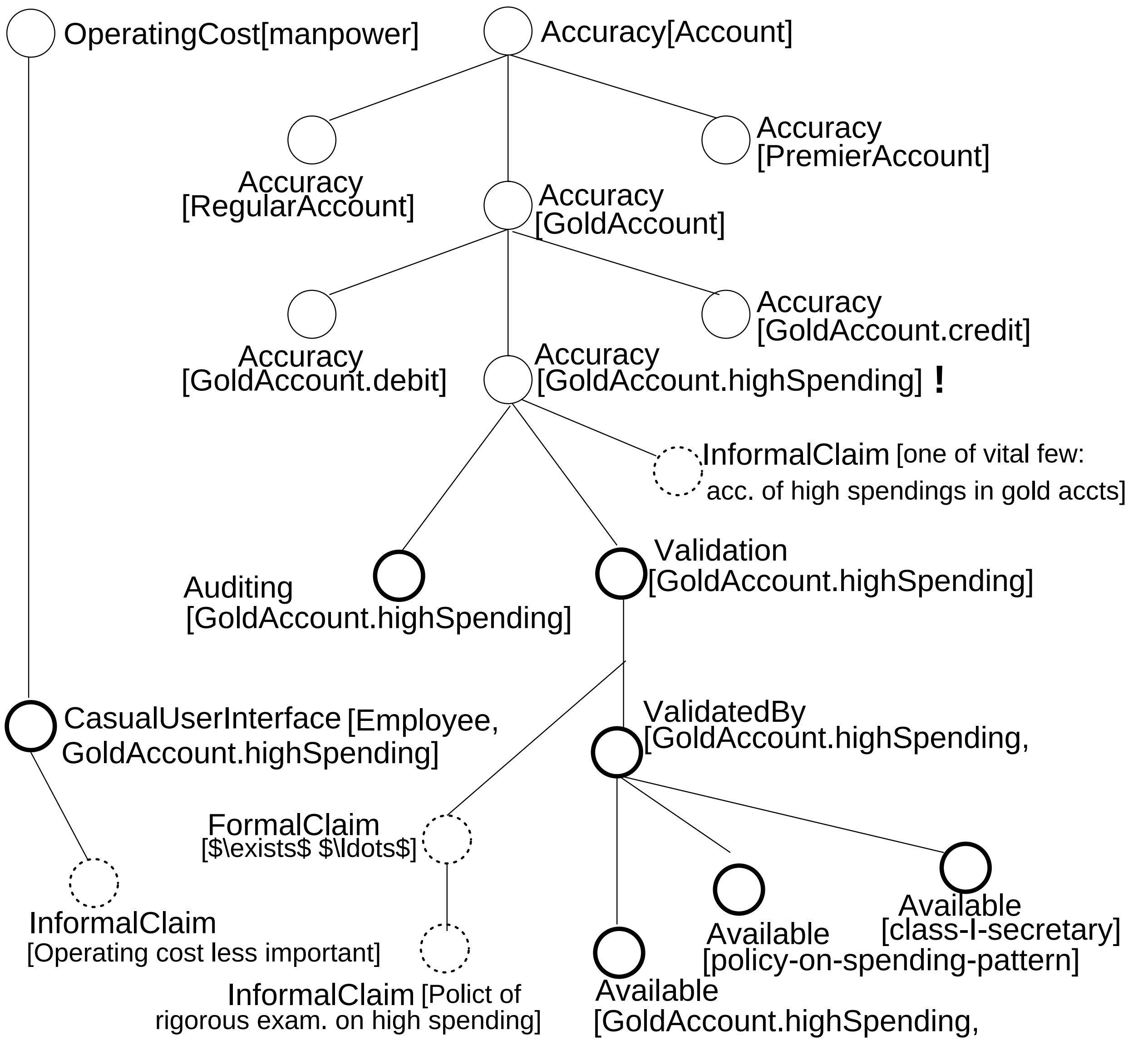

Figure 3.4 Pictorial representation of the three types of softgoals.

and Accuracy[PremierAccount], the contributory relationship between the parent and its offspring is expressed as:

ContributionRelationship accuracy-of-account-and-three-offspring
    parent NFR Accuracy[Account]
    offspring NFR Accuracy[RegularAccount],
        NFR Accuracy[GoldAccount],
        NFR Accuracy[PremierAccount]
    contribution type AND

Here NFR indicates the type of the parent and offspring. In general the type can be Softgoal or one of the three types of softgoals (i.e., NFR, SatisficingTechnique, and Argument) or even Contribution. It can also be omitted, if no ambiguities arise.

Figure 3.5 illustrates how the above contribution is depicted pictorially in a softgoal contribution graph. As before, circles denote softgoals. In the

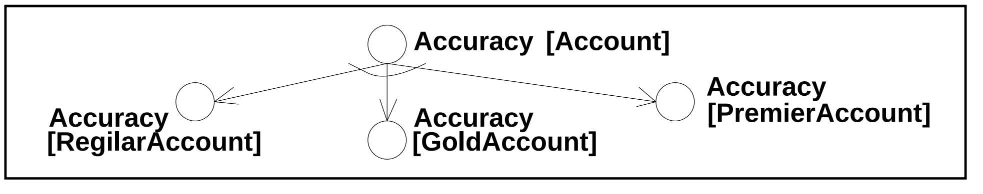

Figure 3.5 Pictorial representation of the contribution accuracy-of-account-and-three-offspring.

figure, four non-functional requirements are shown as goals. Here, offspring are shown underneath the parent softgoal. AND contribution is shown by an arc connecting three lines stemming from the parent and ending at its offspring.

As with softgoals, contributions can also be in a more compact form such as:

AND(Accuracy[Account], {Accuracy[RegularAccount],
Accuracy[GoldAccount],
Accuracy[PremierAccount]})

This contribution can also be expressed, yet in another form, in a more programming language-like style:

$$
\begin{aligned}
< \text{Accuracy[Account]}, \quad & \text{Accuracy[RegularAccount]}\ \mathsf{AND} \\
& \text{Accuracy[GoldAccount]}\ \mathsf{AND} \\
& \text{Accuracy[PremierAccount]} >
\end{aligned}
$$

As with a softgoal, in addition to the parent, offspring and contribution type, each contribution has a label attribute, possibly with other attributes such as label, criticality, author, and time of creation.

A contribution may, however, not be agreeable among different people, for example, concerning its contribution type such as AND and other types of contributions (+/– or partial/full satisficing). Thus, just like softgoals, contributions too need to be satisficed either through a formal refinement process or through arguments provided by the designer.

In order to give a formal definition of contribution types, let us now introduce the notions of *satisficed*, *denied*, *satisficeable* and *deniable*. The first notion, *satisficed*, refers to softgoals or contributions that are considered satisfactory by the developer. The second notion, *denied*, refers to softgoals or contributions that are considered dissatisfactory ("unsolvable" in problem-solving terminology [Nilsson71]). What about satisficeable and deniable?

Sometimes a softgoal or contribution will be found to be satisficed — thanks to one refinement — and, at the same time, denied — due to another. For instance, the accuracy goal for high spendings data may be satisficed — thanks to a validation procedure which significantly enhances the accuracy of such data, although a validation procedure cannot be assumed perfect. However, the same goal can also be denied at the same time — due to a casual user interface which permits general access to this information.

This is certainly a conflicting situation, and the two notions of satisficed and denied are not sufficient to deal with such a situation. Instead, we need to distinguish a softgoal or contribution being satisficed or denied from a softgoal or contribution being *potentially* satisficeable or deniable. Accordingly, we introduce two more notions, *satisficeable* and *deniable*, to deal with the latter case.

With the above four notions in place, the set of contribution types are now presented below. Appendix B provides axioms which formalize the semantics of each contribution type.

The first two contribution types, AND and OR, relate a parent to a group of offspring:

AND$(parent, offspring)$: if all the offspring are satisficed
when the contribution itself is satisficed,
the parent is satisficeable.

if any of the offspring is denied
when the contribution itself is satisficed,
the parent is deniable.

In other words, AND (*parent*, *offspring*) implies that the parent is satisfactory if all the offspring are satisfactory and the contributory relationship between the parent and the offspring is no other than AND. For example, with AND (Accuracy[Account], Accuracy[RegularAccount], Accuracy[GoldAccount], Accuracy[PremierAccount]), Accuracy[Account] will be satisficeable if all the three offspring are satisficed and the AND contribution itself is satisficed.

As for OR (*parent*, *offspring*), the parent is satisfactory if any of the offspring are satisfactory and the contributory relationship between the parent and the offspring is no other than OR.

OR$(parent, offspring)$: if any of the offspring is satisficed
when the contribution itself is satisficed,
the parent is satisficeable.

if any of the offspring is denied
when the contribution itself is satisficed,
the parent is deniable.

Figure 3.6 illustrates how OR is depicted pictorially in a softgoal contribution graph. It also shows other contribution types to be described below. OR

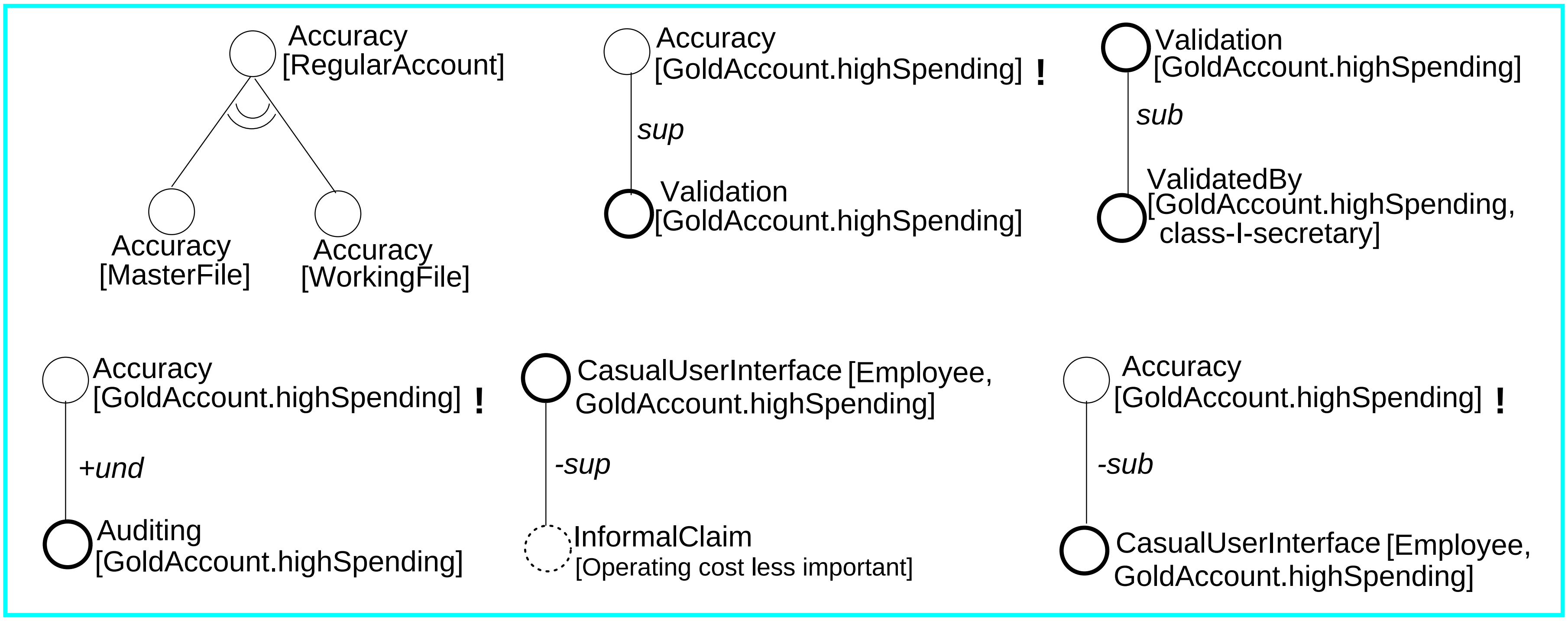

Figure 3.6 Pictorial representation of several contribution types.

contribution is shown by a double arcs connecting three lines stemming from the parent and ending at its offspring. A softgoal contribution graph, which is the source of most of the contributions types, is given at the end of this section.

While AND and OR relate a group of softgoals or contributions to a softgoal or contribution, `sup` and `sub` do between one softgoal or contribution and another.

$\mathsf{sup}(parent, offspring)$:	if the offspring is satisficed when the contribution itself is satisficed, the parent is satisficeable.
$\mathsf{sub}(parent, offspring)$:	if the offspring is denied when the contribution itself is satisficed, the parent is deniable.

The contribution type sub is intended to convey the sense that the offspring contributes partially to the satisficing of the parent. In other words, if P_1 is a sub(softgoal or contribution) of P_0 then there are softgoals or contributions $P_2, \ldots, P_n$ which cannot achieve the satisficing of P_0 without the contribution of P_1.

For example, with

$$\begin{array}{r}\mathsf{sub}(\text{Validation[GoldAccount.highSpending]},\\ \text{ValidatedBy[GoldAccount.highSpending, class} - \text{I} - \text{secretary]}),\end{array}$$

ValidatedBy[GoldAccount.highSpending, class-I-secretary] contributes partially to the satisficing of Validation[GoldAccount.highSpending]. In order to convert sub to sup, more experts in gold accounts and spending patterns than class I secretaries might also need to be brought in to do the validation.

Two additional contribution types are introduced to represent negative influences of one softgoal or contribution on another.

$-\mathsf{sup}(parent, offspring)$: if the offspring is satisficed when the contribution itself is satisficed, the parent is deniable.

$-\mathsf{sub}(parent, offspring)$: if the offspring is denied when the contribution itself is satisficed, the parent is satisficeable.

In words, if G_1 is a -sub(softgoal or contribution) of G_0 then denial of G_1 leads to the satisficing of G_0 and satisficing of G_1 contributes to the denial of G_0.

Finally, it is useful to define the eql (equivalent) contribution type in terms of the contribution types introduced here:

$$\begin{aligned}\mathsf{eql}(parent, offspring) \equiv\ & \mathsf{sup}(parent, offspring)\ and\\ & \mathsf{sup}(parent, offspring)\ and\\ & \mathsf{sub}(parent, offspring)\ and\\ & \mathsf{sub}(parent, offspring)\end{aligned}$$

At times, it may be hard to determine *a priori* the logical relationship between a set of offspring and their parent softgoal without further expansion of the softgoal contribution graph. For example, the designer may see that a certain hiring policy for technical staff is relevant, without being sure of its impact on a particular softgoal, say, in justifying the assignment of a class II secretary to the task of validating account data. This situation is accommodated through three variations of the undetermined contribution type:

und$(parent, offspring)$ indicates the possible presence of positive or negative influence between the parent and the offspring.

Likewise, +und and -und indicate respectively possible positive or negative influence between two softgoals or contributions.

Figure 3.7 illustrates what contribution types are there between and among goals and contributions, which were missing from the earlier Figure 3.4.

Figure 3.7 is still incomplete as it is missing labels of softgoals and contributions which indicate the degree to which they are satisficed. This is the topic of the next section.

3.4 THE EVALUATION PROCEDURE

Presented up to this point are two of the five main concepts of the NFR framework: softgoals and contributions. These concepts enable the developer to treat non-functional requirements as goals (more formally softgoals) and then satisfice them by repeatedly refining them into more specific ones. *But how does the developer determine whether the goals are satisficed?*

Another concept of the NFR framework is the evaluation procedure which determines the degree, via a label, to which any given non-functional requirement is being addressed by a set of design decisions. Given a partially constructed softgoal contribution graph, the evaluation procedure determines the status of each softgoal or contribution on the softgoal contribution graph through the assignment of a label.

The evaluation procedure is useful in selecting among alternatives. In the presence of competing alternatives, the developer can use the procedure to see what impact a particular selection has on the satisficing of her goals. If one

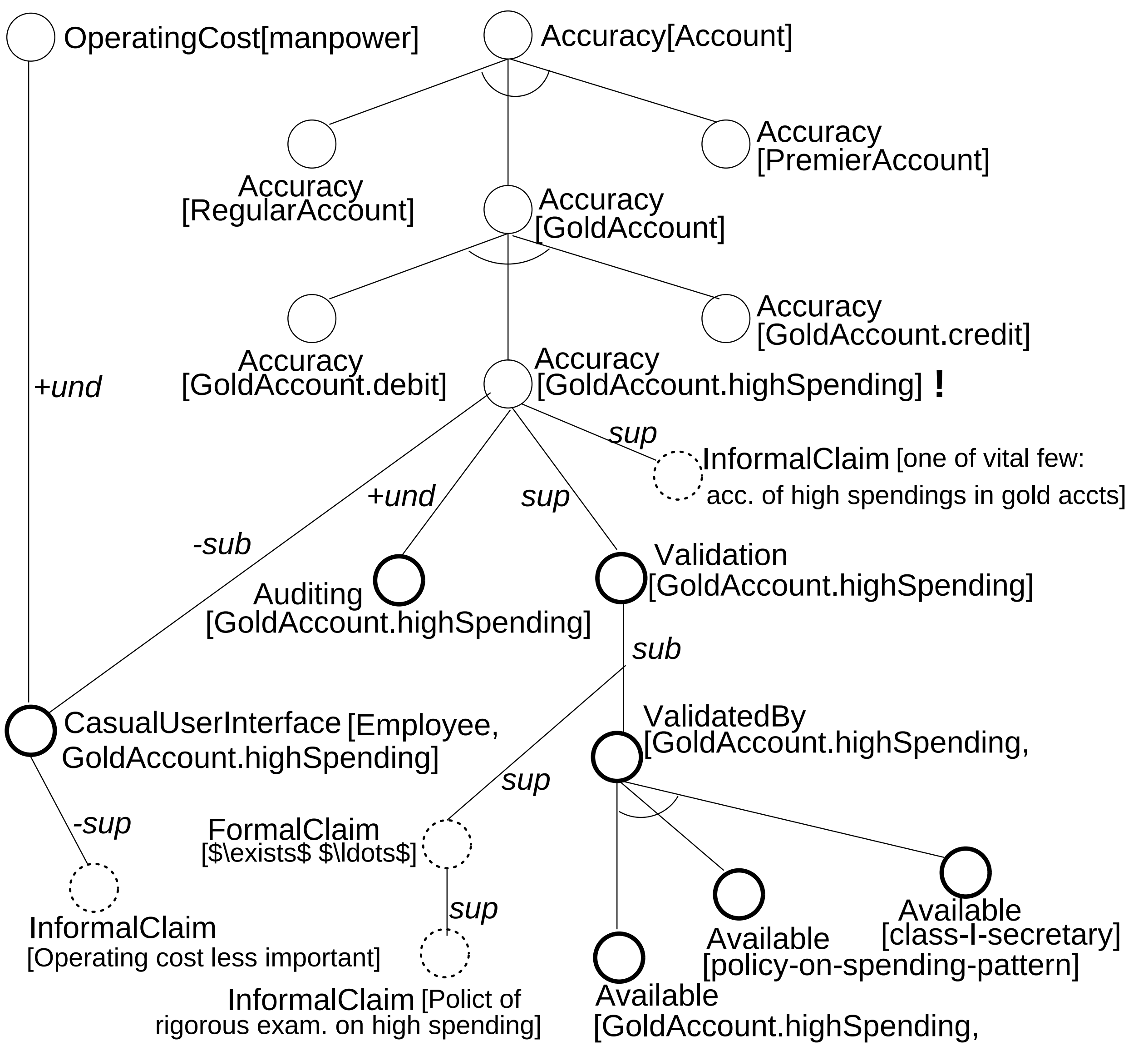

Figure 3.7 Pictorial representation of a softgoal contribution graph with contribution types.

selection hurts some important goals, she can simply deny it, choose another and again use the procedure to see the impact of the new choice. By repeating this process, she can aim at making a choice which yields the most benefit with minimum sacrifices.

As a starting point for this, the evaluation procedure needs a set of labels. In accordance with the description in the previous section, a softgoal or contribution of the graph is labelled

- *satisficed* (denoted by S) if it is satisficeable and not deniable;
- *denied* (denoted by D) if it is deniable but not satisficeable;
- *conflicting* (denoted by C) if it is both satisficeable and deniable; and
- *undetermined* (denoted by U) if it is neither.

The U label, in particular, is intended to represent situations where either there is both positive and negative support, albeit inconclusive, for a given softgoal or contribution, or there is neither. Upon its introduction, a softgoal or contribution is assigned U by default.

When each softgoal and contribution in the softgoal contribution graph is assigned a label (including U), the evaluation algorithm is activated and propagates labels from offspring to parents. It consists of two basic steps. For each softgoal or contribution on a given softgoal contribution graph, the algorithm first computes the individual effect of each satisficed outgoing contribution. Secondly, the individual effects of all outgoing contributions are combined into a single label taking one of the four possible label values.

As described in the beginning of Section 2, the NFR framework adopts dialectical style of reasoning in which arguments are made in support or denial of why non-functional requirements are considered fulfilled. This leads to the premise built into the framework that only some of the relevant knowledge is formally represented, the rest remaining with designers, during the process of dealing with non-functional requirements.

Given this open-ended nature of the argumentation process, the NFR framework calls for an *interactive* evaluation procedure where the designer may be asked to step in and determine the appropriate label for a particular softgoal or contribution having supporting but inconclusive evidence.

$label_{offspring}$	contribution type				
	sub	sup	−sub	−sup	und
S	U^+	S	U^-	D	U
D	U^-	U^-	U^+	U^+	U
C	?	?	?	?	U
U	U	U	U	U	U

Table 3.1 The *individual effect* of offspring label upon its parent label.

For this reason, the labels characterizing the influence of one set of offspring towards a parent include S, D, C, and U, as mentioned before, but also

- U^+ representing inconclusive positive support for a parent,
- U^- representing inconclusive negative support for a parent, and
- ? indicates a situation where the designer is to determine the label that characterizes the contribution of a softgoal or contribution towards another.

Note that the labels U^-, U^+ and ? are introduced by the first step of the evaluation algorithm and are eliminated by the second when the set of all contributions from all outgoing contributions associated with a given softgoal or contribution are combined into a single label, S, D, C or U.

The First Step. Table 3.1 shows the propagation rules along different contribution types (always from offspring to parent). According to these rules,

- sup propagates S,
- sub propagates D,
- −sup inverts an S label into a D, and
- −sub inverts a D label into a S one.

The propagation rules for AND and OR contributions are based on the ordering of labels

$$\mathsf{S} \geq \mathsf{U}, \mathsf{C} \geq \mathsf{D}$$

and is defined as follows:

If $\quad \mathsf{AND}(G_0, \{G_1, G_2, \ldots, G_n\}) \quad$ then $label(G_0) = \min_i(label(G_i))$

If $\quad \mathsf{OR}(G_0, \{G_1, G_2, \ldots, G_n\}) \quad$ then $label(G_0) = \max_i(label(G_i))$

Figure 3.8 illustrates pictorially how the first step of the evaluation procedure works on some selected contribution types and labels.

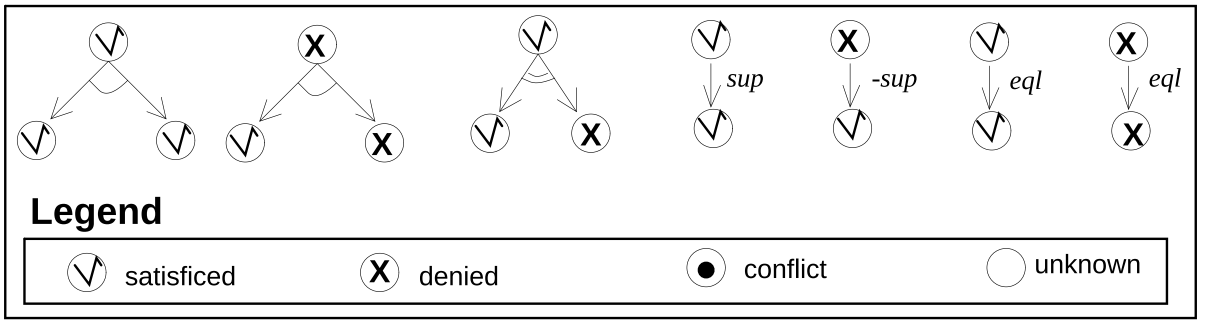

Figure 3.8 Pictorial representation of label propagation for selected contribution types and labels during the first step.

The Second Step. Once all contributed labels have been collected for a given softgoal or contribution, the second step of the evaluation procedure combines them into a single label. Assuming that $\mathtt{L}$ is the bag[1] contributed to a given softgoal or contribution, consisting of labels from the set $\{\mathsf{S}, \mathsf{D}, \mathsf{C}, \mathsf{U}, \mathsf{U}^-, \mathsf{U}^+\}$, the U^+ and U^- labels are first combined by the designer into one or more S, D, C, and U labels. The resulting set of labels is then combined into a single one, by choosing the minimal element, $\min_{l \in \mathtt{L}}(l)$, and assuming a label ordering

$$\mathsf{S}, \mathsf{D} \geq \mathsf{U} \geq \mathsf{C}.$$

Figure 3.9 illustrates pictorially how the second step of the evaluation procedure works on some selected contribution types and labels. Note that the

[1] $\mathtt{L}$ is a bag because duplicate labels are useful; for instance several positive supporting contributions indicated by several U^+'s may be combined into an S label by the designer.

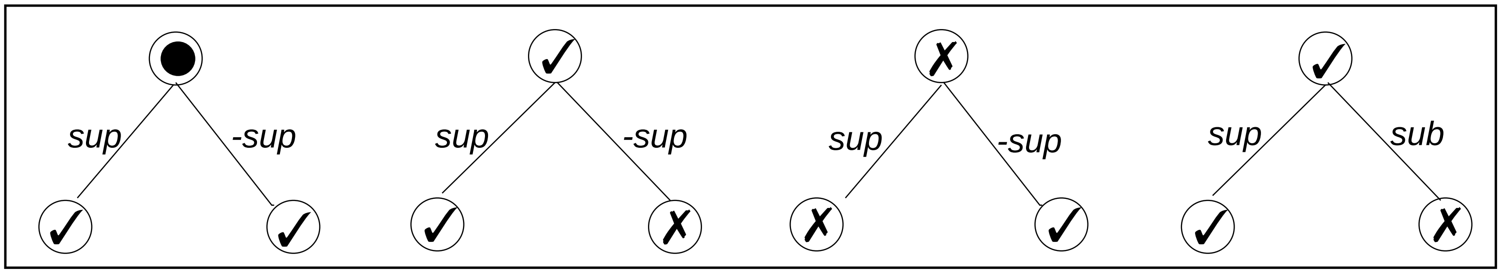

Figure 3.9 Pictorial representation of label propagation for selected contribution types and labels during the second step.

last case in Figure 3.9 needs the designer's input, along the `sub` contribution. Assuming that the designer has indicated that the D label has insignificant influence on the parent goal, the label at the parent is S.

Figure 3.10 illustrates the softgoal contribution graph that might be generated by the simple example we have been introducing piecemeal.

In the figure, the label of Accuracy[GoldAccount.highSpending] could have been assigned a C label, had the designer assigned a S to CasualUserInterface[Employee, GoldAccount.highSpending] and also indicated that this label should result in a D along the `−sub` contribution between the two goals.

3.5 DISCUSSION

3.5.1 Key Points

Fundamental issues of software engineering include quality, production time, cost, and ultimately making both software engineers and customers happy. These issues of "better, cheaper, faster, happier", or non-functional requirements, should be dealt with from the initial conception of a software system, all the way through planning, risk analysis, development, customer evaluation, release, operation and reengineering, to its graceful retirement.

The framework offers five components which are essential for dealing with non-functional requirements: *softgoals*, *contributions the evaluation procedure*,

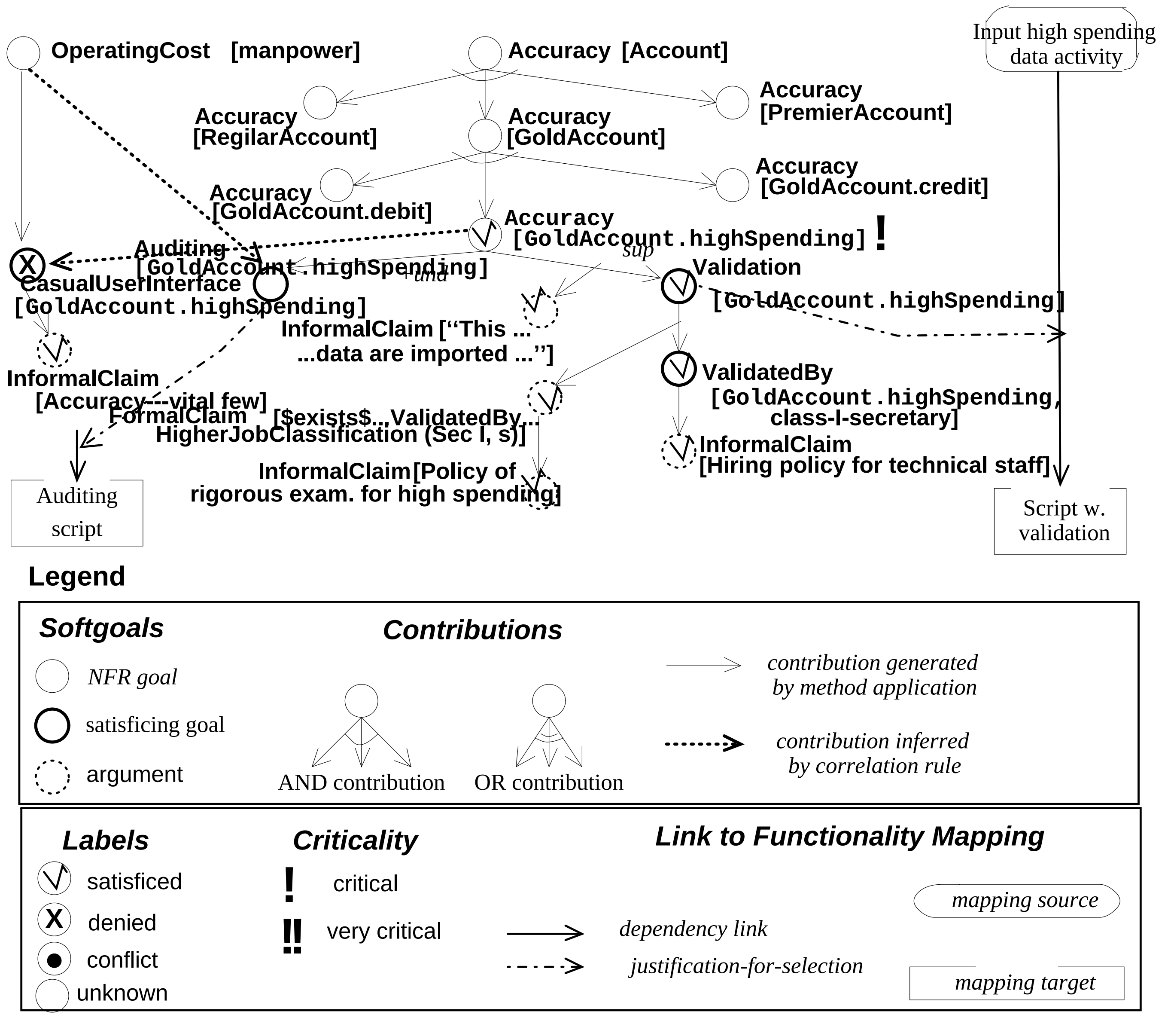

Figure 3.10 Softgoal contribution graph for accurate account attributes.

methods and *correlation rules*. This chapter has presented the first three of the them.

Firstly, the notion of softgoal enables non-functional requirements to be treated as goals which need to be *satisficed* instead of being absolutely satisfied. Non-functional requirements are then disambiguated via decompositions into other non-functional requirements which can be prioritized. These non-functional requirements are next satisficed by satisficing techniques, either design or implementation components, which can be used to meet functional requirements. Just like non-functional requirements, satisficing techniques can also be disambiguated via decompositions into other satisficing techniques which can be prioritized. In the presence of multiple competing techniques, their tradeoffs are taken into consideration before some of them are finally chosen to be included in the target design or implementation. Throughout the refinements of non-functional requirements, their satisficing, prioritization and selection among competing satisficing techniques, arguments can be captured to support for particular design decisions.

Secondly, the notion of contribution (or contribution relationship) enables the relationship between softgoals to reflect the degree to which the offspring satisfice(s) the parent. Associated with each contribution is one of several contribution types, indicating full/partial positive/negative influence. Besides softgoals, a contribution can also involve contributions, hence making it possible to talk about the relationship between a softgoal and a contribution, between a contribution and a softgoal, or even between a contribution and another contribution, etc.

Finally, the evaluation procedure determines the impact of design decisions upon the satisficing of softgoals and contributions. This is done by propagating labels of the offspring upward to the parents, while taking into consideration the contribution types between the offspring and the parent.

These three concepts help represent essential concepts of non-functional requirements and systematically deal with them throughout software life-cycle. This is done by organizing in a softgoal contribution graph design constraints and their associated alternatives, decisions and rationale. This organization then serves as criteria for selecting among target alternatives in meeting functional characteristics of a software system. This way, the NFR framework offers a goal-driven, process-oriented approach to dealing with non-functional requirements, in contrast to an evaluation-centric, product-oriented approach, in software engineering.

3.5.2 Sources of Literature

**************** REFINE

The notion of “satisficing” was used by Herbert Simon, earlier in the context of economics, and later in the context of design [Simon81] where he actually uses the term to refer to decision methods that look for satisfactory solutions rather than optimal ones. The term is adopted here in a broadened sense since in the context of non-functional requirements, even the notions of a solution or optimality of a solution may be unclear.

In the bigger picture of information system development, a source object, say a component of a requirements specification, is mapped into one (or possibly several) target object(s), say components of a design specification [Chung91b]. The contributions among these objects are shown through contributions on the left- and right-hand sides of Figure 3.10.

This organization is very much in the spirit of AND/OR trees used in problem-solving [Nilsson71]. Unlike AND/OR softgoal trees, where the relationship between a collection of offspring and their parent can only be *AND* or *OR*, in our proposed framework there can be several different types of relationships or *contribution types* describing how the satisficing of the offspring (or failure thereof) relates to the satisficing of the parent.

The need for at least some contribution types is evidenced in [Boehm78], which states that some quality characteristics are necessary, but not sufficient, for achieving others. Boehm et al. then use a four-grade scale to correlate each quality metric with quality attributes in the final product. The need for different types of contributions is also supported by Hauser et al. [Hauser88], who, in order to state how much each engineering characteristic affects each customer quality requirement, use four types of values: strong positive, medium positive, medium negative or strong negative.

An earlier, less formal description of some correlations among accuracy, security, cost, and user-friendliness can be found in [Chung91a]. Leveson [Leveson86] presents an elegant discussion of various issues on safety, along with a convincing argument for the need of explicitly capturing tradeoffs. Leveson states that the effect of taking water is relative to the person who takes it. In other words, that water is good for human health is too coarse, but should be discussed, among other things, in terms of the average and threshold amount of water, as well as the age, body weight, health condition, etc., of the person who intakes it. This is, in spirit, also similar to the notions in [Garvin8?] which describes a wide spectrum of different dimensions of quality. For instance, a quality concern such as performance, that lies on the one extreme, tends to be more objective than a concern on user perception, that lies on the other extreme, which is more subjective. Hauser et al. [Hauser88] presents a scheme, informally and without correlation rules, for stating how much each engineering characteristic affects each customer quality requirement in terms of four types of values: strong positive, medium positive, medium negative or strong negative. They are similar to those in [DiMarco90] and generally ones used in qualitative reasoning frameworks [AI84].

It is interesting to compare our evaluation procedure with those of truth maintenance systems (TMSs) [deKleer86], [Doyle79]. They record and maintain beliefs, their justifications and assumptions, while distinguishing facts from defeasible beliefs, which are either accepted or rejected. As with TMSs, our graph evaluation procedure recursively propagates values of offspring to parents. However, our procedure is not automatic, but interactively allows the designer to deal with inconclusive evidence. While we have *AND* and *OR*, comparable to TMS conjunction and disjunction, our contribution types have additional values, all of which are inputs to computing *individual effect* in our first step. In applying the propagation rules of Table 3.1, *contributions*, which are not included in TMS beliefs, must be *satisficed*. Unlike TMSs, we then *combine* individual effects of label values including qualitative (*conflicting*) and open-ended (*undetermined* and "?") ones, using a label ordering in the second step.

CONTENTS

3 THE NFR FRAMEWORK: FOUNDATIONS — SOFTGOAL CONTRIBUTION GRAPH 1

3.1 Softgoals and Contributions 3

3.2 Types of Goals 5

3.3 Types of Contributions 13

3.4 The Evaluation Procedure 20

3.5 Discussion 25