Finding Optimal Solution for Satisficing Non-Functional Requirements via 0-1 Programming

Bin Yin\(^1\), Zhi Jin\(^2\), Wei Zhang\(^2\), Haiyan Zhao\(^2\), Bo Wei\(^i\)

\(^1\)Academy of Mathematics and Systems Science
Chinese Academy of Sciences
Beijing, China

\(^2\)Key Laboratory of High Confidence Software Technologies
Ministry of Education, Peking University
Beijing, China

Abstract—Non-Functional Requirements (NFRs) are vital for the success of software systems. Generally speaking, NFRs are some implicit expectations about how well the software will work, often known as software quality. Building better software, the NFRs should be considered as criteria for design decision. However, different NFRs may produce different criteria on the implementation strategies of the software functions. A trade-off analysis is needed for getting an optimal plan during design decision to satisfice NFRs as well as possible. By focusing on the NFRs that can be quantitatively specified, this paper proposes an approach to finding such an optimal solution for helping to make better decision. This approach regards the NFRs as the constraints on the implementation strategies of the software functions and models the selection of implementation strategies as a 0-1 programming problem. Then, a 0-1 programming solver can be used to find the optimal solution. An example is given to demonstrate the feasibility of this approach.

Keywords: Non-functional requirements; Quality Factors; Design Design; 0-1 programming problem;

I. INTRODUCTION

It has been widely recognized that functional requirements (FRs) describe what the software needs to do and the non-functional requirements (NFRs) specify criteria to judge how well the software does \([1] \) [2]. Currently, there is consensus that NFRs are vital for the success of software development. It is infeasible to produce software that meets stakeholders’ needs without taking NFRs into account \([3]\).

Researchers have already paid many attentions to deal with NFRs. Among them, the NFR Framework \([4] \) [5] [6] is one of the representative works to model and analyze NFRs in requirement phase. It models NFRs as softgoals, referring the global qualities of a software system, and refines them stepwise by a set of operationalization softgoals by using the Softgoal Independency Graph. Several satisficing statuses are adopted to express how well a softgoal is addressed. The NFR Framework also provides a bottom-up evaluation process, called Label Propagation Process, to evaluate independently the satisficing status of the high-level softgoals when given a set of low-level operationalization softgoals.

Some NFRs can be quantitatively specified. These NFRs can be measured or can be represented as quantitative properties \([27]\), such as performance requirements. Such NFRs put measurable constraints on the software-to-be. For example, in cruising control systems, the reliability of the braking function may be required to reach above 99.9\% but may not for other functions. In Email systems, that the time between sending an email to a destination and the destination receiving the email may need to be less than five seconds. In on-line shopping systems, the payment function needs to be totally secured but not for the browsing function. Furthermore, these NFRs obviously place constraints onto particular functions rather than the entire system. Dealing with them without taking care of the particular functions may be meaningless.

Often, many software functions may have multiple possible implementation strategies. Making a good choice among them is important for building better software. Obviously, the selection of the implementation strategies should be on the basis of these quantitative properties. But it is quite often that for the same function there might be more than one NFR placing constraints onto it. For example, suppose that the payment function in the software-to-be is required to be secured and needs to be as fast as possible. Current techniques can lead to many strategies for securing the function. But these strategies often take a lot of time and many resources. They have negative impacts on the performance of the payment function. A trade-off analysis is needed for getting an optimal plan for implementing software functions and at the same time satisficing the NFRs as well as possible. How to find such an optimal solution for the NFRs becomes a research question.

Selecting better strategies for implementing the function requirements and at the same time satisficing the non-functional requirements is very important for building better software. However, at present, this could be done on the basis of the developers’ experience in most software projects. There are few methods and tools that can be used for helping the developers to make good decision.

This paper proposes an approach to finding optimal solution on the implementation strategies of software functions for satisficing NFRs. This approach adopts the Problem Frame (PF) approach. It models the software
behaviors by using the problem diagram. Based on the problem diagram, it constructs a phenomenon dependency graph (PDG) for explicitly representing the process of achieving a functional requirement. The phenomenon dependency relations in PDG are of two types: ensured by software and by domains. Those phenomenon dependency relations that are ensured by software are software operations; each of them may have multiple implementation strategies. By placing the NFRs’ constraints to the phenomenon dependency relations, the selection of the implementation strategies for realizing the software function becomes a 0-1 programming problem. Then a 0-1 programming solver can be used to find the optimal solution.

The rest of the paper is organized as follows. Section 2 is about the key ideas of our approach. Section 3 introduces this approach in detail. Section 4 presents a case study. The related works are discussed in Section 5. And finally, we conclude this paper in Section 6.

II. PRELIMINARY OF OUR APPROACH

This section introduces the key ideas of our approach and explains the terminologies we will use.

A. Extension of the Problem Frames Approach

For modeling and analyzing those NFRs that produce the design criteria, we need to model the software behaviors firstly. The PF approach [7] [8] [9] is good at modeling and capturing the system behaviors.

The PF approach uses the problem diagram to specify a software development problem. A problem diagram contains five parts. The first three are the machine specification (S), the domain assumptions (W) and the requirements (R). The other two are the set of shared phenomena between the machine and the domains and the set of shared phenomena between the domains and the requirements. We call the former the specification phenomena (PSpe) and the latter the requirement phenomena (Preq).

Here, we use an example from [8], i.e. the occasional sluice gate controller, to show the five parts in a problem diagram. The sluice controller (machine) receives the commands from the sluice operator (domain) and manipulates the motor and gate (domain) according to these commands, so that requirements “Raise & Lower gate” can be satisfied. Fig. 1 shows the problem diagram of this problem.

Specifically, the problem diagram in PF approach explicitly represents the interactions between the machine (i.e. the software) and the problem domains by using shared phenomena. This can be used to derive the software behaviors.

From another angle to look at this diagram, we can semantically describe above three specifications as sets of phenomenon dependency relations.

More concretely, the machine specification (S) can be represented by a set of dependency relations among the specification phenomena. Each of the dependency relations expresses the roles that the machine plays when sharing the phenomena. For example, in the example shown in Fig.1, when machine shares the phenomenon “SO!Stop” with “Sluice Operator”, it semantically is required to initiate another shared phenomenon “SG!Off” with “Gate & Motor”. We call such a dependency relation the machine operation (Opt) that can be represented formally as:

\[ \text{Opt: } P_{spe} \rightarrow P_{spe} \]

For the same reason, the domain assumptions (W) can also be semantically represented by a set of domain statements (Sm) about the dependency relations between the requirement phenomena and the specification phenomena. These domain statements specify the relations that can be enabled by the domain between the specification phenomena and the requirement phenomena. In terms of the principle of the PF approach, the domain assumptions fall into two categories. One is about the requirement reference phenomenon that means that the requirement only refers to certain phenomena of the domain. This kind of statements is represented as:

\[ \text{Stm: } P_{req} \rightarrow P_{spe} \]

The other one is about the requirement constraining phenomena that mean that the requirement doesn’t just refer to the domain phenomena but also stipulate some desired relationships or behavior involving them. This kind of statements is represented as:

\[ \text{Stm: } P_{spe} \rightarrow P_{req} \]

The requirements (R) consist of the functional requirements (FRs) and the non-functional requirements (NFRs). PF approach only deals with the FRs explicitly. FRs can be specified as the dependency relations among the requirement phenomena, which can be represented as:

\[ \text{FR: } P_{req} \rightarrow P_{req} \]

The NFRs part will be discussed in next subsection.

For each requirement belonging to FR, a phenomenon dependency graph (PDG) can be obtained according to the phenomena dependency relations in both Opt and Stm. This PDG shows the implementation process of the functional requirement. We will present the construction method for PDG in next section.

To be noted that the specifications are of different modes. The machine specification gives the optative description about the machine’s desired behaviors, the domain

![Figure 1. An Example of problem diagram of sluice gate control problem](image-url)
specification is an *indicative* description on the objective truth about the domain and the requirements specification is an *optative* description about what the stakeholder would like to be true in the problem domain.

The PF approach considers the software development as a problem which is to find a machine specification with the given domain assumptions and the requirements. The task of software developers is to find suitable implementation strategies (e.g. algorithms or implementation techniques) for the required machine behaviors/operations to meet the functional requirements. That brings in another set of descriptions, i.e. the set of implementation strategies for machine operations. Normally, there might be more than one implementation strategies for each machine operation and each implementation strategy can be a candidate plan for implementing the corresponding machine operation.

### B. NFRs, Operationalization Strategies and NFPs

There are different statements about NFRs, including: (1) NFRs express desired qualities of the system to be developed [3]. (2) NFRs are requirements that place restrictions on the product being developed and the development process, and specify external constraints that the product must meet [10]. (3) NFRs are quality characteristics of the software-to-be, such as accuracy, security and performance [4]. Obviously, NFRs have been regarded as constraints or desired properties of the software-to-be.

Many NFRs place constraints on concrete software functions, rather than on the entire system. They place constraints on the implementation strategies of the software behaviors. For example, security requirements are about protecting some software assets. Therefore, the system operations referring to these to-be-protected assets should adopt some suitable strategies to protect these assets against being attacked for achieving the desired security requirements. The performance requirements may stipulate desired performance factors (throughput, response time, latency and so on) of particular software operations. As a result, the efficiency of the algorithms becomes a constraint when they are chosen to be the implementation strategies of the operations.

Sometimes, NFRs are prescriptive and undermined. But some of them can be refined to the NFRs that can be measured or can be represented as quantitative properties of particular functions of software-to-be. These quantitative properties are helping for making good design decision. We name these quantitative properties as non-functional properties, i.e. NFPs. The NFPs obtained by refining NFRs are called *desired NFPs* of particular function.

On the other hand, in terms of the PF approach, each software function is completed by some software operations. Often, there are different implementation strategies for the same operation and each implementation strategies may possess different non-functional properties (NFPs). For example, in On-line system, the login operation may have two implementation strategies, i.e. using a heavy-weight thread pool to allow ten thousands users to login simultaneously but take up just 1G memory, or applying a light-weight thread pool to support only one thousand users to login simultaneously but take up just 1G memory. We call these implementation strategies possess their own *realizable NFPs*.

### C. Implementation Strategy Selection as 0-1 Programming Problem

As mentioned above, each functional requirement has a PDG that is constructed to show its implementation process. On the other hand, each functional requirement may have multiple NFRs. That gives the multiple desired NFPs of the software operation in PDG. Furthermore, each operation in PDG has a set of implementation strategies, each of which possesses its own realizable NFPs.

However, for any software operation, only one implementation strategy can be selected in the design solution. We define a variable “s;” to represent the selection status of implementation strategy “i” of operation “i”, i.e. “s;=1” means that strategy “j” of operation “i” is selected and “s;=0” means it is non-selected.

Normally, NFRs fall into two categories: mandatory and optional. A mandatory NFR stipulates the value range of the desired NFP. For any software operation, if the value of the realizable NFP is within this range, the desired NFR is satisfied; while an optional NFR does not limit the value of the desired NFP. It only requires selecting an implementation strategy for the software operations that the realizable NFP is optimized (e.g. maximized or minimized).

Among the NFRs, assume that only one NFR is optional and the others are mandatory. Then the selection of implementation strategies aims to fully satisfy the mandatory NFPs and satisfy the optional NFP as well as possible.

In mathematics, programming is the process of solving a system of equalities and inequalities, collectively termed constraints, over a set of unknown read variables, along with an objective function to be maximized or minimized [11]. The 0-1 programming [11] is a special case of programming whose variables are required to be 0 or 1.

Let’s be back to the selection of the implementation strategies. We find that it can be reduced into a programming problem. In which, the optional NFP plays the role of the objective function, the mandatory NFPs are taken as constraints and the selection variables of implementation strategies (s; ) are unknown variables of programming. It is noted that the selection status of an implementation strategy is binary: selected or non-selected. The 0-1 programming problem is apt to model this problem. Some mature programming methods, such as the branch and bound algorithm [12] and the generalized benders decomposition [13] could be used to solve the problem.

### III. APPROACH DESCRIPTION

The approach includes five steps: to model the software behavior, to determine the desired NFPs, to acquire the realizable NFPs, to construct the expressions of realizable NFPs of software behavior and to find the optimal solution.

Fig. 2 shows the process of this approach. The rest of this section will go through all the steps with details.
A. Model Software Behavior

The first step is to model the software behavior by using the extended PF approach. The input is a problem diagram. It contains five groups of descriptions including: the machine specification, the domain assumptions, the requirements, the specification phenomena and the requirement phenomena. The output is the phenomenon dependency graph. It depicts the software behavior that meets the functional requirement. This step consists of three sub-steps.

Step A.1: Model problem description.

Based on the problem diagram, we can define the software operations (Opt), the statements of domain assumptions (Stm) and the functional requirements (FR) respectively. Each of them is a set of the phenomenon dependency relations. For capturing the phenomenon dependency relation, we need firstly to introduce some notations as follows.

**Ph**: the set of phenomena captured in problem diagram

According to the PF approach, the captured phenomena fall into two categories: the event phenomenon and the state phenomenon\(^1\). So set **Ph** is divided into two disjoint sub-sets: **Ph**\(_{event}\) and **Ph**\(_{state}\), i.e.

\[
\text{Ph} = \text{Ph}_{\text{event}} \cup \text{Ph}_{\text{state}}
\]

Let \(D = \{d_1, ..., d_n\}\) be the set of domains\(^2\) in this problem, we define two atomic propositions:

- \(\text{Occur}(e), e \in \text{Ph}_{\text{event}}\): The event \(e\) is triggered.
- \(\text{Is}(s_d), s_d \in \text{Ph}_{\text{state}}, d \in D\): Domain \(d\) is in state \(s\). Here, \(d\) is a casual domain.

With these, we include another set **Con** to represent the set of conjunctions of the “Is” atomic propositions, which can be represented as follows:

\[
\text{Con} = \{\text{con}|\text{con} = \bigwedge_{s_d \in K} \text{Is}(s_d), K \in 2^{\text{Ph}_{\text{state}}}\}
\]

The phenomenon dependency relation falls into three types in terms of characteristics of these descriptions. The functional requirements can be formed as follows:

\[
\text{con}_1 \land \text{Occur}(e) \rightarrow \text{con}_2
\]

\[
e \in \text{Ph}_{\text{event}} \land \text{con}_1, \text{con}_2 \in \text{Con}
\]

It can be read as: if the condition \(\text{con}_1\) is true and event \(e\) is triggered, then condition \(\text{con}_2\) needs to be enabled to be true. The machine operations and the domain assumptions are in two forms. The first is:

\[
\text{con} \land \text{Occur}(e_1) \rightarrow \text{Occur}(e_2)
\]

\[
e_1, e_2 \in \text{Ph}_{\text{event}} \land \text{con} \in \text{Con}
\]

It means that if the condition \(\text{con}\) is true and event \(e_1\) is triggered, then event \(e_2\) needs to be triggered. The second is in the form of:

\[
\text{con} \land \text{Occur}(e) \rightarrow \text{Is}(s_d)
\]

\[
e \in \text{Ph}_{\text{event}} \land s_d \in \text{Ph}_{\text{state}} \land \text{con} \in \text{Con}
\]

It means that if condition \(\text{con}\) is true and event \(e\) is triggered, then domain \(d\) needs to be in its state \(s\).

After this step, the operations in the machine specification, the statements in the domain assumptions and the functional requirements are represented as sets of phenomenon dependency relations. They are named as **Opt**, **Stm**, **FR** respectively.

Step A.2: Construct implementation process of functional requirement

In this sub-step, for each requirement \(r\) in the **FR**, we construct the implementation process by formally proving the following formula:

\[
\text{Opt, Stm} \vdash \text{r}, \text{r} \in \text{FR}
\]

First, let requirement \(r\) be formed as:

\[
\text{R}_{\text{pre}} \rightarrow \text{R}_{\text{con}}
\]

In which, \(\text{R}_{\text{pre}}\) stands for the premise of functional requirement \(r\) and \(\text{R}_{\text{con}}\) is about the expected effects on problem domains. Then, we apply the natural deduction proof of propositional logic [14]. Here, a semi-automatic tool named coq [15] has been used in this step. The input is the set of formulas, i.e.

\[
\text{Input} = \text{Opt} \cup \text{Stm} \cup \text{R}_{\text{pre}}
\]

The to-be-proved formula is \(\text{R}_{\text{con}}\). Then coq produces the output that is a sequence of formula as the proof of formula:

\[
\text{Opt, Stm} \vdash \text{r}
\]

---

\(^1\) The value phenomenon is regarded as a special case of state phenomenon in this paper.

\(^2\) The domain falls into three categories: casual domain, lexical domain and biddable domain. The lexical domain is a special case of casual domain.
The sequence of formula in this proof shows the deduction procedure. As coq adopts backward reasoning, this sequence of formula is just in the reverse order of the implementation process of requirement $r$. By reversing the output sequence and reassigning an order number (i.e. 1, 2, 3, …) to each formula in the reversed sequence, the implementation process of $r$ is obtained. Each element in the implementation process is a triple as:

$<$Number, Formula, Reason $>$

It states that formula Formula is proved using premise Reason in step Number. Here, Number is the order of the element. Formula is the formula that has been proved in current step. Reason gives the reason why the formula is true. All atomic propositions in $R_{con}$ should appear as a Formula.

It is noticed that Reason falls into three types.

Type 1: it is one of the atomic propositions in $R_{pre}$. That means this formula is one of in the premise of $r$.

Type 2: it is a phenomenon dependency relation in Opt or Stm. That means this formula is a software operation or a statement in domain assumptions.

Type 3: it is a list of existing order numbers. That means this formula is proved by some existing elements. Where the last number in this list refers to a formula belongs to Opt or Stm that has appeared in the proof sequence.

If Reason of an element is of the type 1 or type 3, its Formula is an atomic proposition, otherwise a phenomenon dependency relation.

Step A.3: Construct phenomenon dependency graph.

After getting the implementation process of a functional requirement, the next step is to construct its phenomenon dependency graph (PDG). PDG is used to show the system behavior visually for this functional requirement. It is a directed acyclic graph, in which each node corresponds to an atomic proposition (i.e. a phenomenon) and each edge corresponds to a phenomenon dependency relation in Opt or Stm.

The inputs of this step include the implementation process obtained in Step A.2 and an empty table named as tab that will be used to record the formulas from Opt or Stm that have appeared in the proof.

Let $PDG=(V_{PDG}, E_{PDG})$ be initially an empty graph. This step iteratively deals with each element in the implementation process to construct a PDG incrementally.

Let $ele=<\text{num}, \text{form}, \text{rea}>$

be the element being dealt with currently. As shown above, the elements in the implementation process falls into three cases. Each case derives different operations on PDG.

Case 1: rea is of type 1. Then, add a vertex labeled by form in PDG, i.e.

$V_{PDG} = V_{PDG} \cup \{\text{form}\}$

Case 2: rea is of type 2. Then, insert form in tab as an entry with index num.

Case 3: rea is of type 3, i.e.

$rea = \{\text{num}_1, \text{num}_2, ..., \text{num}_{n-1}, \text{num}_n\}$

Then the first n-1 numbers refer to n-1 formulas that have been proved and serve as the premise in the current proof step. That means that the nodes for these formulas have been inserted in PDG as vertexes. Let $V$ contain all the vertexes corresponding to these formulas. Then

- Add a vertex labeled by form in PDG, i.e.
  
  $V_{PDG} = V_{PDG} \cup \{\text{form}\}$

- for each node $n$ in $V$, add an edge $e$ in PDG which link $n$ to the vertex labeled by form, i.e.
  
  $E_{PDG} = E_{PDG} \cup \{e\}$

- get formula form with index $\text{num}_n$ from tab and label $e$ with form

B. Determine desired NFPs

As mentioned above, for each functional requirement $r$ ($R_{pre} \rightarrow R_{con}$), there might be several NFRs, which require the PDG of $r$ to exhibit different desired NFPs. This step is determining the desired NFPs by refining the NFRs to the constraints on the functional requirements. This step needs the participation of stakeholders.

C. Acquire realizable NFPs

Each node in the PDG of $r$ is a phenomenon that occurs when realizing $r$ and each edge in the PDG of $r$ represents a machine operation or a domain statement. The path from the nodes in $R_{pre}$ to the nodes in $R_{con}$ determines the NFPs of the PDG of $r$.

Moreover, the edges in the PDG are about the machine operations and the domain statements. The realizable NFPs of the domain statements are decided by the domains, which can be obtained from the domain experts. While, normally, a machine operation may have multiple implementation strategies, each of which possesses its own realizable NFPs. This step is to acquire these realizable NFPs with the help of domain experts and experienced software developers.

Step B and C are just information acquisition. That is not the core concerns of this paper. These NFPs are inputs of the following step.

D. Construct expressions of realizable NFPs of software behavior

For judging whether the realizable NFPs can satisfy the desired NFPs, it needs to calculate the NFPs of the PDG. Let $r$ be a functional requirement and PDG$(r)$ be the PDG of $r$. We denote the realizable NFPs of PDG$(r)$ as follows:

$NFP(r), r \in FR$

It is noted that the NFPs of the machine operations and the domain statements have not been assigned values yet at this step. Thus, NFP$(r)$ is represented as an expression with reference to those variables.

Jawaria Sadiqin [16] proposes a quantification mechanism for NFRs, in which a quality model is proposed according to ISO/IEC 9126 [17]. The quality model proposes
four types of the metric values: Existence (E), Time (T), Percentage (P) and Numeric value (NV). By following this, this paper categorizes the values of NFPs into three types in terms of the different quantitative measurement ways.

- **Numeric Value (NV):** the NFPs that are measured with numeric value, such as the consumed energy, the memory used, etc.
- **Percentage (P):** the NFPs that are measured with probability, such as confidentiality, reliability, etc.
- **Time (T):** the NFPs that are measured with time, such as response time, etc.

With this categorization, NFP(r) can be represented in three ways.

**NFPs measured with numeric value.**

A cumulative way is applied to the NFPs measured with numeric value. It is represented as follows:

\[
NFP(r) = \sum_{i=1}^{n} NFP(e_i), E = \{e_1, \ldots, e_n\}
\]

**NFPs measured with probability.**

It is assumed that the machine operations and the domain statements are independent of each other. According to the multiplication rule in probability theory, a multiplicative way is applied to NFPs measured with probability. It is represented as follows:

\[
NFP(r) = \prod_{i=1}^{n} NFP(e_i), E = \{e_1, \ldots, e_n\}
\]

**NFPs measured with time.**

A labeling way is used to NFPs measured with numeric value. At first, the vertexes whose in-degree is 0 are labeled as 0. Then, the other vertexes are labeled stepwise by using the labeling criterion below:

- \(\text{label}(v_i) = \text{MAX}(\text{label}(v_m), \ldots, \text{label}(v_n)) + NFP(e_i)\)
  - \([v_m, \ldots, v_n]\): the set of previous vertexes of \(v_i\)
  - \(e_i\): the incoming edge of \(v_i\)

After all of the vertexes in PDG(r) have been labeled, the NFP(r) can be represented like this:

\[
NFP(r) = \text{MAX}(\text{label}(v_i), \ldots, \text{label}(v_j))
\]

\([v_i, \ldots, v_j]\): the set of vertexes whose out_degree is 0

**E. Find optimal solution**

In the last step, the NFPs of the PDG for the functional requirement are represented as expressions with reference to the NFPs of machine operations and domain statements. As machine operation may have several implementation strategies and each of which possesses its own realizable NFPs, it is needed to make choice among them. Moreover, different NFRs may produce different criteria on the implementation strategies. A trade-off analysis is needed for getting an optimal plan during design decision to satisfy NFRs as well as possible.

This step aims to model the selection problem as a 0-1 programming. At first, we introduce two variables. The first one is:

\[
s_{ij} \in \{0, 1\}
\]

It stands for the selection status of the implementation strategy “j” of the operation “i”. “s_{ij}=1/0” means the strategy is selected or non-selected. The second one is:

\[
NFP_{ij}
\]

It stands for the realizable NFP of implementation strategy “j” of operation “i”. If implementation strategy “j” is selected, the NFP of operation “i” equals NFP_{ij}.

As an operation can have multiple implementation strategies, but only one strategy can be adopted in the implementation process, the following constraint is needed:

\[
s_{i1} + s_{i2} + \ldots + s_{in} = 1
\]

which ensure that operation “i” has n implementation strategies and only one strategy can be selected. And then, NFPs of operation “i” can be represented like this:

\[
NFP(\text{opt}_i) = s_{i1} * NFP_{i1} + s_{i2} * NFP_{i2} + \ldots + s_{in} * NFP_{in}
\]

In this paper, we assume that the NFPs of the domain statements are known, which can be obtained from the domain experts.

For each functional requirement, there might be several NFRs. Some are mandatory and some are optional. This paper assumes that only one optional NFR is allowed. Then, the selection of implementation strategies can be reduced into a 0-1 programming, where the optional NFP plays the role of the objective function, the mandatory NFPs are considered as constraints and the selection variables of implementation strategies (s_{ij}) are the binary variables in 0-1 programming.

Note that the constraints that come from the NFPs measured with time need to be dealt with specially, as the expression of NFP contains operation MAX. The programming problems with MAX operation need to be divided into two programming problems. For example, assume that a 0-1 programming problem contains a constraint: MAX(a,b)>5. This programming problem needs to be divided into the following two programming problems. One has two additional constraints: a<b and b<5. The other has two different additional constraints: a≥b and a≤5. The result of the original programming equals the best result of these two sub-programming.

**IV. Case study**

This section uses an example to demonstrate the feasibility of our approach. Suppose one company plans to develop a cell phone application to issue notifications to the employees. One of the core functional requirements of this application is:

*The administrators can publish notifications using this application. The published notifications can be downloaded to cell phones of employees and shown on display screens.*
According to this functional requirement, four problem domains can be identified: the Administrator, the Notification Database, the Local storage of cell phones and the Display screens of cell phones. The domain of Administrator is biddable. It can trigger event of submitting notifications. The machine receives this event and store submitted notifications into the Notification Database, which is lexical too. The published notifications are sent to cell phones of employees and stored in local storage, which is lexical too. At last, the domain of display screen shows the notifications. It is a casual domain.

For this functional requirement, the problem diagram of the machine is shown in Fig. 3.

![Problem diagram of cell phone notification machine](image)

Figure 3. Problem diagram of cell phone notification machine

Then, the stakeholder further proposes several NFRs including:

- **Integrity**: this machine must consider the integrity of notifications. The reason is that: the notifications may involve confidential information and some security strategies are needed to protect them against being intercepted and interpolated by malicious attackers form the Internet.

- **Time**: some notifications are urgent and need to be sent to employees as soon as possible. Thus, the machine should shorten the consumed time from publishing to display.

- **Energy**: the machine is running on cell phones. Since the cell phone has limited energy, the energy cost of this application should be taken into account.

### Step A: model software behavior

As shown in Fig. 3, there are seven phenomena captured in the problem diagram. They comprise the set \( Ph \).

\[
Ph = \{ a, b, c_{ND}, d, e_{LS}, f, g_{DP} \}, \text{in which}
\]

- \( Ph_{event} = \{ a, b, d, f \} \)
- \( Ph_{state} = \{ c_{ND}, e_{LS}, g_{DP} \} \)

Then, we construct the domain statements (Stm), the machine operations (Opt) and the functional requirement of ‘sending notification’ (FR). Table I gives all the descriptions.

### Table I. Descriptions of Statements, Operations and Functional Requirement

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stm</td>
<td>( \text{Occur}(b) \rightarrow Is(c_{ND}) )</td>
</tr>
<tr>
<td>Stm</td>
<td>( \text{Occur}(d) \rightarrow Is(e_{LS}) )</td>
</tr>
<tr>
<td>Stm</td>
<td>( \text{Occur}(f) \rightarrow Is(g_{DP}) )</td>
</tr>
<tr>
<td>Opt</td>
<td>( \text{Occur}(a) \rightarrow \text{Occur}(b) )</td>
</tr>
<tr>
<td>Opt</td>
<td>( \text{Occur}(b) \rightarrow Is(c_{ND}) )</td>
</tr>
<tr>
<td>Opt</td>
<td>( \text{Occur}(b) \rightarrow Is(e_{LS}) )</td>
</tr>
<tr>
<td>Opt</td>
<td>( \text{Occur}(d) \rightarrow Is(g_{DP}) )</td>
</tr>
<tr>
<td>FR</td>
<td>( \text{Occur}(a) \rightarrow Is(c_{ND}) \land Is(e_{LS}) \land Is(g_{DP}) )</td>
</tr>
</tbody>
</table>

From Table I, the functional requirement is

\[
r = \text{Occur}(a) \rightarrow Is(c_{ND}) \land Is(e_{LS}) \land Is(g_{DP})
\]

where \( R_{pre} = \text{Occur}(a) \) and \( R_{con} = Is(c_{ND}) \land Is(e_{LS}) \land Is(g_{DP}) \).

Using coq to prove

\[
Opt, Stm \vdash r
\]

We can obtain the implementation process of \( r \), which is shown in Fig. 4.

![Implementation process of r](image)

Figure 4. Implementation process of \( r \)

By following step A.3 in section III, we can obtain the corresponding phenomenon dependency graph, which is shown in Fig. 5.

![Phenomenon dependency graph for r](image)

Figure 5. Phenomenon dependency graph for \( r \)
Step B: Determine desired NFPs
There are three NFRs, which place constraints on this PDG. The integrity and the consumed time are mandatory NFRs. The energy cost is optional. From the NFRs, the desired NFPs can be obtained.

Mandatory NFRs:
- Integrity: As the experience shows that the probability of the notifications being intercepted by malicious attackers is less than 2%, this desired NFP for integrity is: I(r) ≥ 98%.
- Time: As the desired time it takes for successfully sending notifications should be less than 20 milliseconds, this desired NFP for time is: T(r) ≤ 20.

Optional NFR:
- Energy: As it is needed to consume the energy as less as possible when completing function r, the desired NFP for energy is: Min(E(r))

Step C: Acquire realizable NFPs
The realizable NFPs of the machine depend on two parts, i.e. the NFPs of the domain statements and the NFPs of the machine operations. The NFPs of the domain statements are given in the domain assumptions. They are domain facts. There are three domain statements involved in the implementation process of function r. Table II shows the NFPs of these three domain statements.

Table II. NFPs of Statements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Statement</th>
<th>Integrity</th>
<th>Time</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>stm1</td>
<td>Occur(b) → Is(element)</td>
<td>I(stm1)=99.9%</td>
<td>T(stm1)=2</td>
<td>E(stm1)=0</td>
</tr>
<tr>
<td>stm2</td>
<td>Occur(d) → Is(element)</td>
<td>I(stm2)=99.8%</td>
<td>T(stm2)=1</td>
<td>E(stm2)=6</td>
</tr>
<tr>
<td>stm3</td>
<td>Occur(f) → Is(element)</td>
<td>I(stm3)=99.9%</td>
<td>T(stm3)=3</td>
<td>E(stm3)=7</td>
</tr>
</tbody>
</table>

As for the machine operations, there is a slight difference from the domain statements. The machine is to-be-built. It has no assumptions. It will be implemented by some implementation strategy, which is chosen in the design phase. But at the design phase, there might be multiple choices based on the up-to-date techniques. The design decision needs to be made according to the NFRs.

Operation “opt,” of the machine enables receiving the request from the administrator and storing notifications in database through the Internet. Both the integrity and the time need to be taken into account. The developer gives two implementation strategies, i.e. the cipher text transmission and the plain text transmission. As the cipher text transmission is good to data integrity while the plain text transmission spends less time.

Operation “opt,” of the machine enables sending the published notifications onto employees’ cell phones. Also, the developers think that are two choices, i.e. two formats that have been widely used to exchange information between computers and cell phones, the JSON format and XML format. The JSON format is simple and easy to use, but it may imply security risks. The XML format is more mature and more secure than JSON format, but parsing XML format needs more time.

Table III shows the NFPs of these implementation strategies.

Table III. NFPs of Implementation Strategies

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Operation</th>
<th>ImpStr</th>
<th>Integrity</th>
<th>Time</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt1</td>
<td>Occur(a) → Occur(b)</td>
<td>Cipher text transmission</td>
<td>I1=99.9%</td>
<td>T1=5</td>
<td>E1=0</td>
</tr>
<tr>
<td>opt2</td>
<td>Occur(b) → Occur(d)</td>
<td>Plain text transmission</td>
<td>I2=99%</td>
<td>T2=1</td>
<td>E2=0</td>
</tr>
<tr>
<td>opt3</td>
<td>Occur(d) → Occur(f)</td>
<td>Json format</td>
<td>I3=99.3%</td>
<td>T3=2</td>
<td>E3=7</td>
</tr>
<tr>
<td>opt4</td>
<td>Occur(d) → Occur(f)</td>
<td>Xml format</td>
<td>I4=99.8%</td>
<td>T4=5</td>
<td>E4=11</td>
</tr>
<tr>
<td>opt5</td>
<td>Listening model</td>
<td>Timing model</td>
<td>I5=99.9%</td>
<td>T5=7</td>
<td>E5=8</td>
</tr>
</tbody>
</table>

Step D: Construct expressions of the realizable NFPs of PDG for requirement r
The realizable NFPs of the PDG for requirement r are represented as expressions with reference to the NFPs of the domain statements and the machine operations according to the types of the NFPs. The results are shown here.

- \( E(r) = \sum_{i=1}^{3} E(\text{opt}_i) + \sum_{j=1}^{3} E(\text{stm}_j) \)
- \( I(r) = \prod_{i=1}^{3} I(\text{opt}_i) \cdot \prod_{j=1}^{3} I(\text{stm}_j) \)
- \( T(r) = \max( T(\text{opt}_1) + T(\text{stm}_1), T(\text{opt}_2) + T(\text{stm}_2), T(\text{opt}_3) + T(\text{stm}_3) ) \)

Step E: find optimal solution
This step reduces the selection of the optimal implementation strategies into a 0-1 programming problem by defining the objective function and the constraints. The 0-1 programming problem for this case study is shown in Fig. 6.

This paper chooses LINGO [18] as the solver and get the optimal implementation strategies, which are shown in the table IV.

Table IV. Results of the 0-1 Programming

<table>
<thead>
<tr>
<th>Name</th>
<th>Operation</th>
<th>Implementation strategy</th>
<th>Selection status</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt1</td>
<td>Occur(a) → Occur(b)</td>
<td>Cipher text transmission</td>
<td>√</td>
</tr>
<tr>
<td>opt2</td>
<td>Occur(b) → Occur(d)</td>
<td>Plain text transmission</td>
<td>×</td>
</tr>
<tr>
<td>opt3</td>
<td>Occur(d) → Occur(f)</td>
<td>Json format</td>
<td>√</td>
</tr>
<tr>
<td>opt4</td>
<td>Occur(d) → Occur(f)</td>
<td>Xml format</td>
<td>×</td>
</tr>
<tr>
<td>opt5</td>
<td>Listening model</td>
<td>Timing model</td>
<td>×</td>
</tr>
</tbody>
</table>

With this design decision, the NFPs of function r could be:
The required non-functional requirements are satisfied.

\[
I(r) = 98.71\%, T(r) = 17, E(r) = 28
\]

V. RELATED WORKS

In recent years, many efforts have been done for modeling and analyzing NFRs. Besides those work that deal with the NFRs separately, many of them extend the existing methods for dealing with functional requirements by including new concerns on the non-functional requirements.

The first group of efforts targets the Use Cases approach. For example, as a supplement to Use Cases, Sindre and Opdahl [19] invent the concept of Misuse Case to help eliciting security-related NFRs. Misuse cases allow early focus on security by describing security threats and then requirements. They also provide a process of eliciting security requirements with misuse case, which includes five steps: identify critical assets, define security goals, identify threats, identify and analyze risks, define security requirements.

Inspired by the Control Theory, Liu & Jin et al. [25] introduce Control Cases for helping modeling the dependability of software. A control cases based model is presented to integrate the modeling of the functional requirements and dependability requirements.

By combining the Use Case and the NFR Framework, Chung et al. [6] propose a pattern approach to capture and reuse functional and non-functional requirements knowledge. The Use Case is used to capture functional requirements, on the basis of which, the goal-oriented NFR Framework is adopted to capture NFRs.

Some other efforts work on the extension of the PF approach. Hatebur et al. [20] present a new kind of problem frames tailored for representing security problems, called security frames. The security frames help to comprehend, locate and represent security problems. And then a set of architectural patterns are proposed in order to solve security problems.

Lin et al. [26] introduce the notion of anti-requirements (requirements of malicious attackers) into the PF approach and propose three kinds of Abuse Frames to analyze security threats and vulnerabilities. The abuse frame can be composed with a base problem by mapping domains. And a composed problem can help to analyze threads and show how the anti-requirement is satisfied.

Elisabeth A. Strunk and John C. Knight [21] combine the PF approach with assurance cases to analyze dependability of the system. In this approach, the PF approach is used to represent the functional requirements and the assurance cases help to analyze frame concerns about dependability requirements.

Our approach also targets the PF approach. The emphasis of our approach lies in modeling the software behaviors using the PF approach and explicitly placing the NFRs' constraints onto the implementation strategies of these software behaviors. Then the problem of selecting a suitable implementation strategies to satisfying the NFRs is reduced to a 0-1 programming problem, which can be solved by existing solver. The other difference is that our approach deals with the quantitatively measurable NFRs, while the others use qualitative way.

On the other hand, there are some works on selecting operationalization strategies along the goal-oriented approach. Jennifer Horkoff and Eric Yu [22] have proposed an interactive backward reasoning approach to finding solutions in goal models. This work is based on the i* Framework. The evaluation producer propagates top-down from high-level target goals. The role of human intervention is included in this producer, which aims to resolve conflicting contributions among NFRs. The procedure also encodes propagation rules in conjunctive normal form and applies a SAT solver to search for an acceptable solution.

Paolo Giorgini et al. [23] propose a formal goal model in the Tropos methodology. This goal model supports two types of analysis, forward (bottom-up) reasoning and backward (top-down) reasoning. Unlike the above work, this goal model use a noteworthy variant of SAT named Minimum-Weight Propositional Satisfiability (MW-SAT), in which each Boolean variable is given a positive integer weight. The MW-SAT is a problem to find an assignment satisfying the target expression, which minimizes the sum of weight. Thus, in this goal model, each soft-goal can be assigned a quantitative factor.

Bo Wei et al. [24] propose an automatic reasoning mechanism for NFR goal models. In this mechanism, the partial satisfying status occurring in the reasoning process is propagated to parent node, rather than be clarified through interaction with stakeholders. This automatic reasoning mechanism significantly increases the efficiency of the whole evaluation process when the goal model has large number of nodes.

Christopher Burgess [28] also aims to perform label propagation on SIGs automatically without human
intervention. They extend the NFR Framework by introducing the nodes of interdependency rulesets, which contain if-then rules and are used to define the label to be propagated to the NFR softgoal.

Our approach also reduces the problem of selecting implementation strategies into a mathematical model. Different from them, our approach focuses on the quantitatively measurable NFRs and categorizes them into three types that have different quantitative measurement ways. Then, a quantitative optimization model (i.e., 0-1 programming) can be obtained, rather than a SAT model. That helps to find the optimal set of implementation strategies.

VI. CONCLUSIONS

This paper proposes an approach to finding the optimal implementation strategies for satisfying the NFRs. This approach will help the developers to make good design decisions. The main features of our approach include:

- It provides an explicit expression of software operations for realizing its functions by extending the problem frames approach. It also allows specifying different implementation strategies for each software operation;
- It focuses on the quantitative NFRs that can be specified as measurable properties. These properties place explicit constraints to the functions of the software-to-be and then restrict the selection of the implementation strategies;
- It reduces the selection of the implementation strategies to a 0-1 programming problem. Then the available mathematical solvers can be used to automatically solve the selection of implementation strategies.

In the future, we will apply our approach to real case studies to further validate the proposed approach. We are also going to develop a practical tool to support our approach. Furthermore, regarding some NFRs are qualitative, how to synthesize the qualitative NFRs and the quantitative NFRs is within the efforts we will take in the next step.

ACKNOWLEDGMENT

This work is supported financially by the Nation Basic Research Program (973 Program) of China (grant No. 2009CB320701).

REFERENCES

[27] M Glinz, “Rethinking the notion of non-functional requirements,” in Third World Congress for Software Quality, Munich, Germany, 2005, pp. 55-64.