Bringing QoS Specifications to the Architectural Level

Alexandre Sztajnberg
IME/UERJ and GTA/COPPE/UFRJ, RJ, Brazil

Orlando Loques
Instituto de Computação, UFF, Niterói, RJ, Brazil

Abstract. Designing QoS aware applications is not a trivial task. It would be interesting to have QoS aspects of an application designed separate from the functional aspects. It would be also beneficial to have a mapping from the QoS requirement specification to a possible implementation that could be systematized. In this position paper, we focus on how QoS aspects of an application can be specified in a software architecture level, within the R-RIO approach. Bringing QoS concepts to the architecture level can help designers to reason about non-strictly-functional problems before implementing the system. This approach also separates application's functional concerns from the non-strictly-functional ones, allowing QoS description reuse and dynamic evolution.

1. R-RIO Approach

The R-RIO, Reflective-Reconfigurable Interconnectable Objects [Loq 99], approach is based on: (i) a component-based design methodology that relies on composition of functional modules and non-strictly-functional connectors to build application architectures; (ii) an architecture description language (ADL), called CBabel, with which are described the component selections, interconnections and composite elements that will be used to structure the applications, and non-strictly-functional aspects such as coordination, distribution, fault tolerance or QoS that will allow customization of those applications; and (iii) a supporting reflective middleware that allows managing the configuration of application architectures.

2. QoS contracts in CBabel

In the R-RIO approach, the concept of contracts is used to specify, in the architecture level, module behavior details and functional restrictions, and coordination and interaction agreements between the modules.

We are currently extending CBabel to turn it able to describe QoS contracts. We borrowed a subset of the QML language [Fro 98] and adapted its structure and terminology to an architecture description scope. QoS contracts are described separately from the components and a *Qos-profile* clause relates them. A simple QoS contract can be described in a *QoS-connector* as follows:

```
connector QoSDistrib_Coord{
  QOScontracts {
    (Producer, Buffer) renegotiate;
    (Consumer, Buffer) terminate;
} }
```

Producer, *Consumer* and *Buffer* modules should have their *QoSProfiles* already described when binding into a contract. There can also be a violation policy handling associated with a contract (e.g., *renegotiate* and *terminate*).

Once an application's QoS aware architecture is described with CBabel it is possible to submit it to a coherence and consistency checking. For instance, a client should not require a QoS that a server cannot provide. This allows the designer to reason about the QoS prior to any other step towards implementation.

3. Mapping

Given a CBabel architecture description containing QoS contracts, a standard recipe maps QoS descriptions into actions and data that are stored as meta-level information. R-RIO middleware interprets this information to configure the adequate resources and initiate appropriate procedures to enforce the QoS. To this end, R-RIO uses two special modules. **QoS Experts** can interpret QoS parameters stored in the meta-level and synthesize a *QoS*-connector that will enforce QoS. The Expert will be invoked at binding time. **QoS Monitors** are special modules that will continuously measure the actual QoS parameters and check if they are in the required range. According to the violation handling policy described in the contract, the QoS Monitor can ask the QoS Expert to reconfigure the QoS-connector, if a violation occurs. Each QoS category (e.g, fault-tolerance) is supported by an Expert and each QoS contract is supported by a Monitor. There is no need for a particular Expert to handle just one given QoS contract, as it will get in action few times compared to a Monitor.

4. Support

As QoS contracts in CBabel are not restricted to multimedia, a diverse infrastructure is necessary to enforce every QoS category. For instance, the resources required to assure reliability are different from those required to assure performance. The main advantage of the R-RIO's approach to QoS is the mapping mechanism that can be used to handle any QoS category. It is up to the Experts and Monitors to actually enforce QoS. New QoS categories can be handled, as new Experts and Monitors are available. R-RIO will not provide QoS enforcement mechanisms, but will encapsulate existing ones.

We argue that to enable separation of concerns and dynamic evolution, not the functional modules, but meta-level components and the middleware, should be QoS aware. For example, in face of new demands during running time, a client module could require that a given service now should be fault-tolerant. If QoS were not separate from the base functionality, the involved modules would have to encapsulate fault-tolerance enforcement code even if it would not be used immediately. Our technique uses meta-level component selection (connectors) and composition to map QoS described with CBabel into a deployable implementation.

Currently we are investigating how 2K's QoS infrastructure [Nah 99] could be integrated with R-RIO as the target support to synthesize R-RIO's QoS-connectors.

Acknowledgments: Alexandre Sztajnberg is supported in part by a grant from CAPES-Brazil, BEX 1014/99-4.

References

[Fro 98] Frolund, S. and Koistinen, J., "Quality-of-service specifications in distributed object systems", Distributed Systems Engineering, IEE, No. 5, pp. 179-202, UK, 1998.

[Loq 99] Loques, O. *et al,* "Integrating Meta-Level Programming and Configuration Programming", OORaSE'99, Denver, USA, November, 1999.

[Nah 99] Nahrstedt, K., Li, B., "A Control-based Middleware Framework for QoS Adaptations", IEEE Selec. Areas in Comm., Special Issue on Service Enabling Platforms, 1999.