Use Constraint Hierarchy for Non-functional Requirements Analysis

Ying Guan and Aditya K. Ghose

Decision Systems Lab, School of IT and Computer Science
University of Wollongong, NSW 2522, Australia
{yg32,aditya}@uow.edu.au

Abstract. Non-functional requirements are critical in web engineering applications, but often ignored. Usually, these are articulated as statements of objectives, as opposed to prepositional assertions. A key challenge in dealing with objectives is that there is no obvious means of deciding when they are satisfied. In effect, these objectives are never fully satisfied, but *satisficed* to varying degrees. Alternative design decisions need to trade-off varying degrees of satisfaction of potentially mutually contradictory non-functional requirements. The key contribution of this paper is the use of the hierarchical constraint logic programming framework [3, 6] in dealing with non-functional requirements. We show how NFRs can be formulated as soft constraints and how the machinery associated with constraint hierarchies can be used to evaluate the alternative trade-offs involved in seeking to satisfy a set of non-functional requirements that might pull in different directions.

1 Introduction

Non-functional requirements [2] are concerned about the quality characteristics of a software system. NFRs are extremely important for the design of a system. Any errors, omissions, inaccuracies to take NFRs into account may cause unexpected failures of systems. Non-functional requirements are specially important in *web engineering* applications, but often ignored. Usually, NFRs are articulated as statements of objectives, as opposed to prepositional assertions (that evaluate to true or false). For example, the stakeholders may want the security levels of the system to be high, the performance be also high, and the cost be maintained as low as possible, etc. A key challenge in dealing with objectives is that there is no obvious means of deciding when they are satisfied. In effect, these objectives are never fully satisfied, but *satisficed* to varying degrees. Alternative design decisions need to trade-off varying degrees of satisfaction of potentially mutually contradictory non-functional requirements.

The key contribution of this paper is the use of the hierarchical constraint logic programming framework [3, 6] in dealing with non-functional requirements. Constraint logic programming was developed to extend the ability of traditional logic programming to deal with knowledge (facts and rules) expressed as Horn clauses with specially designated *constraint predicates.* The resulting systems were more efficient than standard logic programming systems because of their ability to use to special-purpose *constraint solvers,* which, in effect, understood the "meaning" of the constraint predicates, and dealt with them in more efficient ways than the resolution proof procedure that most logic programming systems relied on. Constraint logic programming also offered better expressivity. Hierarchical constraint logic programming (HCLP)

D. Lowe and M. Gaedke (Eds.): ICWE 2005, LNCS 3579, pp. 104–109, 2005.

was developed to deal with the fact that many of the constraints articulated by users in real-life problems are *soft constraints,* i.e., these were constraints that one would ideally seek to satisfy, but which could be violated (or satisfied to a lesser degree) if absolutely necessary. Soft constraints typically have varying degrees of priority, hence the HCLP framework permits the specification of *constraint hierarchies,* i.e., sets of soft constraints labelled with varying degrees of priority. Our larger project seeks to deploy the full capability of the HCLP framework in dealing with non-functional requirements. In the current paper, for the sake of brevity, we only focus on the constraint hierarchy component of framework. Our focus is on showing how NFRs can be formulated as soft constraints and how the machinery associated with constraint hierarchies can be used to evaluate the alternative trade-offs involved in seeking to satisfy a set of non-functional requirements that might pull in different directions.

The rest of this document is organized in the following manner. In Section 2, we give an overview of non-functional requirement. In section 3, we detail the constraint hierarchy level. In section 4, an example is given to illustrate how to apply our proposed method to solve conflicts among NFRs of software requirements and section 6 is the conclusion.

2 Non-functional Requirements

A key challenge in dealing with NFRs is articulating them in terms of metrics, on which one could then apply thresholds or seek to maximize or minimize. In Table1, we list possible measures for some NFRs, along the lines of the proposal in [1]. Those possible metrics would permit us to formulate constraint-style representations of NFRs. In this table, we only list part of those attributes that are easy to be specified using numbers, while there still exist other attributes that are difficult to be expressed in explicit numeric way, for instance, reliability, portability, etc. We believe that quantitative metrics for these can also be developed in the future, adding strength to our proposal.

The purpose of the NFRs level in our approach is to capture and represent non functional requirements. The structure for this level is defined below:

Definition 1. Non-functional requirement level is described as a tuple:
NFRL = < Q, A> where:

- Q is set of non-functional requirements,
- A is set of quality factors associated with each non-functional requirements in Q.

3 Constraint Hierarchy Level

After requirement elicitation, designers usually need to express preferences on each non-functional requirements of this system as well as on functional requirements. Preferences on non-functional requirements can be specified as soft constraints and the functional requirements can be expressed as hard constraints. In this level we need to formalize soft constraints for quality factors specified in NFRs level. In order to make those constraints computable, we choose a proposed mechanism hierarchical constraint hierarchy [3,6]. Constraint hierarchies (CHs) belong to traditional frameworks for handling of over-constrained problems. They allow expressing hard constraints which have to be satisfied and several preference levels of soft constraints which violations are minimized level by level subsequently [5].

Table 1. Possible Measures of some quality attributes

Quality Attribute	Possible Measures
Security	Time/effort/resources required, probability of detecting attack, percentage of services still available under denial-of-services attack; restore data/services; extent to which data/services damaged and/or legitimate access denied; resources needed for satisfying these demands; cost to satisfying these demands.
Efficiency	response time, miss rate, data loss, concurrent transaction number,
Availability	time interval when the system must be available, available time, time interval in which system can be in degraded mode, repair time, task time, number of problems solved
Cost	cost in terms of elements affected,effort,money;extent to which this affects other functions or quality attributes
Maintainability	cost in terms of number of elements afftected,effort,money
Accuracy	number of error ,rate of fail or successful operations to total operation, amount of time/data lost
Usability	use system efficiently, minimize impact of errors

Firstly, let us have a brief review of the hierarchical constraint logical programming. To introduce the constraint hierarchies, we use the definition of constraint hierarchies in [6]. A *constraint* is a relation over some domain D. A constraint is thus an expression of the form $p(t_1,...t_n)$ where p is an n-ary symbol in domain D and each t_i is a term. A *labeled constraint* is a constraint labeled with a strength, written lc where c is a constraint and l is a strength. In a constraint hierarchy, the stronger a constraint is, the more it influences the solution of the hierarchy. A *constraint hierarchy* is a finite set of labeled constraints. And in the same level, weight can be used to determine which constraint is more important. A *valuation* for a set of constraints is a function that maps free variables in the constraints to elements in domain D over which the constraints are defined. A *solution* to a constraint hierarchy is such a set of valuations for the free variables in the hierarchy that any valuation in the solution set satisfies at least the required constraints. An error function $e(c\theta)$ is used to indicate how nearly constraint c is satisfied for a valuation θ. CHs define the so called comparators aimed to select solutions (the best assignment of values to particular variables) via minimizing errors of violated constraints. Currently, there three groups of comparators: global, local and regional comparators. For a local comparator, each constraint is considered individually, for a global comparator, the errors for all constraints at a given level are aggregated using combining function g. For a regional comparator, each constraint at a given level is considered individually. There are a number of comparators by defining the combining function g and the relations <>g and <g for each (the symbol <> means equal). Global comparator includes weighted-sum-better, worse-case-better and least-squares-better.

After the quick review about CH, now we can introduce constraint hierarchy level. NFR level states quality factors of the system, while not all variables relating to quality factors can be assigned values. So the variables of constraint hierarchy is a projection on the whole set of quality factors. In table 1, we have listed some possible measures for those quality attributes that are easy to be measured using hard numbers. Then for each non-functional requirement, they have a set of quality factors that have been

assigned with values. And each non-functional requirement has its label assigned manually by stakeholders. Therefore, all NFRs and quality factors associated to them could compose a constraint hierarchy.

Before comparing valuations, we need to identify a prefer solution in constraint form. For example, we may set the prefer cost to be 0 and response time to be 0, although they cannot be satisfied. This solution can be used as a method to compare the proposed solutions after the computation of the hierarchical model, usually, solution with the smallest distance from this predefined solution is set as the best solution.

Now we can give the comparison process for constraint hierarchy: 1) For each quality factor, choose the value from a constraint about it in the highest label level; 2) if cannot find the solution after step1, then choose the value from a constraint about it with the highest weight; 3) if cannot find the solution after step2, then use comparator to choose the value from a constraint about it which has the smallest error sequence; 4) I if cannot find the solution after step3, then compare values from constraints with the value of this quality factor from prefer solution, choose the value with shortest distance.

4 Case Study

This section briefly illustrates how the approach we proposed can be applied, through a case study of the analysis and design of a web-based financial trading system, Financial bundle trading system (FBTS) [5].

FBTS is a web-based continuous electronic market that traders can use to execute bundle orders. With a bundle order, a trader can order a combination of stocks or assets. FBTS is an automated, continuous auction market that executes bundle orders to buy and sell.

The main non-functional requirements for FBTS could be elicited as: *Security, Efficiency, Cost, Maintainability, Usability* and *Accuracy.* The structures for each non-functional requirement are listed in table 2; CH is listed in table 3.

Table 2. Non-functional Level of FBTS

Q	A
Security	Response time, resources needed for satisfying these demands; cost in terms of elements affected.
Efficiency	response time, error rate, transaction speed, number of possible concurrent transaction, cost in terms of elements affected
Accuracy	rate of fail or successful operations to total operation, rate of data lost, cost in terms of elements affected
Usability	rate of impact errors
Cost	budget cost
Maintainability	recovery time, repair time, cost in terms of elements affected

A prefer solution for this constraint hierarchy is the values of cost, response time, error rate, recovery time, repair time and rate of fail operations are 0 respectively, and transaction speed and concurrent transaction are 5000 respectively. Based constraint hierarchy stated in table 2, a simpler form of constraint hierarchy can be generated.

Table 3. Constraint hierarchy for FBTS

Q	Label	Weight	A	Constraint
Security	Strong	1	Response time (rt), resources needed for satisfying these demands; cost in terms of elements affected (c).	rt = 1s c > 21,000
Efficiency	Strong	1	response time (rt), error rate(er), transaction speed(ts), number of possible concurrent transaction (ct), cost in terms of elements affected (c)	rt= 0.5s er < 0.01% ts > 1000/min ct >1000 c >25,000
Accuracy	Strong	0.8	rate of fail or successful operations to total operation(fr), rate of data lost(rdl), cost in terms of elements affected(c)	fr < 0.01% rdl< 0.01% c > 22,000
Cost	Medium	1	budget cost(c)	c < 20,000
Maintain-ability	Weak	1	recovery time(ryt), repair time(rpt), cost in terms of elements affected(c)	ryt < 1 min rpt < 2min c > 15,000

Strong rt=1, c>21,000
Strong rt=0.5, ts>1000, ct>1000, c>25,000, er<0.0001
Strong fr<0.0001, c>22,000, rdl<0.0001
Medium c<20,000
Weak c>15,000, ryt<1, rpt<2

From the above constraint hierarchy, we can see that there exist conflicts for *cost (c)* in *Strong* level and *Medium* level and for attribute *response time (rt)* in *Strong* level. In this constraint hierarchy, there is no *required* constraint, so *Strong* is the biggest strength. Correspondingly, for attribute cost, we only consider the value in the *Strong* constraints. At *Strong* level, there is still the *response time* conflict. The weights for these two constraints are equal; we cannot depend on weight to select the solution. So, we choose comparator to compare these two solutions. If we choose response time greater than 1 second, the error sequence is [[0],[0.5]], while if we choose response time being 0.5 second, the error sequence is [[0.5],[0]]. We still cannot get the better solution. Finally, we compare the distance between these two solutions and the prefer solution predefined, and the solution with response time has the shorter distance, so *rt* 0.5 is chosen. The best solution is *rt < 0.5, ts > 1000, ct > 1000, c> 25,000, ryt< 1, rpt < 2, er < 0.0001, fr < 0.0001.*

5 Conclusions

In this paper we have proposed a meta-level framework that can be used to detect and solve potential conflicts among non-functional requirements by constructing constraint hierarchy based on all possible quality factors relate to the prospective system. With the constraint hierarchy and the selection steps stated in section 3, possible solutions can be generated after the comparisons among constraints. Our proposed approach only focuses on conflicts that might arise among the cooperation between different non-functional requirements with the assumption that conflicts among stakeholders

have already been eliminated after negotiation. In this paper, we only focus on those non-functional requirements that are easy to be specified using hard numbers, while there are still many other non-functional requirements, such as, reliability, portability, etc., which are not mentioned in this paper. These will be remained as future work.

References

1. Bass Len, Paul Clements, Rick Kazman: Software architecture in practice, Boston, Addison-Wesley, c2003
2. Chung, L., Nixon, B., Yu, E., and Mylopoulos, J.: Non-Functional Requirements in Software Engineering, Kluwer Academic Publishing, 2000.
3. Molly Wilson, Alan Boring: Hierarchical Constraint Logical Programming, Journal of logic programming,1993
4. Ming Fan, Jan Stallaert, Andrew B. Whinston: A Web-Based Financial Trading System, Computer archive, Volume 32, Issue 4 (April 1999), Pages: 64 - 70, 1999
5. Hana Rudová,: Constraint Satisfaction with Preferences, Ph.D. Thesis,2001
6. Molly Wilson: Hierarchical Constraint Logic Programming, Technical Report 93-05-01, University of Washington, May 1993. (PhD Dissertation)