An Approach to Design with Differentiated Services by Contract

Romulo Cerqueira, Sidney Ansaloni, Orlando Loques
IC/UFF
Niterói, RJ, Brazil
{curty, ansaloni, loques}@ic.uff.br

Alexandre Sztajnberg
DICC/IME/UERJ
Rio de Janeiro, RJ, Brazil
alexszt@ime.uerj.br

Abstract

This poster presents an approach to describe and deploy system contracts with service guaranties having dynamic functional and non-functional requirements, which include generic types of QoS. The approach defines the relationships between contracts, differentiated quality of services, and the resource management process. We also propose a multi level view to map these relationships as first class architectural entities.

1. Introduction

In our approach application's differentiated quality of service requirements are described by CBabel ADL contract extensions and mapped to the R-RIO framework and user application components. The infrastructure required to manage these contracts follows a standard architectural pattern, which can be directly mapped to specific components included in R-RIO's supporting middleware. This allows a designer to write a contract and follow a standard recipe to insert in those components the extra code required to its enforcement.

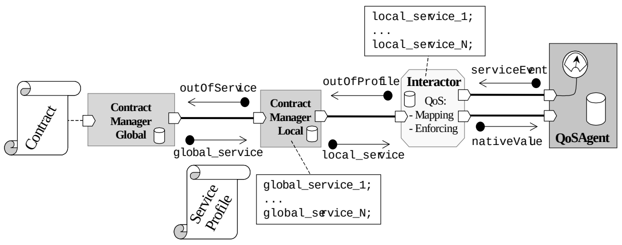

Figure 1. Components used to enforce QoS

2. The Extended R-RIO Framework

The original R-RIO framework provided support to architecture level contracts in an ad-hoc fashion. Currently, our infrastructure can handle generic contracts with dynamic component-based service adaptations. This generality was achieved through a pattern (fig. 1) based on three meta-level components: **Contract Manager** that interprets contract descriptions to extract service negotiation information and map each service to a set of global services. When every service inside the negotiation clause has been unsuccessfully tried, an *out-of-service* state is reached, the service is stopped, and a contract violation message is returned to the user; **Interactor** translates pre-defined services to system support level services, requests those services to QoS Agents, and can receive *out-of-range* notifications from these QoS Agents. When this occurs, the Interactor can try to readapt the resource or notify the Contract Manager to initiate another negotiation; **QoS Agents** encapsulate the access to system level mechanisms to actually make resource allocations and to monitor required property values. Fig. 2 presents a CBabel contract describing non-functional requirements for a "DeskTop TV" service.

```
contract {
  service {
    start audioTVServer with
      Processing.codec = g723; } audio_qos_1;
  service {
    start audioTVServer with
      Processing.codec = pcm;  } audio_qos_2;
  service {
    start videoTVServer with
      Processing.codec = h263; } video_qos_1;
  service {
    start videoTVServer with
      Processing.codec = h261; } video_qos_2;
  negotiation {
    video_qos_1 -> video_qos_2;
    video_qos_2 -> out_of_service;
    audio_qos_1 -> audio_qos_2;
    audio_qos_2 -> out_of_service;
} } deskTopTV;
```

Figure 2. DeskTopTV QoS contract

3. Conclusion

Our approach allows non-functional requirements to be specified using high-level contracts associated to an ADL. The mechanisms required to interpret and enforce the contracts follow an architectural pattern that can be implemented by a standard set of components. We have evaluated the approach through use cases. This showed that, depending on the particular contract, specific parts of the code of the supporting components may have to be adapted. We believe that our approach can help to identify and make the required adaptations. In addition, once defined, a particular contract and the supporting components can be reused in different applications.

4. References

[1] Cerqueira, R., et al., "Deploying Non-Functional Aspects by Contract", Middleware 2003 - 2nd Workshop on Reflective and Adaptive Middleware, Rio de Janeiro, Brazil, June, 2003.