Analysis of quality dependencies in the composition of software architectures

Javier F. Briones, Miguel de Miguel, Juan Pedro Silva, and Alejandro Alonso

Universidad Politécnica de Madrid
{jfbriones, mmiguel, aalonso, psilva}@dit.upm.es

Abstract. A dependable system has to meet some quality criteria in order to provide certain reliance on its operation. The quality of a system depends on the complex composition of the quality of its subsystems. Specifications of non-functional properties are commonly used to describe provided quality, required quality, resource usage and resource availability. Enclosing these specifications along with architectural models allows performing preliminary quality assessments (design-phase analysis). We allow configuration choices for quality specifications to represent design choices, deployment choices, or component adaptability. We focus on a composition study that answers to: is it possible to meet system and subsystem requirements given provided qualities? and, which configuration allows satisfying the requirements? The contributions of this work are: *i*) to formalize the composition based on quality levels and constraints, *ii*) to analyze existing quality dependencies in such a composition, *iii*) to show how we represent and evaluate dependencies in a model-driven environment.

Key words: non-functional properties, preliminary assessment, software architecture, constraint satisfaction

1 Introduction

Dependable systems are those on which reliance can justifiably be placed on the functionality they deliver. Engineering systems to the highest reasonably practicable standards of dependability is a great challenge. Reliance covers several aspects such as reliability, availability, safety and security; all of them can be characterized using non-functional properties. Engineers study non-functional properties of the system to estimate its probable dependability attributes and accordingly, recommend quality measures to increase reliance on the system (e.g., additional requirements). The study can be carried out at different granularity levels, but the earlier in the development is accomplished the more valuable the benefits. For instance, it could be applied during the design of the architecture taking into account system and subsystem extra-functional requirements; afterwards and throughout the refinement of the architecture, measures to improve system's quality are considered and allocated to the different subsystems and thus responsibilities broken down. Only when functional and non-functional

specifications of the whole system are completely defined and a rigorous development process is put in practice, enough confidence can be placed on the system being developed.

Let's consider the following three examples: *a*) a service to perform image processing may output data of different qualities according to the available processing and memory resources, *b*) a control system component have three operation modes each one working on different ranges of values for the input data, and *c*) a software driver for a motion-detection sensor whose accuracy depends on the hardware used existing two options available. Image quality, resource availability, operation mode, input data ranges, hardware characteristics and detection accuracy should be part of the quality specification.

We claim that preliminary quality assessments should be part of the rigorous process. The goal is to determine whether the system architecture will be able to deliver the required quality before starting the implementation. We focus on the composition of the different subsystems from a non-functional point of view. We formulate the composition problem as a configuration problem based on three main ideas: *i*) there are two types of quality specifications, offered and required constraints, that have to be matched together, *ii*) quality specifications are grouped into quality levels, *iii*) system configuration consists of selecting a quality level for every configuration point. Configuration points represent design choices (example *c*), deployment choices (*b*), or component adaptability (*a*). There is no difference at an early design-phase from the point of view of evaluating non-functional specifications. And this is the main goal, not only attaching non-functional specifications to model elements of the system architecture, but evaluating these specifications.

To explain the critical importance of the difference between offered and required constraints let's use a simple example. An entity can be set up according to two modes: the first one causes the value for certain non-functional property fr (a failure rate) to be in the range $(0, 0.01)$, the second one in the range $(0, 0.05)$. There is a required condition to be held: $fr < 0.03$. This required condition is satisfied for any value of the first range, but this is not true for the second range. If the entity is configured using the second mode, the value of the property could be lower than 0.03 but it might be greater and this is not admissible.

We think our work is a novel contribution in many ways. Previous work in quality of service management do consider quality levels to group specifications of non-functional properties but there is no offered-required matching. In the domain of web service procurement, they do find the best provider based on matching providers' specifications with the client's required specification; however, this matching is far more simple (two configuration choices: the first choice with only one level that includes only required constraints, and the second choice with several levels but each level only contains offered constraints). Additionally, our work is the only formalization aiming at evaluating specifications of non-functional properties, what should be a good step forward to leverage generic tools (instead of current ad-hoc developments) for analyzing

non-functional issues. A tool has already been implemented as proof of concept to evaluate architectural models.

After reviewing some related work, section 2 formalizes non-functional properties and quality constraints. Section 3 formulates the composition study based on levels and constraints. Section 4 analyzes quality dependencies and how we evaluate them within a model-driven environment.

1.1 Related work

Initial efforts dealing with non-functional aspects of composition have been undertaken in the web-service domain ([9, 12, 7]). They cover extra-functional aspects such as availability, security, response time, throughput and cost. Most of the approaches applied to web services rely on the dynamic view of the system and they are commonly based on the study of the system workflow and execution paths. They use techniques of QoS aggregation seeking to ensure SLAs since best-effort policies are not admissible for many applications.

In the domain of critical systems, the key issue found in the literature is to enforce contracts between subsystems. [1] remarks the importance of determining beforehand whether a given component can be used within mission-critical applications. A subsystem needs to provide a non-functional specification providing a kind of contract with its clients. [11] considers contracts as a modeling technology to foster the assembly of component-based applications, using case tools to check whether the contracts are fulfilled. For the authors, abstract contracts identified during the analysis phase are transformed in more concrete contracts to accommodate design, implementation, deployment and runtime phases; also new contracts can emerge as the development process advances. [10] takes this idea further promoting QoS-aware model transformations to gradually resolve QoS requirements in order to deliver efficient code. [4] presents the contracting system, ConFract, for an open and hierarchical component model, Fractal, that dynamically builds contracts from specifications at assembly time. It distinguishes three types of contracts: contract between client and server interfaces, external composition contract referring only to the external component interface to express usage, and internal composition contract to express assembly and internal behavior rules of the implementation.

Regarding QoS specification formalisms, QoS Modeling Language (QML [6]) and the UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms are the more popular. We [5] use an extended version of the profile complemented with the use of the Object Constraint Language.

In our work, we formalize the problem of composition of non-functional properties in a generic way, not being limited to a specific domain or to a specific dependability aspect. It does not consider contract/requirement transformations and deals only with a single modeling phase: the desing phase. It deals with the issue of matching offered and required quality constraints that, in our opinion, has not been properly covered yet.

2 Concepts formalization

2.1 Entities

We use the generic term *entity* to refer to the design-phase representation of a piece of the target system. Given that the study of non-functional properties can be done with high-level design models, the concept of entity could apply to software and hardware. It could apply as well to different software architecture and programming styles, in particular, component-based, service-oriented architectures or concurrent programming. An entity could be a hardware component, a software component, a software service or a significant function of the system specified from an analysis point of view (e.g., an operation, a task, a message passing). An entity can also be a resource with non-functional properties describing the quantity of resource, its availability or allowed access policies.

2.2 Non-functional properties

Non-functional properties (NFPs) cannot only be used to characterize dependability aspects (e.g., security, availability or safety) but also aspects not directly related to the implemented functionality (e.g., maintainability, adherence to standards and cost). It depends on the domain which NFPs are of interest; for instance, reliability and timing properties are of great important in the embedded systems domain. Numerous properties can be used as quality indicators (e.g., failure rate, throughput, latency) and most of them can be considered at different levels of granularity (e.g., system, component, service, operation). We use the term NFP to refer the development-time representation of a system quality property and not the value taken/measured once the system deployed.

A non-functional characteristic may be expressed using one or more properties. For example the characteristic "Latency" could include the following properties: arrival pattern, minimum period, maximum period, jitter, burst interval, burst size, requirement type, and output jitter. Each property has its own *domain*. For instance, maximum and minimum period are positive real numbers; burst size is a positive natural number; or requirement type one among {hard, soft, firm}. NFPs enable to deal with complex characteristics, like one that represent an aleatory variable and its characterize based on some of the parameters of the variable. There are several existing taxonomies of NFPs. One of them is the ISO general QoS architecture [ISO/IEC JTC1/SC7 N2059].

Let's consider I non-functional properties. Let x_{ij} be the variable representing the occurrence j of the non-functional property i and $\mathrm{Dom}(x_{ij})$ the domain of the variable. Let p_{ij} be a specific value taken for the variable x_{ij}. There might be several occurrences of a non-functional property when different entities take into account the same property. The domain of the non-functional property i does not need not to be the same for the different occurrences.

We also define the tuples $\mathbf{x}$, to represent the collection of non-functional properties in the system and $\mathbf{p}$, to represent an assignation of values to all the

properties being considered in the system. $\mathbf{p}$ could be e.g., an observation of the system.

$$\mathbf{x} = (x_{11}, \ldots, x_{21}, \ldots, x_{I1}, \ldots)$$
$$\mathbf{x} \in \mathrm{Dom}(x_{11}) \times \cdots \times \mathrm{Dom}(x_{21}) \times \cdots \times \mathrm{Dom}(x_{I1}) \times \ldots$$
$$\sigma(\mathbf{x}) = \mathbf{p} = (p_{11}, \ldots, p_{21}, \ldots, p_{I1}, \ldots)$$

2.3 Quality constraints

To design a system according to its specification, non-functional properties need to meet existing constraints. Some of the sources *quality constraints* arise are: *i*) service level agreements; *ii*) extra-functional requirements due to clients or environment; *iii*) explicit limitations in the value of non-functional properties (e.g., to increase system reliability); *iv*) specifications of QoS provided-interfaces; *v*) extra-functional requirements declared by a component/service; *vi*) relations between non-functional properties of the different system parts (e.g., inter-component dependencies); *vii*) quality transformations performed by a subsystem; *viii*) resource usage and resource availability; and any other system condition that necessary constrains the value of one or more non-functional properties.

Simple constraints might be represented as ranges (e.g., $[0, 0.001)$ and $[64\,\text{kbps}, \infty]$). More complex constraints need additional methods for their expression as they might imply, for instance, a dependency between non-functional properties. We will use *mathematical statements* to represent constraints. Statements may include, for example: comparisons $(\leq, <, =, >, \geq)$, arithmetical operations $(+, -, *, /)$, logical operations $(\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow)$, quantifiers $(\exists, \forall)$, operations over sets $(\in, \cup, \cap)$, functions $(\log, \max)$, variables of any domain (boolean values, real numbers, set of integers, set of literals) or constants (π, any value). They could also include any modal operator of a temporal logic (e.g., linear temporal logic and computation tree logic). Certainly, statements must be well-formed and the number and type of the operands involved in every operation has to be correct.

The variables of the mathematical statements are the non-functional properties. A statement does not need to involve all the existing non-functional properties, nor did it need to involve all the occurrences of a property. In those cases, some of the elements of the tuple $\mathbf{x}$ are not included. A simple example of constraint could be: "the value of the property 9 in the entity 1 must be greater than the value of the same property in the entity 2", represented with the first of the following statements.

$$A_1(\mathbf{x}) = A_1(x_{91}, x_{92}) \equiv x_{91} > x_{92}$$
$$A_2(\mathbf{x}) = A_2(x_{14}, x_{15}) \equiv (x_{14} > 3) \wedge (x_{15} > 2)$$
$$A_3(\mathbf{x}) \equiv \forall j (x_{4j} > \min_{j=1..J}(x_{7j}))$$

2.4 Offered and required constraints

We justified the existence of offered and required constraints in the introduction. An offered constraint characterizes a property whose value may fluctuate during operation provided that the offered constraint is met. That is, only the satisfaction of the offered constraint is ensured but not a steady value for the properties involved. On the other hand, a required constraint is a condition over the *values taken* by one or more properties. Offered constraints might imply that there is not a single value a property can take, and to check a required constraint we need to do it for every value a property might take. Summing up, we need to ensure that for every value that meets the offered constraint all the required constraints are satisfied. Both, required and offered can be expressed as mathematical statements, $R(\mathbf{x})$ and $O(\mathbf{x})$, but whether a statement represents an offer or a requirement gives a special nuance to the statement that changes equations and analyses. The relationship between offered and required constraints, in case there is only one of each type, is as follows:

$$\forall \mathbf{p} : O(\mathbf{p}) \Rightarrow R(\mathbf{p})$$

2.5 Quality levels and level choices

In many situations, a constraint cannot be set up independently. For instance, to set up an offered constraint there is a need to set up first a specific required constraint (e.g., to provide data of a specific precision there is a demand on the arrival data rate). A *quality level* is the set of offered and required constraints that need to be set up together. We use the variable y_j to represent a set of alternative levels and l_j for a selected level in the choice. l_j is a pair of sets, the first set includes the offered constraints in the level and the second one includes the required constraints. When there are more than one choice, we use the tuples $\mathbf{y}$ and $\mathbf{l}$ to collect choices and selected levels respectively.

$$\mathbf{y} = (y_1, \ldots, y_J)$$
$$\mathbf{l} = (l_1, \ldots, l_J) \qquad l_j = (l_j^O, l_j^R) = (\{O_{j1}(\mathbf{x}), \ldots\}, \{R_{j1}(\mathbf{x}), \ldots\})$$

We define the formalisms $\mathcal{O}(\mathbf{l}, \mathbf{x})$ and $\mathcal{R}(\mathbf{l}, \mathbf{x})$ to express the joint satisfaction of all the offered or required constraints from the selected levels.

$$\mathcal{O}(\mathbf{l}, \mathbf{x}) \Leftrightarrow \forall O(\mathbf{x}) \in \cup_{j=1..J}\ l_j^O : O(\mathbf{x})$$
$$\mathcal{R}(\mathbf{l}, \mathbf{x}) \Leftrightarrow \forall R(\mathbf{x}) \in \cup_{j=1..J}\ l_j^R : R(\mathbf{x})$$

2.6 Constraints on quality levels

Up to now, we have only considered constraints on non-functional properties. Nevertheless, constraints on level choices could be of use in many situations.

This type of quality constraint identify high-level dependencies. For instance, the activation of certain quality level by an entity does require some other quality levels to be operative, of the same entity or other entities. As we did with previous constraints, we express level constraints by means of *mathematical statements* $F(\mathbf{l})$ that may involve several quality level choices $\mathbf{l}$. A simple example of constraint on quality levels could be: "if level l_1 is selected then l_3 and l_5 must be selected", that could be represented with the following statement.

$$F = \{(y_1, y_3, y_5) \in \mathrm{Dom}(y_1) \times \mathrm{Dom}(y_3) \times \mathrm{Dom}(y_5) | (y_1 = l_1 \Rightarrow y_3 = l_3 \wedge y_5 = l_5)\}$$
$$F(\mathbf{y}) = F(y_1, y_3, y_5) \equiv (y_1 = l_1 \Rightarrow y_3 = l_3 \wedge y_5 = l_5)$$

3 Preliminary quality assessment

System engineers should define non-functional properties, quality constraints and quality levels. Only when this is done, they can carry out a composition study to check whether the different entities can be assembled from a quality point of view. The problem formulation is three-folded: *i*) is it possible to meet system and entity requirements given existing entities?, *ii*) which configuration allows satisfying the requirements?, *iii*) which of these configurations is the best one according to certain criteria? There are several concepts related to the the solution of the problem raised:

Configuration A configuration is a set up of alternative constraints/levels for system choices.

Admissible configuration It is a configuration that ensures that every required constraint in the configuration is satisfied. The collection of every admissible configuration s will be represented with S.

Optimal admissible configuration Admissible configurations could be compared if there exists a function $f(s)|_{s \in S}$ to compare them. If we suppose that the higher the function value the better the system configuration, the optimal admissible configuration s^* fulfills the following equation:

$$s^* \in S \mid \forall s \in S : f(s^*) \geq f(s)$$

3.1 Composition based on quality levels

In the composition scenario raised in this paper, offered and required constraints are grouped into quality levels. We aim at finding an admissible configuration i.e., a combination of levels $\mathbf{l}$ that makes the system composable. A system can be composed when for every $\mathbf{p}$ that meets the offered constraints of the selected levels, all the required constraints of the selected levels are satisfied. The set S includes every admissible configuration.

$$\exists \mathbf{l} \forall \mathbf{p} : \mathbf{F}(\mathbf{l}) \wedge (\mathcal{O}(\mathbf{l}, \mathbf{p}) \Rightarrow \mathcal{R}(\mathbf{l}, \mathbf{p})) \quad (1)$$

$$S = \{\mathbf{l} \mid \forall \mathbf{p} : \mathbf{F}(\mathbf{l}) \wedge (\mathcal{O}(\mathbf{l}, \mathbf{p}) \Rightarrow \mathcal{R}(\mathbf{l}, \mathbf{p})\} \quad (2)$$

4 Quality dependencies

We work in a *model-driven development* environment so we seek a way to include quality information (i.e., non-functional properties and quality constraints) along with system models, and the ability process that information in a model-driven way.

We use constraint satisfaction techniques to find admissible configurations. A constraint satisfaction problem (CSP) is defined as a triple $(\mathbf{x}, Dom(\mathbf{x}), \mathbf{C}(\mathbf{x}))$ where $\mathbf{x}$ is a set of variables, $Dom(\mathbf{x})$ is a set of domains, and $\mathbf{C}(\mathbf{x})$ is a set of constraints. The goal of a constraint satisfaction problem is defined as:

$$\exists \mathbf{p} \in Dom(\mathbf{x}) : \mathbf{C}(\mathbf{p}) \tag{3}$$

Due to the fact that everything is a constraint in a CSP, to model the composition problem as one of constraint satisfaction we need to find all the constraints that enable to match equations 1 and 3. Consequently, besides offered $O(\mathbf{x})$ and required $R(\mathbf{x})$ constraints, we have to find other constraints included in quality dependencies. This section deals with all the quality dependencies found in the composition problem.

The first approach to solve the composition problem was to use search algorithms and constraint checking. The solver is a tree algorithm that traverses the search space evaluating at each node those dependencies that can be evaluated because all the variables of the dependency are assigned. This solver requires a way to evaluate constraints, returning true or false, given an assignment for all the variables of the dependency. It does not implement constraint solving techniques so it does not evaluate constraints with unassigned variables; it is easier, having the simple mathematical constraint $x_{11} \geq x_{12} + x_{13}$, to evaluate the function for $(x_{11} = 6, x_{12} = 2, x_{13} = 3)$ than calculating the value of x_{11} for $(x_{11}, 2, 3)$.

4.1 Offered and required constraints

Offered and required constraints restrict the domain of allowed values for one or more non-functional properties. A constraint that relates two (or more) non-functional properties implies an *explicit dependency* between the properties e.g., a dependency between certain video quality property and a resource-usage property. This explicit dependency is represented by the inclusion of more than one property x_{ij} in a quality constraint $O(\mathbf{x})$ or $R(\mathbf{x})$. An example could be $R_1(imgFrame, mem1) \equiv mem1 >= 32 * imgFrame$. Quality dependencies can also be found when there are quality constraints on the same non-functional properties, and these constraints need to be satisfied together. For instance, if there is another constraint relating certain sound quality property and the same resource-usage property, there is an *implicit dependency* between video and sound quality. As a result, a component might be unable to deliver the highest quality for sound and video at the same time because there are not enough resources. This implicit dependency is represented by the joint satisfaction of constraints (tuple of statements) $\mathbf{O}$ or $\mathbf{R}$. In the example, we add the constraints

$R_2(sndSample, mem2) \equiv mem2 >= 8 * sndSample$ and $R_3(mem1, mem2) \equiv mem = mem1 + mem2$

We [2] are currently using the OMG standard "UML Profile for Modeling QoS and FT Characteristics and Mechanisms". We categorize non-functional properties creating what it is called a QoSCatalog. Required and offered constraints are modelled as QoSRequiredConstraint and QoSOfferedConstraint. Mathematical statements used to specify these constraints are defined in the Object Constraint Language. We reuse available OCL tools to evaluate constraints. We used the Eclipse UML OCL plug-in to evaluate OCL expressions against instances of elements of the QoSCatalog, so we have to create these instances prior to evaluation. Mathematical statements currently allowed are those that can be represented with OCL expressions and are well-formed expressions.

4.2 Quality negotiations

We call quality negotiation to the set of offered and required constraints on the same non-functional properties. We use the same term used in (run-time) adaptable systems however, our negotiation occurs at design-time and the negotiation broker is the composition tool. For a configuration to be admissible, every combination of NFP values that satisfy the offered constraints in the configuration has to meet required constraints. For instance, a required transmission rate greater than 2 Mbps is satisfied by an offered rate of 3 Mbps, but not by one of 1 Mbps. During the composition required qualities need to match offered qualities. A quality negotiation is a dependency that involves offered and required constraints on the same non-functional properties. This dependency is included in the entailment $\mathcal{O}(\mathbf{l}, \mathbf{p}) \Rightarrow \mathcal{R}(\mathbf{l}, \mathbf{p})$ that synthesizes the relationship between offered and required constraints. In the example, $R_{11}(tx) \equiv tx >= 2$, $O_{21}(tx) \equiv tx >= 3$, and $O_{21}(tx) \Rightarrow R_{11}(tx)$.

We cluster constraints programmatically that is, system engineers do not have to group related offered and required constraints. We use the QoSContext of the constraints, an element of the QoS Profile that identifies which properties are used by the constraint. To evaluate negotiations we check a set of required constraints against a set of offered constraints. This is not a standard use of OCL, so we had to create tool-support to allow this checking. The Eclipse UML OCL plug-in was reused to parse and handle OCL expressions. Our evaluator checks if for any NFP value satisfying offered constraints, the required constraints are satisfied. Currently, we are able to evaluate quality negotiations which include real and integer values and some operators ($+$, $-$, $>$, $<$, $\geq$, $\leq$, *and*, *or*). We are extending the number of types (string , enumeration) and operators ($=$, $*$, $/$). We need to study the viability of using other logical OCL operators (implies, forAll, exists). Our final goal is to evaluate negotiations including any well-formed OCL expression.

4.3 Quality functions and constraints on levels

A quality function describes how an entity internally behaves from a quality point of view that is, how output quality is obtained from input quality (e.g., resource-usage). Some entities do not need a quality function, e.g, resources; to model a network only its throughput should be specified as output quality and there is no input quality. As opposed to quality negotiations, quality functions need to be modeled by system engineers (typically by those who implement the subsystem). We do not restrict how this function is modeled on the condition that it is machine-processable e.g., *i*) a "low-level" statement relating input and output NFPs (using required quality constraints), *ii*) a constraint expression that needs to be satisfied by the quality levels selected (using constraints on levels). The first approach does not need any additional implementation as functions are expressed using required constraints. Nevertheless, we preferred to implement the second approach as well. The main reason for this choice is that quality functions are dependencies intra-entities and the developer of the entity has all the knowledge to characterize it.

To better explain the use of constraints on levels to characterize quality functions, let's use as example a video compression software component that is configurable with different algorithms and where selected algorithm and compression parameters are interrelated. Quality levels could be used to group: *i*) algorithms and algorithm configurations, *ii*) resource-usage, *iii*) and other compression parameters. The design could call "Best algorithm" to a level that includes the best algorithm in its best configuration, "Best compression rate" to a level including the highest compression rate, and "Most efficient" to a level including the most efficient resource-usage. For the component, only the "Best algorithm" provides the "Best compression" and is the "Most efficient". Other algorithms cannot provide these two levels together. Quality functions expressed as constraints on levels are explicit dependencies when more than one level y_j are included in the constraint $F(\mathbf{y})$. In the example, $F(y_1, y_2, y_3) \equiv y_2 = bestCompression \wedge y_3 = mostEfficient \Rightarrow y_1 = bestAlgorithm$.

Some add-ons to the QoS profile were needed to allow modeling configuration based on levels. For instance, there is a need to specify the choice that groups the quality levels in a configuration point(see [3]). As with quality negotiations, we had to create an evaluator to check level constraints. Quality functions do need to be evaluated for a specific system configuration as opposed to quality negotiations where required constraints were evaluated against offered constraints. The evaluator overrides the default OCL evaluator to have contextual information about the configuration that needs to be checked i.e., to check whether some levels are active within a system configuration.

5 Conclusion

This paper, after explaining some required concepts, formulates the composition problem based on quality levels by matching offered and required constraint. Problem solving is simplified by using choices with a finite number of quality

levels what enables the use of constraint satisfaction techniques. These techniques use search algorithms that are complete and optimal for the problem formulated. Search algorithms transform the problem into evaluating dependencies. This is the reason why quality dependencies were analyzed in detail. We have developed a model-driven tool to analyze quality composition and this paper provides some detail of how dependency evaluation has been implemented. One of the main reason to use a model-driven approach is that quality information is attached to system elements so consistency and traceability is achieved seamlessly.

How is the problem solved if quality levels are not used? We have studied how to solve the problem and there are different alternatives e.g., linear algebra, linear programming, mixed integer programming, non-linear programming or constraint satisfaction. All of the alternatives but constraint satisfaction highly depend on the shape of the constraints/statements. Constraint satisfaction externalize the evaluation of dependencies so that evaluators can be created for the kind of constraints existing in the system. *What kind of problems cannot be solved using constraint satisfaction?* Imagine a system composed by five subsystems and four of them have their non-functional properties defined. It is not possible to know the range of values allowed for the last subsystem with constraint satisfaction. A workaround would be to fake some quality levels and check quality composition but we still do not calculate the exact range. *What kind of problems cannot be solved using quality levels?* Imagine a subsystem that is able to offer any value of a range and each precise value signifies a different cost. For example, if a deterministic network protocol allows making reservations of any throughput rate and we fake a discrete number of levels, we could be wasting network bandwidth forcing a higher rate than needed (we needed 53 and we are obligated to 64 kbps).

As future work we want to allow including custom operators in OCL expressions. Users will be able to create evaluator for the desired operators and plug them into the tool. We are also exploring the use of constrain logic programming to add constraint solving techniques such as propagation and pruning.

References

1. Antoine Beugnard, Jean Marc Jézéquel, Nol Plouzeau, and Damien Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999.
2. Javier Fernández Briones, Miguel Angel de Miguel, Alejandro Alonso, and Juan Pedro Silva. Modeling quality of service adaptability. In *Enterprise Distributed Object Computing Workshops (Advances in Quality of Service Management), International Conference on*, volume 0, pages 50–27, Los Alamitos, CA, USA, 2008. IEEE Computer Society.
3. Javier Fernández Briones, Miguel Angel de Miguel, Alejandro Alonso, and Juan Pedro Silva. Quality of service composition and adaptability of software architectures. In *12th IEEE International Symposium on Object component service-oriented Real-time distributed Computing*, 2009.
4. Philippe Collet, Roger Rousseau, Thierry Coupaye, and Nicolas Rivierre. A contracting system for hierarchical components. In George T. Heineman, Ivica

Crnkovic, Heinz W. Schmidt, Judith A. Stafford, Clemens A. Szyperski, and Kurt C. Wallnau, editors, *Component-based Software Engineering*, volume 3489 of *Lecture Notes in Computer Science*, pages 187–202. Springer, 2005.

5. Miguel Ángel de Miguel. QoS modeling language for high quality systems. In *WORDS*, pages 210–216, Los Alamitos, CA, USA, 2003. IEEE Computer Society.
6. Svend Frolund and Jari Koistinen. Quality of services specification in distributed object systems design. In *COOTS'98: Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems*, pages 1–1, Berkeley, CA, USA, 1998. USENIX Association.
7. Michael C Jäger. *Optimising Quality of Service for the Composition of Electronic Services*. PhD thesis, Technische Universitt Berlin, 2007.
8. Jean Marc Jézéquel. *Model Driven Engineering for Distributed Real Time Embedded Systems*, chapter Real Time Components and Contracts. Hermes Science Publishing Ltd, London, 2005.
9. Daniel A. Menascé. QoS issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.
10. Arnor Solberg, Jon Oldevik, and Jan Oyvind Aagedal. A framework for qos-aware model transformation, using a pattern-based approach. In Robert Meersman and Zahir Tari, editors, *CoopIS DOA ODBASE 2*, volume 3291 of *Lecture Notes in Computer Science*, pages 1190–1207. Springer, 2004.
11. Torben Weis, Christian Becker, Kurt Geihs, and Noël Plouzeau. A uml meta-model for contract aware components. In *UML01: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 442–456, London, UK, 2001. Springer-Verlag.
12. Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.