Reconsidering Quality of Service Analysis in Dynamic and Open Systems

Franck Chauvel
SINTEF ICT - MOD Group
Oslo, Norway
franck.chauvel@sintef.no

Sébastien Mosser
SINTEF ICT - MOD Group
Oslo, Norway
sebastien.mosser@sintef.no

Arnor Solberg
SINTEF ICT - MOD Group
Oslo, Norway
arnor.solberg@sintef.no

Abstract

While recent advances in middleware and Software Engineering are at the edge of making Software Reuse a reality, the newly given dynamicity and openness seriously challenge validation, especially regarding *Quality of Service* (QoS). Modern QoS models require highly accurate quantitative descriptions of future execution environments, whereas these environments are getting more and more uncertain as dynamicity and openness increases. As a result, our ability to validate QoS concerns diminishes until a point where its worthiness must be reconsidered. In this context, we envision in this paper an alternative approach to reconcile dynamicity, openness and QoS. It combines run-time and design-time validation together with advanced modelling of uncertainty. We describe the overall process and discuss several obstacles to its realization, including QoS requirements modelling, QoS validation and run-time QoS fault-detection.

1 Introduction

With the emergence of middleware platforms supporting run-time adaptations (*e.g.*, WS, OSGi, JEE), software systems are becoming more and more open and dynamic. These new abilities, which improve software reuse, also directly challenge our ability to validate these systems. This is further complicated when extra-functional requirements like *Quality of Service* (QoS) must be met. Overlooking QoS undoubtedly leads to overpriced rollbacks in the development, or even jeopardise the delivery of final software artifacts. To mitigate QoS issues, modern approaches [1] validate *end-to-end* QoS requirements against predictive QoS models representing the system and its environment. Precise quantitative descriptions (*e.g.*, probability distributions, memory capacities, CPU speeds) are critical to ensure high accuracy of QoS predictions. However, the more dynamic a system is, the more difficult it becomes to precisely characterise its environment, and, in turn, the less significant existing techniques are. The same is true for end-to-end requirements, whose numerical estimation becomes more and more irrelevant as the system bounds become uncertain. This inherent incompatibility between dynamicity and accuracy challenges the worthiness of QoS analysis: the effort spent at design-time validating QoS requirements must be put in perspective using the significance of the validation results at run-time.

We are convinced that our ability to tame dynamicity and openness is bounded to the way we address the uncertainty it implies. We believe that postponing validation until run-time to minimise uncertainty while using proper mathematical foundations shall help reconcile dynamicity and QoS engineering. In this setting, the contribution of this vision paper is to examine to which extent *recent models at run-time* techniques [2] can be combined with advanced modelling/reasoning techniques addressing uncertainty in the first place [3]. To this end, we first present our approach as a development process focusing on QoS requirements. We then reviewed the key properties that an *ideal* QoS model must exhibit to enable the validation of QoS requirements under openness and dynamicity. We then discuss the construction, the validation and the injection of such QoS requirements into running systems.

2 Motivations

Let us consider an application server, as a practical example of above-mentioned QoS issues. In a general sense, a server processes requests for resources (*e.g.*, files, HTML pages, service invocations) and users are often expecting its response time to be bounded, (we consider here the server-side response time, *i.e.*, the time between the reception of a request and the emission of the related response). A typical QoS requirement describing the server is "*the server response-time must be less than 3 s for 90 % of requests in a single day*".

Various models and techniques can be used to validate that this requirement will hold at run-time, such as Markovian models for instance. In a general setting, servers' response time depends on two main factors, namely, the arrival rate (*i.e.*, the speed at which requests arrive, in $req.s^{-1}$), and the service rate (*i.e.*, the speed at which the server processes them, in $req.s^{-1}$). The task of QoS expert is therefore to anticipate, at design time, the environments the system may be facing. In Markovian models, this is achieved by means of probability distributions quantifying the likelihood of possible arrival rates, and service rates. For the sake of the example, let us assume the two following distributions: f, which describes the arrival rate, is a normal distribution centred around 50 $req.s^{-1}$ with a deviation of 25 $req.s^{-1}$, whereas g, which describes the service rate, is also a normal distribution centred around 100 $req.s^{-1}$ with a deviation of 25 $req.s^{-1}$. The Markovian model assesses that our requirement holds. However, in real life, when the actual arrival rates will exceed the service rate, our requirement will actually not hold anymore, as the server will either start to store or to reject pending request, depending of the overload management policy.

So what did we get wrong? Actually, the original QoS requirement and the one that we validated, are not the same. The one we validated is: "*assuming the distributions f and g for the arrival and service rates, respectively, the response time must be less than 3 s for 90 % of the requests in a single day*" and is much more restrictive.We could obviously relax our expectations on the expected execution environment by using uniform distributions, but we would not be able to validate our requirement: there will always be a non null probability that the arrival rate exceeds the service rate, in which case the response time will tends towards infinity.

The previous example shows the intrinsic difficulty of QoS requirement modelling and validation, even on simple examples. Existing QoS requirements reflect our expectations from a user perspective, whereas our validation techniques reflects the system's ability from the "pragmatic" engineer standing point. Open and dynamic systems acerbate this conflict and a paradigm shift is needed to validate QoS in such a context.

3 Overview of the Approach

The paradigm shift we envision is the *continuous anticipation of the inherent conflict between the user expectations and the system abilities*, so as to trigger preventive adaptations. The proposed approach combines advanced mathematical modelling of uncertainty with recent advances in models@runtime. Fig. 1 illustrates the development process making concrete continuous QoS validation in open and dynamic systems. This process reflects our will to turn the traditional once-and-for-all QoS validation into a continuous validation process at runtime. It is structured as follows:

- **QoS Requirements Elicitation** is jointly carried out by the user and the QoS expert. The objective here is to obtain detailed QoS requirements that can be later turned to operational artifacts, stating the expectations from a user point of view.

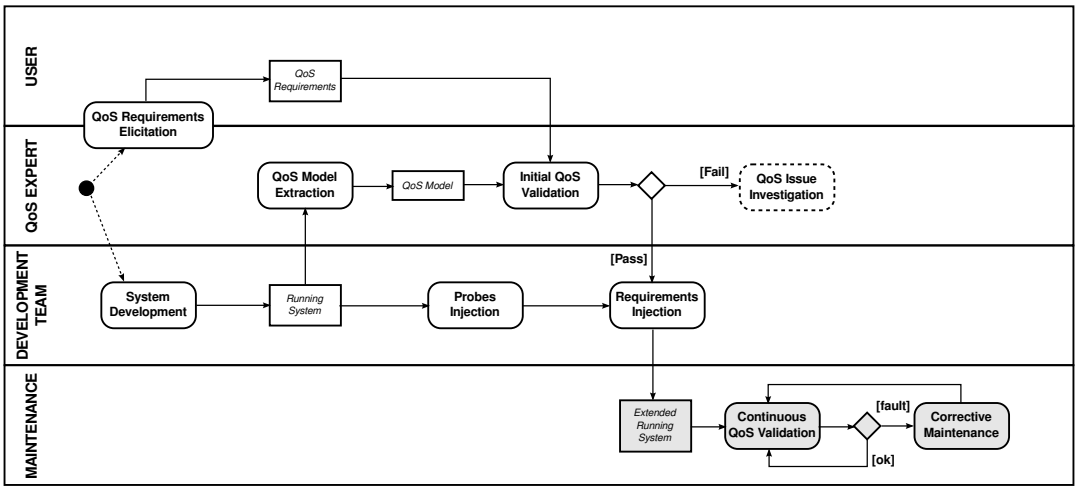

Figure 1: Overview of the process, designed to tackle QoS validation for open/dynamic systems

- **QoS Model Extraction** is carried out by the QoS expert and aims at obtaining a model describing the dynamics of the system's QoS, describing the system's abilities. Recent advances in QoS models shows that automated extraction is needed to cope with the difficulty of quantifying the various parameters needed to obtain accurate QoS predictions [1, 4].
- **Initial QoS validation** is also carried out by the QoS expert, but is an initial comparison between the system abilities and the user expectations. When this first validation fails, the QoS expert has to determine whether they are due to ill-formed requirements, or to error in the running system. Once issues have been mitigated, the process restarts with QoS model extractions or with alternative requirements.
- **Probes and Requirements Injection** is the combination of the running system (extended with additional monitoring capabilities) and the user requirements: the mismatch between requirements and systems abilities can be dynamically checked while the system is running, taking into account unforeseen environments discovered at runtime.
- **Continuous QoS Validation**, supported by probes injection, aims at anticipating when QoS requirements might actually be violated. This is further complicated by the potential softness of QoS requirements, whose violation may deal with softness regarding amplitude, duration or frequency.
- **Corrective Maintenance** is needed to moderate QoS issues anticipated at run-time. This might be done automatically using self-adaptation techniques [5] or semi automatically [6].

3.1 Towards Assume/Guarantee-based QoS Requirements

The first obstacle to the process sketched on Fig. 1 is the lack of a model for QoS requirements. By contrast with traditional QoS requirements, we believe that key point is not to quantify

end-to-end QoS, but to constraint its possible values by capturing the "envelope" in which the QoS is expected to evolve, depending on the context. Fig. 2 illustrates what we envision as a good QoS requirement, regarding our server example.

```
whenever (service_rate > arrival_rate): // Nominal behaviour
    response_time must be always below service_rate * max_waiting_req
    rejection_rate should be around 0

whenever (service_rate <= arrival_rate): // Overload
   response_time must be always below service_rate * max_waiting_req
   rejection_rate should be around arrival_rate - service_rate
```

Figure 2: Alternative QoS Requirements bounding the response-time of a server application

In Fig. 2, we express that when the service rate exceeds the arrival rate, the response time shall be bounded by the time spent processing all the requests that are already pending, and no request shall be rejected. By contrast, if the arrival rate exceeds the service rate, new requests shall be rejected at a rate which is the difference between arrival rate and service rate. This example highlights several key properties of what we think are good QoS requirements:

- **Completeness.** Such requirements forms a *complete partition* regarding the different situations that might occur, namely nominal behaviour, or overloads in our example. Such a partition discards superfluous details but retains the critical distinctions. Building such partitions is critical to tame open and dynamic systems.
- **Uncertainty.** Such QoS requirements are inherently uncertain. First, they are imprecise as they capture *relative orders of magnitude*, instead of accurate prediction. For instance, we only assume that the difference between the arrival rate and the service rate is a good approximation of what must be the rejection rate. In addition, QoS requirements are inherently permissive or soft, as they make explicit what is acceptable in which contexts. Imprecision and Softness are the counter part of completeness, to tame open and dynamic systems.
- **Clear Intention & Operational Extension**. They have a clear intention: "bound the response time" in our example. This critical from the user perspective. Besides, assuming that the values at play can be observed at run-time, such requirements must be turned into operational checkers. This is critical from the QoS Expert perspective: Considering the effort to elicit and track one single QoS requirement through the development process, we directly question the worthiness of non-operational requirements.
- **Qualitative.** The envelope that is presented here is qualitative: it does not refer to crisp values (except for 0, which can be considered as a qualitative value). Such qualitative description, uses merely basic arithmetic operators, as non-linearity emerges from the identification of distinct cases (*e.g.*, nominal *versus* overloads).

How one can come up with such requirements? We believe that such requirements can be elicited using the two main abilities of human reasoning: *abstraction* and *decomposition*. *Decomposition* is our ability to break down complex problems into smaller ones that can be separately solved, whereas *abstraction* is our ability to get rid of irrelevant details and to retain only critical distinctions. In the previous example, abstraction selects the properties of interest: service rate, arrival rate, rejection rate, and maximum number of waiting requests. By contrast,

decomposition identifies our two cases: one characterising the nominal behaviour of the system, the other one characterising overloads.

How to manage uncertainty? Aside of the well known Probability theory, there have been several attempts during the past 50 years to provide sound mathematical foundations for reasoning under uncertainty [3]: Fuzzy Sets, Evidence Theory, Rough Sets, Grey Theory and numerous hybrid theories. Each of these theories addresses specific aspects of uncertainty, *e.g.*, eventuality, cognitive imprecision, approximation, incompleteness. We are firmly convinced that, while abstraction and decomposition will help designers elicit QoS requirements, such formal foundations will permit to effectively exploit their inherent imprecision and softness. Modern Control Theory relies for instance on Fuzzy Set Theory to tackle complex non-linear and multivariate problems.

3.2 QoS Model Extraction and Validation

QoS Models, at least performance models [1], have always suffered a very low acceptance due to the difficulty to quantify the numerous parameters that they require. However, recent advances in parameter estimation now permit to extract such parameters directly from running artifacts, typically traces or source code [4]. Alternatively, it is also possible to extract QoS models using approximations techniques combined with performance testing: The model then mimic the behaviour observed during testing.

QoS Models extraction is a great step towards a general and effective QoS engineering. In our process, it is critical to continuously validate the QoS of the system as often as needed, by changes in the context. Coming back to our server example, and assuming an existing QoS model, it becomes possible to validate that the response time is bounded, in nominal conditions as well as during overloads.

Models extraction is however still not sufficient as it results in accurate predictive models whose complexity is far too high to be embedded at runtime. There is hence a need for models approximations and efficient parameter identification to reflect context changes observed at runtime, and enable in turn continuous validation.

3.3 Probe Injection and Continuous Validation

The last obstacle to the process introduced in Fig. 1 is the ability to deploy probes that observe the various parameters involved in QoS requirements, and the ability to decide whether a given requirements will be violated or not. Measuring values at play in QoS requirements implies, in the general case, the development of specific probes. The design and the integration of such probes undoubtedly impacts the system itself, so it exhibits relevant information. While probes represent an additional development, their integration in the rest of the system can be lightened by advanced software engineering techniques, *e.g.*, aspect-oriented programming or dynamic deployment.

Deciding whether a given QoS requirement will be violated or not depends on two aspects: *(i)* our ability to calculate, at run-time, the expected envelope and trajectory of the QoS properties of interest, and *(ii)* our ability to handle the imprecision and softness inherent to QoS requirements. Regarding run-time evaluation, we envision two main approaches, the dynamic interpretation of QoS requirements at run-time, or if their semantic is not computationally efficient enough, the injection of some precomputed numeric approximations. Regarding imprecision, it is critical to properly handle all acceptable relative deviations, both in term of

amplitude and frequency. Efficient QoS fault detection is critical to ensure the good usability and stability of the system, especially is fault detection triggers self-adaptive mechanisms. In this perspective, we aim at reusing existing classification techniques (*e.g.*, clustering, pattern recognition), already effective in other engineering fields.

4 Conclusions & Road Map

In this paper, we envision a paradigm shift to support the modelling of QoS requirements in adaptive systems that are open and dynamic systems. This approach is complementary to usual QoS methods, as it aims to address systems that are typically not designed to support usual QoS models. We are especially interested in looking for convergence with the development various services architectures, including Internet of Things and Internet of Services. Regarding IoT, we are especially exploring within the ThingML project [7], which defines a model-driven framework enhancing the development of embedded systems.

On the short term, we are targeting at first the formal definition of a QoS requirements semantics that would handle uncertainty while preserving computational efficiency. Another task is the definition of QoS fault detection mechanisms that would report gradual degradations, supporting deviations in frequency or amplitude and enable preventive adaptations. On the mid-term perspective, our focus will be on the development of QoS requirements framework, that would combine traditional verification with the imprecision and softness of QoS requirements. Finally, in the long term perspective, we will focus on the evaluation of the feasibility, of the effectiveness and of the performance in securing QoS in dynamic and open systems.

Acknowledgements. This work is funded by the SINTEF strategic projects SiSaS and MODERATES.

References

[1] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *IEEE Transactions on Software Engineering*, vol. 30, pp. 295–310, may 2004.

[2] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg, "Models@ Run.time to Support Dynamic Adaptation," *IEEE Computer*, vol. 42, no. 10, pp. 44–51, 2009.

[3] H. Bandemer, *Mathematics of Uncertainty*. Springer, 2006.

[4] K. Krogmann, M. Kuperberg, and R. Reussner, "Using Genetic Search for Reverse Engineering of Parametric Behavior Models for Performance Prediction," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 865–877, 2010.

[5] F. Chauvel, H. Song, X. Chen, G. Huang, and H. Mei, "Using QoS-Contracts to Drive Architecture-Centric Self-adaptation," in *Proceedings of the 6th International Conference on the Quality of Software Architectures (QoSA 2010)*, vol. 6093 of *Lecture Notes in Computer Science*, pp. 102–118, Springer, 2010.

[6] B. Morin, O. Barais, G. Nain, and J.-M. Jézéquel, "Taming Dynamically Adaptive Systems using Models and Aspects," in *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009)*, pp. 122–132, IEEE, 2009.

[7] F. Fleurey, B. Morin, A. Solberg, and O. Barais, "MDE to Manage Communications with and between Resource-Constrained Systems," in *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems (MODELS 2011)*, vol. 6981 of *Lecture Notes in Computer Science*, pp. 349–363, 2011.