Using Constraint Programming for the Automatic Detection of Conflicts in Quality Requirements*

Antonio Ruiz–Cortés, Amador Durán, Rafael Corchuelo and Miguel Toro

E.T.S. Ingenieros Informáticos. Universidad de Sevilla
Avda. de la Reina Mercedes s/n, Sevilla 41012. Spain
aruiz@lsi.us.es, http://tdg.lsi.us.es

Abstract

Requirements negotiation is quite an interesting, ongoing research area. Current requirements engineering models usually propose a negotiation process with similar methods and goals. Unfortunately, only a few have partial automatic support. In this paper, we revisit one of the most mature models, Boehm's Win–Win model. Win–Win is a qualitative, process–oriented model so that it is specially suited to be used at the early stages of requirements engineering, when knowledge about requirements is still vague, but not for quantitative, product–oriented contexts where a more precise, exact knowledge about the requirements is needed.

In this paper, we present a proposal to extend and refine Win–Win in order it can be used in product–oriented contexts. The main benefit of our approach is that the same conceptual model for requirements negotiation can be used during all software development process, instead of using different models in different phases.

Keywords: requirements negotiation, quality requirements.

1 Introduction

Boehm's spiral model for requirements engineering (RE) is one of the most widely known in the literature [1, 2, 17]. According to [6], it can be classified as a qualitative, process–oriented method because it is intended to be used at the early stages of requirements engineering, when the requirements are still vague, and because conflict identification is oriented towards reaching decisions about the architecture of a system.

Win–Win is supported by QARCC (Quality Attribute Risk and Conflict Consultant), a tool that can identify conflicts in quality requirements (QR) by using an auxiliary knowledge base [3]. Turning the detection of conflicts into an automated process is interesting to requirements engineers, managers of applications based on web services [20, 21, 23], developers of agent societies [15], and so on.

*The work reported in this article was partially funded by the Spanish Interministerial Commission on Science and Technology under grant TIC 2000–1106–C02–01

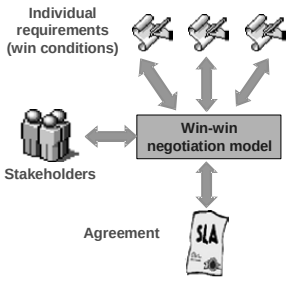

Figure 1: Overall view of the Win–Win requirements negotiation process.

The approach presented in this paper is based on the idea that some problems addressed by quantitative, product–oriented approaches can also be addressed by extending the conceptual framework behind Win–Win. The main benefit of our approach is that the same conceptual model for RE can be used during the whole software development process, instead of using different models. Our proposal relies on two conjectures: (1) every QR can be formally specified as a mathematical constraint, and (2) the requirements engineer is able to elicit, totally or partially, the flexibility degree of stakeholders on QRs before the negotiation process can be started. If we assume these conjectures are true, our proposal allows (i) to detect conflicts more precisely, and (ii) to take provider stakeholders into account.

The rest of the paper is organized as follows. In Section 2, we present an overview of the Win–Win model; next, our approach for enhancing Win–Win is presented in Section 3; finally, we present some conclusions in Section 4.

2 The Win–Win negotiation model

The Win–Win model [1, 2, 17] is based on Theory W [4], whose main motto is "make everyone a winner". Figure 1 illustrates this process. In projects with multiple stakeholders, conflicts amongst win conditions, i.e. individual requirements, arise frequently. When a conflict is detected, all related stakeholders are reported to propose options to solve the conflict. Once all conflicts are solved, it is said that an agreement is attained because the win conditions of which it is composed satisfy everyone's needs.

2.1 Win conditions

A win condition is a stakeholder's requirement considered important and beneficial. Formally, if space R contains all possible requirements specifications, a win condition can be viewed as a constraint on R that splits it into mutually exclusive subsets of requirements specifications that do or do not satisfy the win condition. The set of win conditions associated with a given stakeholder defines a win region, as depicted in Figure 2. Win conditions and win regions are formally described using set theory [17], so that the win region corresponding to i^{th} stakeholder is defined as:

$$W_i = \bigcap_{j=1}^{n} R(w_{i,j})$$

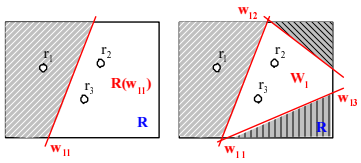

Figure 2: Graphical interpretation of both a win condition and a win region.

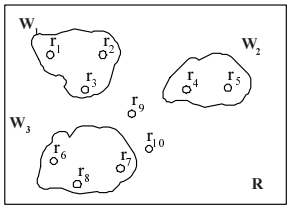

Figure 3: Graphical interpretation of a conflict amongst three stakeholders.

where $R(w_{i,j})$ represents the set of requirements which satisfy the j^{th} win condition of the i^{th} stakeholder, i.e.,

$$R(w_{i,j}) = \{r \in R \mid r \text{ satisfies } w_{i,j}\} \tag{1}$$

The main problem with this definitions is that Boehm's proposal lacks a systematic, formal way to express requirements and win conditions. Instead, natural language is used, which complicates determining if a requirement satisfies a win condition.

2.2 Conflicts

When the intersection of the win regions of two or more stakeholders is empty, it is said that there is a conflict amongst them, as illustrated in Figure 3. Given a set of win conditions, different conflicts may arise, and each one involves a different conflicting group of win conditions, denoted by I_k, that satisfies

$$\bigcap_{w_{i,j} \in I_k} R(w_{i,j}) = \emptyset \tag{2}$$

For instance, assuming that a client's win conditions are given by $w_{1,1}$ = "The budget shall not exceed € 6000" and $w_{2,2}$ = "The system shall be interoperable through CORBA 2.2"; also assuming that an architect's win conditions are $w_{2,1}$ = "The budget shall be € 7000" and $w_{2,2}$ = "The system shall use SOAP". In such a case, two conflicts would arise, being $I_1 = \{w_{1,1}, w_{2,1}\}$, and $I_2 = \{w_{1,2}, w_{2,2}\}$.

The main problem with this approach is, again, the lack of formality, which implies the detection of conflicts is error-prone, and may easily lead to situations in which a conflict is not detected at the appropriate time.

2.3 Options and agreements

Once conflicts have been detected, they are issued to the corresponding stakeholders so that negotiation can be started. Solving a conflict necessarily requires one stakeholder, at least, to

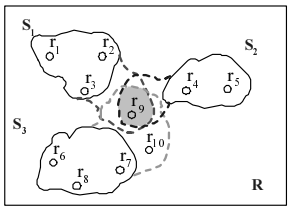

Figure 4: A graphical interpretation of the weakening process.

relax or weaken some of his or her win conditions, which implies that he or she has to yield in its original conditions to reach an agreement, as illustrated in Figure 4. Thus, an option is defined as an alternative that is proposed to resolve a conflict. For a conflict I_k, an option O_k is a proposed relaxation of the set of win conditions involved in the conflict. An option extends a win region (W) to a satisfactory (S) region, as depicted in Figure 4.

In general, solving all the conflicts arisen during a negotiation may be carried out by combining different groups of options. Each combination leads to a different agreement, denoted by A_k, which satisfies

$$\bigcap_{w'_{i,j} \in A_k} R(w'_{i,j}) \neq \emptyset$$

where $w'_{i,j}$ denotes either a weakened win condition or a primary one.

2.4 Automated detection of conflicts

As previously mentioned, the semiautomatic conflict detection implemented in QARCC relies on a knowledge base (KB), which stores well–known conflicts. Table 1 shows a partial view of a KB in which we state that there exist several strategies to endow the architecture of a particular system with a given feature. Each strategy establishes different relationships between the desired quality attribute and the rest.

The relationships between quality attributes are classified as either conflicting or cooperative. Two attributes are said to be conflicting if the value they have cannot be improved or worsen simultaneously. For instance, a possible strategy for portability is layering, which reinforces interoperability and reusability, and may conflict with the cost, schedule or performance attributes: if you attempt to increase portability, this may have a negative impact on performance, and the project might become more costly, and the initial schedule might not be met; however, if you do not worry about portability, the system might run faster, more effectively, and the project might be terminated in less time with smaller costs.

From this point, the identification process works as follows: QARCC is triggered when a stakeholder enters a new condition; it then checks if its quality attributes conflict with the QRs related to previous conditions; if so, a potential conflict has just been detected and the stakeholders must negotiate (c.f. Section 3.3).

Table 1: Quality–attribute strategies and relations (taken from [3]).

Primary attribute	Architecture strategy	Cooperative attributes	Conflicting attributes
Assurance	Input checking	Interoperability, usability	Cost, schedule, performance
	Redundancy		Cost, schedule, performance, evolvability, usability
Interoperability	Input checking	Assurance, usability	Cost, schedule, performance
Evolvability, portability	Layering	Interoperability, reusability	Cost, schedule, performance

3 Our proposal

In this section we show some drawbacks that prevent Win–Win from addressing some problems that are usually solved by quantitative, product–oriented methods; we also enhance the model so that it can deal with such problems.

3.1 Specification of quality requirements

In order to avoid different interpretations of requirements by different stakeholders, they need to be specified precisely and unambiguously. In Win–Win, win conditions are not usually expressed in a rigorous way. For instance, "documents must be delivered in multiple formats" or "developer training must be inexpensive" are examples found in [13]. The precision of these win conditions can be enough for detecting potential conflicts such as "the more formats needed, the more expensive training is", but they are clearly insufficient in general, and the lack of formality may lead to costly errors.

The basic assumption in our approach is that every QR can be both interpreted and specified as a mathematical constraint. The main reason supporting this assumption is that a QR can always be expressed in natural language as a constraint over a set of quality attributes (QAs) [7, 11, 17, 9] that usually range over domains such as integers, booleans, reals, or enumerates, and the relationships amongst them are usually expressed in terms of arithmetic operators. In [6], the authors present a list that includes over 150 quality attributes, and none of them is out of this scope. Our experience [8, 20, 22, 21] also suggests that this conjecture may be considered valid in an ample variety of systems, and it allows us to interpret requirements as if they were constraint satisfaction problems (CSP). This implies we can prove or refute properties automatically by using well–known constraint solvers.

For instance, if we want to express performance QRs, we can consider two attributes: the time to failure (TTF) over the real domain $[0, +\infty)$, and the time to repair (TTR) over the real domain $[0, +\infty)$. Then, we can specify the QR "time to failure must be 100 hours at least and time to repair must be under 1 hour" as the mathematical constraint $TTF \geq 100 \wedge TTR \leq 1$.

This approach can also be applied to win conditions. For instance, the win condition informally expressed in [13] as "documents shall be delivered in multiple formats" can be formally expressed as $\{$ [illegible] $\} \subseteq FMT$, where FMT represents the set of available formats. Furthermore, the win condition "developer training must be inexpensive" can be expressed as [illegible] $\leq CD/10$, if a developer training cost ([illegible]) under 10% of total development

cost (DC) is considered inexpensive.

Mathematical constraints allow to capture the semantics behind a requirement, and thus allows the automatisation of the tasks that we present in the following section.

3.2 Checking properties automatically

3.2.1 Consistency

The simplest property of a QR that can be interpreted as a CSP is consistency. A QR is consistent as long as its description does not contain any contradiction. Formally, requirement r is consistent if and only if its associated constraint r_c is satisfiable:

$$r \text{ is consistent} \Leftrightarrow sat(r_c) = true$$

where sat denotes the satisfiability function associated with the constraint solver used. sat takes a constraint c and returns one of the following values: $true$ if c is satisfiable, $false$ if it is not, and $\perp$ if it is not possible to determine whether c is satisfiable or not. For instance, CLP($\Re$) [14] is not able to determine if the constraint $xy < 7$ is satisfiable because it can only deal with (semi-) linear constraints. Thus, $sat(xy < 7)$ would return $\perp$ if CLP($\Re$) was used as the underlying solver. On the contrary, it would return $true$ if we used ILOG Solver [12].

We have not restricted the kind of constraints we can use to express QRs, thus offering the maximum expressiveness, but no general constraint solver able to solve any constraint exists. Notice that requirement r is considered to be inconsistent if $sat(r_c) = \perp$, which implies we analyse it from a conservative point of view. This might be wrong, but if it is not possible to solve all needed constraints, our approach will not lead to an inconsistent system. Anyway, most problems we have faced do only require linear constraints that can be easily solved by using the well–known SIMPLEX method [5]. Thus, our decision does not substantially reduce the applicability of our proposal.

3.2.2 Satisfiability

Another interesting property of QRs is satisfiability. A QR satisfies another QR when the constraint formed by the conjunction of constraints from both QRs is satisfiable, i.e.,

$$r_1 \text{ satisfies } r_2 \Leftrightarrow sat(r_{c_1} \wedge r_{c_2}) = true$$

This definition allows to define the operational semantics of the win condition regions in Win–Win. Notice that no precise definition of the semantics of this predicate have been provided so far, despite the fact that it constitutes the core of Win–Win.

3.2.3 Conformance

The definition of satisfiability presented in previous section was also adopted by the Object Management Group (OMG) for the definition of their trading service [19], and by the automatic negotiation models used in distributed agent platforms [15]. This approach is very useful when the semantics of a win region is the same for all stakeholders. We refer to this as customer semantics.

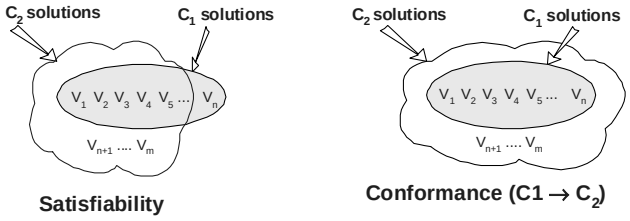

Figure 5: Graphical interpretation of both satisfiability and conformance.

Nevertheless, there are situations in which the semantics of a win region is not the same for all stakeholders. For instance, consider a situation in which a requirements engineer has elicited the win condition "the time to repair must be 60 minutes at most", i.e., $TTR \leq 60$, and the system architect states that "the time to repair will not exceed 70 minutes", i.e., $TTR \leq 70$. Using customer semantics, it is easy to prove that $TTR \leq 70$ satisfies $TTR \leq 60$. We might consider that $TTR \leq 60$ is an agreement for both win regions (c.f. Section 2.3). However, it cannot be considered satisfactory from a customer/provider point of view because if they are disconnected, i.e., they do not meet explicitly to clarify the semantics, the provider might supply $TTR \leq 60$, which does not satisfy our customer's needs.

In quality–aware distributed systems [10, 16] and applications based on web services [21, 20], a stronger notion of satisfiability, called conformance is proposed. If r_1 and r_2 are QRs, then r_1 is said to be conformant to r_2, denoted as $r_1 \rightarrow r_2$, if and only if the set of solutions to r_{c_1} is a subset of the solutions to r_{c_2}. This relationship is known as the implication constraint in [18], and it is defined as

$$r_1 \rightarrow r_2 \Leftrightarrow sat(r_{c_1} \wedge \neg r_{c_2}) = \text{if } a$$

3.3 Detecting conflicts automatically

The semi–automatic conflict detection proposed in [3], later enhanced in Section 2.4, is based on a relatively simple method. It is suitable to detect conflicts between win conditions at early stages of development, but it has two important drawbacks when compared to quantitative, product–oriented methods.

First, it does not address the quantification of the degree of conflict. This implies that conflicts detected automatically must be classified as potential conflicts, since it is possible to consider non–conflicting requirements as if they were in conflict. For instance, interoperability can be achieved with different solutions such as sockets, CORBA IIOP, W3C SOAP, and so on. These solutions decrease communication performance, but not to the same extent: SOAP is slower than sockets and it consumes more bandwidth; CORBA IIOP is slower than sockets, but faster than SOAP. Thus, it is possible that some conflicts exist if SOAP is used, but they might disappear if sockets were used.

Secondly, Win–Win needs all potential conflicts to be registered in a KB that must be kept up–to–date. Maintaining such a KB is not a trivial task because it becomes more and more error–prone and tedious as the number of quality attributes and relationships amongst them increases. Furthermore, registered conflicting relationships must be checked periodically because conflicts may change as time goes by. For instance, three–tier architectures are very popular in today's Internet world although they may have a negative impact on performance, but this is

not considered a serious conflict; ten years ago, however, performance was so crucial that such architectures were not popular at all, i.e., the conflict was more serious ten years ago.

Both problems may be avoided if we use constraints to specify QRs and interpret the detection of conflicts as checking satisfiability or conformance. In equation 2, the condition that helps requirements engineers detect that there is a problem was

$$\bigcap_{w_{i,j} \in I_k} R(w_{i,j}) = \emptyset$$

which can now be formulated as

$$sat(\bigwedge_{i=1}^{n} w_{c_{i,j}}) \neq true$$

where $w_{c_{i,j}}$ represents the constraint associated with the win condition $w_{i,j}$. Roughly speaking, a conflict exists if and only if it is not possible to prove the existence of a common solution to all win conditions. Notice that cases in which satisfiability cannot be proved are considered to be potential conflicts. We might be wrong, but this solution prevents conflicts from going unnoticed.

Obviously, this definition is valid as long as all stakeholders assume customer semantics. If some stakeholder are playing a provider role, the absence of conflicts can be formulated as follows:

$$sat(r_p \wedge \neg \bigwedge_{i=1}^{n} r_{c_i}) \neq true$$

where r_p represents the constraint associated with the provider stakeholder win region and r_{c_i} represents the i^{th} customer stakeholder win region.

4 Conclusions

In this paper, we have enhanced Win–Win so that it can be used in quantitative, product–oriented contexts. Our idea consists of associating mathematical constraints with each win condition, which allows to automate the detection of conflicts. Furthermore, we have added a new notion called conformance that allows to capture provider semantics. This proves that the world of quality requirements can be significantly enhanced if we use mathematical constraints to capture their semantics in a precise way.

References

[1] B. Boehm, P. Bose, E. Horowitz, and M.-J. Lee. Software requirements as negotiated win conditions. In *Proc. of the First Intl. Conference on Requirements Engineering, (ICRE'94)*, pages 74–83. IEEE Press, 1994.

[2] B. Boehm, P. Bose, E. Horowitz, and M.-J. Lee. Software requirements negotiated and renegotiation aids: A Theory–W based spiral approach. In *Proc. of the 17th Intl. Conference on Software Engineering (ICSE'95)*. IEEE Press, 1995.

[3] B. Boehm and H. In. Identifying Quality–Requirements Conflicts. *IEEE Software*, 12(6):25–35, Marzo 1996.

[4] B.W. Boehm and R. Ross. Theory–W Software Project Management: Principles and Examples. *IEEE Transactions on Software Engineering*, 15(7):902–912, 1989.

[5] K.H. Borgwardt. *The simplex method: A Probabilistic Analysis*. Springer–Verlag, 1987.

[6] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non–Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[7] A. M. Davis. The Analysis and Specification of Systems and Software Requirements. pages 119–144.

[8] A. Durán, B. Bernárdez, A. Ruiz-Cortés, and M. Toro. A requirements elicitation approach based on templates and patterns. In *Proc. of the 2^{nd} Workshop on Requirements Engineering (WER'99)*, pages 17–29, Buenos Aires, Argentina, 1999.

[9] X. Franch. Systematic formulation of non–functional characteristics of software. In *Proc. of the International Conference on Requirements Engineering (ICRE'98)*, Colorado, USA, April 1998. IEEE Press.

[10] S. Frolund and J. Koistinen. Quality–of–Service Specification in Distributed Object Systems. *Distributed Systems Engineering Journal*, 5(4), 1998.

[11] T. Gilb. *Principles of Software Engineering Management*. Addison–Wesley, 1988.

[12] ILOG. *ILOG Solver 4.0. User's manual*. 1997.

[13] H. In, D. Olson, and T. Rodgers. Multi–criteria preference analysis for systematic requirements negotiation. In *Proc. of the IEEE International Computer Software and Applications Conference (COMPSAC'2002)*, 2002.

[14] J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[15] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[16] J. Koistinen and Seetharaman. Worth–based multi-category quality–of–service negotiation in distributed object infrastructures. In *Proc. of the Second International Enterprise Distributed Object Computing Workshop (EDOC'98)*, La Jolla, USA, 1998.

[17] M. J. Lee. *Foundations of the WinWin Requirements Negotiation System*. Phd. thesis, University of Southern California, August 1996.

[18] K. Marriot and P.J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

[19] OMG. Trading object service specification. Technical report, Object Management Group, 2000. Version 1.0.

[20] A. Ruiz-Cortés, R. Corchuelo, and A. Durán. An automated approach to quality-aware web applications. In *Proc. of the 4^{th} International Conference on Enterprise Information Systems (ICEIS'2002)*, pages 995–1000, Ciudad Real, Spain, April 2002.

[21] A. Ruiz-Cortés, R. Corchuelo, A. Durán, and M. Toro. Automated support for quality requirements in web-services-based systems. In *Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, pages 48–55, Bologna, Italy, November 2001. IEEE Press.

[22] A. Ruiz-Cortés, R. Corchuelo, R.M. Gasca, and M. Toro. Aplicando técnicas de satisfacción de restricciones para comprobar la conformidad de aplicaciones WEB. In *Actas de la Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA'2001)*, pages 531–540, 2001.

[23] S.Y.W. Su, C. Huang, and J. Hammer. A replicable web–based negotiation server for e–commerce. In *Proc. of the 33rd Hawaii International Conference on Systems Sciences*, pages 1–8. IEEE Press, 2000.