Automated Support for Quality Requirements in Web–Service–Based Systems

Antonio Ruiz, Rafael Corchuelo, Amador Durán and Miguel Toro
Dep. de Lenguajes y Sistemas Informáticos, Universidad de Sevilla
Avda. de la Reina Mercedes, s/n. Sevilla 41.012, Spain
aruiz@lsi.us.es

Abstract

The automatic checking of quality requirements will play a fundamental role in the future market of web services. The reason is that it will allow to build economically–optimal systems whose quality level can be guaranteed. In this paper, we identify some of the main problems with which this kind of futures systems are going to be faced, and also propose a realistic proposal to solve them. The key point is to view quality requirements from a twofold perspective: a natural language sentence and a constraint on a quality attribute. Thanks to this principle, some of the classical disadvantages of formal methods may be overcome.

1. Introduction

The increasing demand and complexity of web applications, the need for reducing development and running costs, and the new opportunities that web services offer are the main reasons behind the forecast popularity of so called Multi–Organisational Web–based Systems (MOWS). In such a system, some organisations offer end–user web services such as banking, booking or information retrieval, whose functionality relies on a number of back–end web services provided by other organisations. In order to achieve optimal MOWS, they must be open systems that must be able to search for the services on which they rely automatically and dynamically. The final decision on which to use should obviously depend on the ratio quality/price [3].

In order to deal with MOWS, a new paradigm called WOP (*Web–Oriented Programming*) is being studied. The goal is to provide software architects with a number of tools to solve critical problems such as *optimality* and *conformance*, which are both closely related to the automatic management and negotiation of quality requirements. In [3], four extensions to current development methodologies and run-time platforms were identified in order to solve these problems:

1. Quality documents must be formalised so that they are understandable and allow for automatic checking. Furthermore, quality documents must include both QoS clauses on attributes such as delays, jitting or mean time to fails, and higher–level clauses on attributes such as cost per connection, security level or binding policies.
2. It is necessary to extend current architectural description languages so that they can take quality requirements into account.
3. It is necessary to extend current distributed run–time platforms such as CORBA, COM+ or .NET so that quality documents are first-class elements.
4. It is necessary to include quality monitoring mechanisms so that it is possible to check if the provider of a web service fulfills the quality level agreement it reached with a customer.
5. It is necessary to include authentication mechanisms that prevent both providers and clients from disowning something they have done.

In this paper, we tackle extensions 1, 2 and 3 and present a proposal for specifying quality requirements that offers and effective solution to checking optimality and conformance automatically. It improves on other proposals in the following aspects: i) it provides a notation to express quality requirements in natural language that facilitates formal reasoning and automatisation; ii) it allows for bilateral negotiation, so that both customers and providers can include negotiation clauses in their conditions and offers; iii) it allows to express optimisation criteria that help locating the

*This article was supported by the Spanish Interministrial Commission on Science and Technology under grant TIC2000–1106–C02–01, and by the CYTED project "WEST".

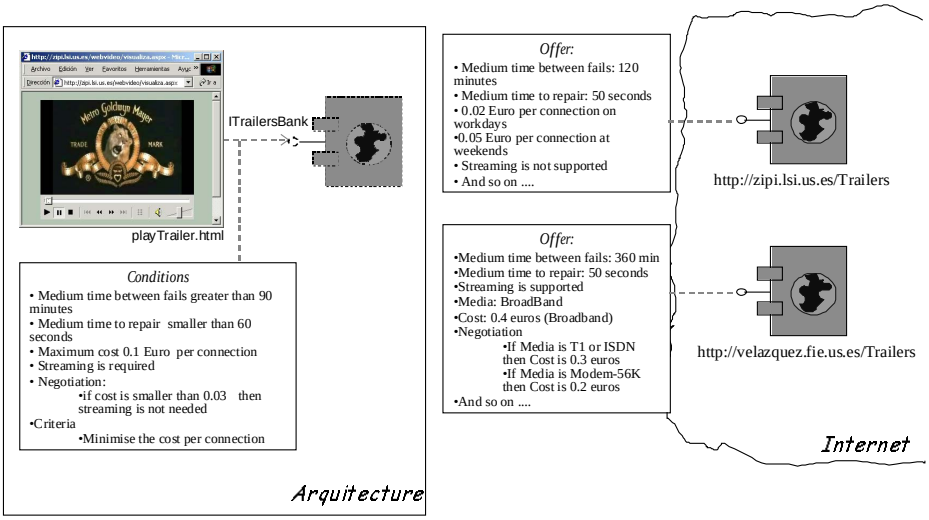

Figure 1. A fragment of the architecture of a web portal that offers films and trailers. It is annotated with quality documents in natural language.

web service that optimally fulfills a number of quality requirements; iv) it allows for temporal constraints, thus allowing for systems whose requirements may change over the time.

The rest of the paper is organised as follows: Section 2 presents some of the problems we tackle in the context of a simple but realistic MOWS; Section 3 describes the main features of QRL, a language that aims at providing software engineers with an adequate tool to express quality requirements; Section 4 sketches an experimental prototype that deals with the management of MOWS and offers a set of components to implement them; Section 5 compares our proposal with other authors' work, and some conclusions and future work are shown in Section 6.

2. Problems in the context of MOWS

2.1. A web video server

In this section, we present a simple MOWS in order to illustrate the main problems our proposal tackles. It is a quality–aware web portal that offers film trailers and aggregates three web services: an authentication service such as Microsoft Passport, a film store and a trailer store. Figure 1 shows a fragment of the software architecture. In order to show a trailer, we use an HTML page that relies on a web component that must implement interface `ITrailersBank`. This component is stereotyped using a world icon and drawn using a dashed line, which means that it is not bound when the system is set up, but requested when it is needed. Thus, any web service satisfying the quality constraints the portal requires may be used and bound at run time so that the system can select the best one of which it is aware each time a film is going be shown.

Hereinafter, we call *conditions* the document that describes the quality level the portal requires, and *offers* the one that describes the requirements a provider can fulfill.

2.2. Conformance

Conformance is defined as the ability to *guarantee* that the web services that compose a MOWS are adequate to satisfy its quality requirements. In this context, the word guarantee should not be understood as uninterrupted satisfaction of the requirements, but as a way to identify the organisation responsible for a failure if something goes wrong. In order to be able to guarantee conformance, it is necessary to check if the conditions for which a customer asks can be fulfilled with the offers of a potential provider.

Figure 1 shows two providers called `zipi` and `velazquez` that offer a service for storing trailers. Informally, it is easy to check that the offer by `zipi` does not satisfy the conditions of the portal because it cannot stream trailers; on the contrary, the offer by `velazquez` fulfill them completely.

Unfortunately, things are not so easy in a real Internet set, and checking for conformance cannot be done manually in most cases due to two main reasons:

- When the number of requirements is high, checking them all for conformance may consume a lot of time,

which may be unacceptable if a customer is waiting for a trailer to be displayed. Also, this process is error prone, and subject to ambiguity if requirements are only expressed using natural language.

- Both conditions and offers depend on criteria that are likely to change because the web is a dynamic scenario where requirements are evolving continuously. Thus, a human procedure is not likely to be successful in such a dynamic set.

2.3. Negotiation

The results of checking a web service for conformance is part of the document that describes both the responsibilities of our portal and the providers of the services on which it relies. In the context of application service providers (ASP), such a document is known as the *Service Level Agreement* (SLA) [1]. Unfortunately, it is not likely that an agreement may be reached immediately because customers always search for inexpensive, but highly quality services and providers always try to sell their high–quality services at higher costs than their low–quality ones.

Therefore, customers and providers usually have to negotiate on the final required quality. For instance, our web portal might agree on using a service that cannot stream trailers as long as the connection cost is not greater than 0.03 Euro. If this negotiation clause is added to the conditions, `zipi` might also be selected. Not only negotiation clauses are applicable on the customer side, but also on the provider side so that they can offer the same functionality at different quality levels and prices. For instance, `velazquez` has three different fares, depending on the speed at which a trailer is provided.

2.4. Optimality

Having an automatic mechanism for checking and negotiating conformance is necessary, but not enough if we are interested in finding a web service that optimally fulfills a set of conditions. In our example, after negotiating, both providers fulfill the required quality level, so any of them might be hired. However, there is also an optimality criterium in the conditions that specifies that the selected web service must minimise the cost, so that `zipi` should be selected according to this criterium. If this criterium had been the minimum time to fails, `velazquez` should have been selected, instead.

In general, optimality criteria are expressed as utility functions over the set of quality attributes in which we are interested. These functions are usually expressed as a linear combination of the quality attributes in which higher coefficients indicate that the attribute to which they applied must be maximised.

2.5. Temporal awareness

We are in no doubt that one of the most important features of a service is its price. In the mobile telephony world, temporal–aware fares was an important feature introduced some years ago, and it attracted an important number of users. In the world of web services, temporal fares are also very important, and both conditions and offers must be annotated with the period of time during which they are valid. For instance, in our example, it is specified that `zipi` charges its users with 0.02 Euro on workdays and 0.05 Euro at weekends.

Designing temporal–aware systems bring architects a lot of advantages because they can, for example, forecast the workload of their servers more accurately, which helps avoiding overload situations.

3. Our proposal

In this section, we briefly outline the main features of QRL (*Quality Requirements Language*), a formal specification language whose expressiveness and execution model allow to solve the problems we have presented previously.

3.1. Formalising requirements

In section 2, we made evident the need for automatising conformance, negotiation, optimality and temporal awareness. Roughly speaking, this implies that we need to transform quality requirements into a formal language with which we can reason automatically.

There are only a handful languages for expressing quality requirements, and even less languages with enough formal support so as to be able to proof properties such as optimality. QRL relies on mathematical constrains as a means to transform quality requirements expressed in natural language into a language that is amenable to formal reasoning. Thus, the problems we referred to in Section 2 can be expressed formally as CSPs (*Constraint Satisfaction Problem*) [8], and proofing conformance and optimality amounts to solving such a problem.

In order to facilitate the translation of a quality requirement expressed in natural language, QRL uses the linguistic patterns presented in [4] (See Figure 2.a). A linguistic pattern, or *L–pattern* for short, is a standard sentence that represents the whose set of requirements that can be expressed using that sentence. It should be as clear and synthetic as possible in order to avoid ambiguity and redundancy as much as possible. An L–pattern consists of a sentence in which there are a number of placeholders written within angles that represent values that must be substituted before obtaining an instance of the pattern. Thus, an L–pattern may also be viewed as a way to define the meaning

```
catalogue com.acme.stdMOWS {
 category Reliability {
     TTF {
         description: "Time to Fails";
         domain: increasing numeric (0, 10*365*24] hour;
         pattern: "The mean time to fails will be ⟨time⟩ at least" ≙ "TTF.mean ≥ ⟨time⟩";
     }
     TTR {
         description: "Time to recover from errors";
         domain: decreasing numeric (0, 14*24] hour;
         pattern: "The mean time to recover from a fail will be less than ⟨time⟩" ≙ "TTF.mean ≤ ⟨time⟩";
         pattern: "The maximum time to recover from a fail will be less than ⟨time⟩" ≙ "TTF.max ≤ ⟨time⟩";
     }
 }
}
```

a) Ontology

```
using com.acme.stdMOWS;

conditions for ITrailersBank {
    require {
        c1: TTF > 90 min
        c2: TTR < 60 seg
        c3: COST ≤ 0.1 Euro
        c4: STREAMING ≥ Yes
    }
    negotiation {
        r1 {
            weaken
                c4: STREAMING = No;
            strenght
                c3: COST ≤ 0.03 Euro;
        }
    }
    weights {
        TTF = -90;
        TTR = 30;
    }
}
```

b) Conditions

```
// Provided by zipi
using stdMOWS.lsi.us.es;

offer for ITrailersBank {
    provide
        c1: TTF = 120 min
        c2: TTR = 50 seg
        c3: COST = 0.02 Euro on [MON..FRI]
        c4: COST = 0.05 Euro on [SAT, SUN]
        c5: STREAMING = No
}

// Provided by velazquez
using com.acme.stdMOWS;

offer for ITrailersBank {
    provide
        c1: TTF = 360 min
        c2: TTR = 50 seg
        c3: COST = 0.4 Euro
        c4: STREAMING = Yes
        c5: MEDIA = BroadBand
    negotiation {
        r1 {
            weaken { c5: MEDIA ∈ {ISDN, T1}; }
            strenght { c3: COST = 0.3 Euro; }
        }
    }
}
```

c) Offers

Figure 2. A specification in QRL of the requirements in Figure 1

of a quality requirement using natural language, but with precise semantics due to its corresponding constraints. This way, if we want to test if a given document is consistent, we only have to check if the set of constraints into which it is translated is consistent or not.

The dual nature of requirements as sentences in natural language and in formal language is simple, but very expressive. So far, we have not found a problem that cannot be expressed using constraints on quality attributes. However, its application in a real–world set has an important problem: the vocabulary. What happens if the customer uses a number of quality requirements the provider cannot understand? Or even worst, what happens if both use the same identifiers but the implied semantics are different? In general, quality documents must have a pointer to the ontology over which they are defined, and an offer can be checked against a number of conditions as long as they share the same ontology or there is a mapping between them. The details concerning such mappings fall beyond the scope of this paper, but there are a number of tools such as BIZTALK that can be used to map an ontology onto another (`www.biztalk.net`).

Figure 2 shows a formal description in QRL of the conditions and offers of the example we presented in Figure 1. We use the same ontology in both documents.

3.2. Negotiation clauses

In QRL, we can define negotiation clauses in both the conditions and the offers, and they are considered valid as long as it weakens a number of clauses and strengths others or introduces new ones.

It is important to emphasise that negotiation is bilateral because if a customer and a provider do not agree on the main clauses of a contract, they both can express additional constraints. In the example, the negotiation clause in the conditions express that the customer can accept a provider that does not have streaming capabilities as long as the connection cost is less or equal than 0.03 Euro. `zipi` does not include any negotiation clauses, but `velazquez` can offer an ISDN or T1 connection as long as the customer can afford to pay 0.3 Euro per connection.

3.3. Optimisation criteria

In QRL, optimisation criteria may be expressed by assigning weights to the set of attributes under consideration. If the attribute vector is represented as $\vec{A}$ and the weight vector as $\vec{W}$, the service we select is a service for which an agreement may be reached that maximises the function $\vec{W}^T \times \vec{A}$. Thus, if an attribute has a negative weight, it is minimised, if its weight is zero, it does not matter, and if its weight is positive, then it is maximised.

In the example in Figure 2 the weights indicate that attribute TTF is more important to the customer than attribute TTR.

3.4. Temporal requirements

Sometimes, the quality level a web–service may require or offer depends completely on the hour, the day or the month. Thus, quality requirements may be temporal. In our example, provider `zipi` states that clause `c3` is valid on workdays, and clause `c4` at weekends.

Such constraints may be expressed in QRL by means of temporal constraints on hours, days of the week or months. Below you can find several examples of such constraints:

- $DELAY < 2$ at $[8..14:30, 16..18:30]$. This constraint indicates that the response time must be smaller than two seconds every day during the specified time slices.
- $DELAY < 2$ on $[MON..SAT]$ of $[JAN..JUL, SEP..DEC]$. This constraint indicates that the response time must be smaller than two seconds from Monday to Saturday at any time, except for August.

4. Run–time support

Figure 3 shows the basic architecture of MEEM (*Management and Execution Environment for* MOWS), an experimental development framework that offers support for the administration and execution of MOWS. The details concerning monitorisation and authentication have been intentionally omitted because they fall beyond the scope of this paper.

MEEM is composed of three main components: an interface repository, an implementation repository, and a quality trader. Its similarity to standard CORBA elements is not by chance because its basic functionality is close to the one defined in CORBA, being the main difference that these components are QA (*Quality Aware*).

Managing a MOWS consists of two tasks, namely: registering interfaces, and registering web services. Registering an interface consists of registering its syntactic signature and the set of conditions that may be necessary in a given organisation. Note that several conditions documents may be attached to the same interface according to the different sets in which a web service implementing it may be used. Registering an implementation consists of registering both information about how to have access to it and also its associated quality document.

The needed run–time support is necessary only when an application asks for a service implementing an interface with a given quality level. In such cases, the quality trader

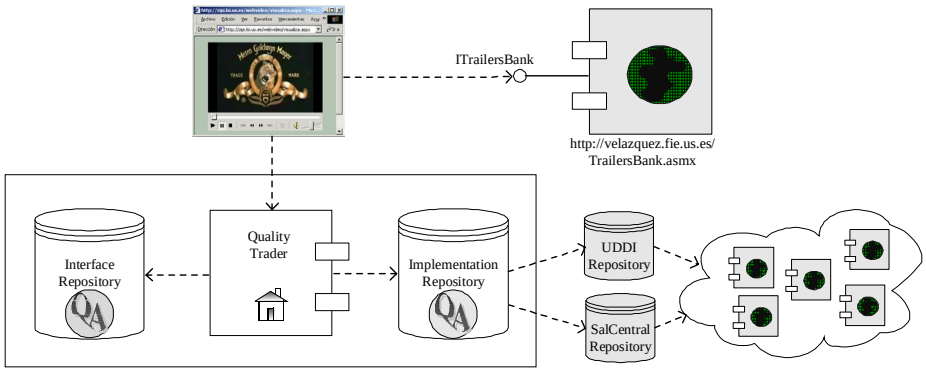

Figure 3. Basic architecture of our MOWS run–time system.

is responsible for searching for the web service that both conforms to the conditions and is optimal over the set of registered services in the implementation repository.

In order to optimise the effectiveness of the quality trader, it has access to a database that associates each pair (interface, conditions) to an implementation, if it exists. This database acts as a cache so that the first time the quality trader has to find an implementation for a given interface at a given quality level, it has to search the whole implementation repository for services satisfying that quality level. If it finds such a service, then a record is stored in the cache so that the next time a customer asks for such a service it may be found quickly and efficiently.

5. Related work

In this section, we compare our proposal with other authors' work. Comparisons are brief and hardly justice to the creativity and hard work of their authors. Our purpose is to give the reader a view of the spectrum of possibilities that are available, their strengths and weaknesses.

There are a number of notations and languages for specifying quality requirements. NoFun [5] and QML [6] fit into this category. Unfortunately, none of them has support for automatic, bilateral negotiation, optimum search or temporal constraints; furthermore QML does not have linguistic support to import ontologies, which hinders its effectiveness in open and heterogeneous systems.

There are a number of authors who focus on describing quality requirements by means of standard sentences or linguistic patterns in natural language. The pattern–based language presented in [4] fits into this category. Unfortunately, this language does not allow for a precise definition of quality requirements, thus leaving room for inaccuracies or vague remarks. QRL uses the same notation as in [4] for specifying linguistic patterns, but it also allows the requirements engineer to annotate each linguistic pattern with its corresponding mathematical constraint in order to state their semantics precisely.

Regarding negotiation, we are not aware of any proposal allowing for so many possibilities as ours. We think that this is due to the fact that other proposals focus on QoS requirements, which are mainly defined on attributes whose value depend on the workload the provider of a service or the network must support. In this context, the most complete automatic negotiation model of which we are aware is the one proposed in [7]. In this model, only providers can make counter-offers when their offers do not satisfy a customer's request or when the offer selected by a customer is no longer valid.

As for optimality, Koistinen [7] proposed the use of some functions for calculating the value of a quality attribute regarding the preferences of the customer. This proposal has several drawbacks, as shown in [2], where the authors present a model based on hierarchies of contracts to specify clients preferences. However, it cannot dealt with providers that can also negotiate its quality level.

Finally, there are also a number of proposals related to the supporting tools. The main advantage of our proposal is that quality requirements expressed in natural language can be assigned precise semantics because they are translated into formal constraints that can be checked automatically for consistency or conformance with respect to another set of constraints. Furthermore, there a number of commercial tools such as ILOG (`www.ilog.com`) that can solve CSPs automatically, so our proposal is highly automatisable.

6. Conclusions and future work

The automatic checking of quality requirements will play a fundamental role in the future market of web services, because it will allow to build quality–aware systems that are optimal from an economic point of view and whose quality level can be guaranteed. In this paper, we have iden-

tified some of the main problems with which MOWS are faced, and we have also presented a proposal that can be used to solve them in practice.

We think that the formal and informal view of quality requirements overcomes some of the problems of using formal methods: i) constraints are an usual tool for software engineers, so it is not difficult to work with them; ii) it is possible to define the semantics associated with a sentence in natural language; iii) it is possible to reason using mathematical constraints; iv) there are a number of commercial libraries with which CPS over reals, integers or enumerations can be solved efficiently, so QRL can be implemented.

As for the expansiveness of our proposal, its main advantages with respects to other are the following: i) it allows for bilateral negotiation rules; ii) it allows for optimisation criteria; iii) it allows for temporal constraints.

Currently, we are trying to apply our proposal to the evaluation of design alternatives. The idea is to use conditions as a means to express the requirements a design must satisfy, and offer documents as a means to express what requirements a design alternative fulfills. Thus, selecting the best alternative amounts to an optimisation problem.

In the future, we are planning on developing an agent–based solution for filling the implementation repository so that web services are not searched for on demand but beforehand, every time a new interface is registered. This way the cache with which the quality trader works can be updated incrementally, and the connection time might be reduced considerably.

References

[1] ASP Industry Consortium. An Overview of The Guide to Service Level Agreements. White Paper, 2001. Disponible en `http://www.aspindustry.com`.

[2] C. Becker, K. Geihs, and J. Gramberg. Representing quality of service preferences by hierarchies of contracts. In *In Proceedings of Elektronische Dienstleistungswirtschaft und Financial Engineering (FAN'99)*, Augsburg/Germany, 1999.

[3] R. Corchuelo, A. Ruiz, J. Mühlbacher, and J. García-Consuegra. Object-Oriented Business Solutions. In *Chapter 18 of ECOOP'2001 Workshop Reader, LNCS n^o (to appear)*. Springer-Verlag, 2001.

[4] A. Durán, B. Bernárdez, M. Toro, R. Corchuelo, A. Ruiz, and J. Pérez. Expressing Customer Requirements Using Natural Language Requirements Templates and Patterns. In *Proc. of the IIIrd Conference on Circuits, Systems, Communications and Computers CSCC'99*, Athens, (Greece), 1999. IMACS/IEEE.

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

[6] S. Frolund and J. Koistinen. Quality–of–Service Specification in Distributed Object Systems. *Distributed Systems Engineering Journal*, 5(4), 1998. Disponible como informe técnico HPL–98–159 en `http://www.hpl.hp.com/techreports`.

[7] 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, San Diego, USA, 1998.

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