Referências

AHO, A. V.; ULLMAN, J. D. **Compiladores**: Princípios, Técnicas e Ferramentas. Guanabara – Koogan, 1995.

ALLEN, R. J. **A Formal Approach to Software Architecture**. 1997. 248p. Tese (Doutorado em Filosofia) – School of Computer Science, Carnegie Mellon University, 1997. Disponível em http://reports-archive.adm.cs.cmu.edu/anon/1997/CMU-CS-97-144.pdf. Acesso em: 24 jan. 2003.

AREVALO, G. B. **Architectural Description of Object Oriented Frameworks**. 2000. 138p. Dissertação (Mestrado em Ciência da Computação) – Ecole des Mines de Nantes, 2000. Disponível em http://www.iam.unibe.ch/~arevalo/publications/Master_Thesis_doc.doc. Acesso em 31 jan. 2003.

AURRECOECHEA, C.; CAMPBELL, A. T.; HAUW, L. A. A Survey of QoS Architectures. **Multimedia Systems**, Berlin, v. 6, n.3, p. 138-151, Maio de 1998.

BOWEN, J. P.; HINCHEY, M. G. Seven More Myths of Formal Methods. **IEEE Software**, v. 12, n. 4, p. 34-41, Julho de 1995.

BRADEN, R.; BERSON, S.; HERZOG, S.; JAMIN, S.; ZHANG, L. **Resource ReSerVation Protocol (RSVP): Version 1 Functional Specification**. IETF Request for Comments (RFC) 2205. Setembro de 1997.

CLEMENTS, P. **A Survey of Architecture Description Languages**. In: International Workshop on Software Specification and Design, 8., 1996, Paderborn, Alemanha, 1996. Disponível em ftp://ftp.sei.cmu.edu/pub/sati/Papers_and_Abstracts/Survey_of_ADLs.ps. Acesso em 29 dez. 2001.

COLCHER, S. **Um Meta Modelo para Aplicações e Serviços de Comunicação Adaptáveis e com Qualidade de Serviço**. 1999. 125p. Tese (Doutorado em Ciências em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, 1999.

DEURSEN, A.; KLINT, P.; VISSER, J. Domain-Specific Languages: An Annotated Bibliography. **ACM SIGPLAN Notices**, v. 35, n. 6, p. 26-36, Junho de 2000.

DURAN-LIMON, H. A.; BLAIR, G. S. **Specifying Real-time Behavior in Distributed Software Architectures**. In: Proceedings of Australasian Workshop on Software and Systems Architectures (AWSA), 3., 2000, Sydney, Australia, Novembro de 2000.

FROLUND, S.; KOISTINEN, J. **Quality of Service Specification on Distributed Object Systems Design**. In: Proceedings of USENIX Conference on Object-Oriented Technologies and Systems (COOTS), 4., 1998, Santa Fe, Novo

México, Abril de 1998. Disponível em http://www.usenix.org/publications/library/proceedings/coots98/full_papers/ frolund/frolund.pdf. Acesso em: 13 dez. 2001.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns**: Elements of Reusable Object-Oriented Software. Addison Wesley, 1995.

GARLAN, D.; KOMPANEK, A.; PINTO, P. **Reconciling the Needs of Architectural Description with Object-Modeling Notations**. In: International Conference on the Unified Modeling Language (UML2000), 3., 2000, York, Reino Unido, Outubro de 2000. Disponível em http://www-2.cs.cmu.edu/afs/cs/project/able/ftp/uml00/uml00.pdf. Acesso em 31 jan. 2003.

GARLAN, D.; SHAW, M. An Introduction to Software Architecture. In: AMBRIOLA V.; TORTORA, G (Ed.). **Advances in Software Engineering and Knowledge Engineering**. Series on Software Engineering and Knowledge Engineering, V. 2. Singapore: World Scientific Publishing Company, 1993. p. 1-39.

GARLAN, D.; SHAW, M. **Software Architecture**: Perspectives on an Emerging Discipline. New Jersey: Prentice-Hall, 1996.

GOMES, A.T.A. **Um Framework Para Provisão de QoS em Ambientes Genéricos de Processamento e Comunicação**. 1999. 162p. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, 1999. Disponível em ftp://ftp.telemidia.puc-rio.br/pub/docs/theses/1999_05_gomes.pdf. Acesso em: 24 jan. 2003.

GOMES, A.T.A.; COLCHER, S.; SOARES, L.F.G. **Modeling QoS Provision on Adaptable Communication Environments**. In: IEEE International Conference on Communications (ICC 2001), 2001, Helsinki, Finlândia, Junho de 2001.

GRUNE, D.; BAL, H. E.; JACOBS, C. J. H.; LANGENDOEN, K. G. **Modern Compiler Design**. John Wiley, 2000.

HALL, A. Seven Myths of Formal Methods. **IEEE Software**, v. 7, n. 5, p. 11-19, Setembro de 1990.

HILLIARD, R. **Using the UML for Architectural Description**. In: International Conference on the Unified Modeling Language, 2., Outubro de 1999, Fort Collins, Colorado, Estados Unidos, 1999. Disponível em http://members.bellatlantic.net/~rfh2/writings/hilliard99a-using-uml-for-AD.pdf. Acesso em 7 mar. 2003.

HOFMANN, C. et al. Approaches to Software Architecture. In: BROY, M.; DENERT, E.; RENZEL, K.; SCHMIDT, M. (Ed.). **Software Architectures and Design Patterns in Business Applications**. Munique: Universidade Técnica de Munique, 1997. p. 3-46. Relatório Técnico TUM-I9746.

HOFMEISTER, C.; NORD, R.L.; SONI, D. **Describing Software Architecture with UML**. In: Proceedings of the Working IFIP Conference on Software Architecture, 1ª, 1999, San Antonio, Texas, Estados Unidos, Fevereiro de 1999.

IEEE Architecture Working Group. **IEEE P1471/D5.2**: Recommended Practice for Architectural Description. http://www.pithecanthropus.com/~awg/public_html/.

LOBOSCO, M. **R-RIO**: Um Ambiente para Suporte à Construção e à Evolução de Aplicações. 1999. 118p. Dissertação (Mestrado em Ciência da Computação) –

Universidade Federal Fluminense, Março de 1999. Disponível em http://www.caa.uff.br/~lobosco/dissertacao.ps.zip. Acesso em 31 jan. 2003.

LUCKHAM, D. C.; VERA J. An Event-Based Architecture Definition Language. **IEEE Transactions on Software Engineering**, v. 21, n. 9, p. 717-734, Setembro de 1995.

MEDVIDOVIC, N. et al. Modeling Software Architectures in the Unified Modeling Language. **ACM Transactions on Software Engineering and Methodology (TOSEM)**, Nova Iorque, v. 11, n.1, p. 2-57, Janeiro de 2002.

MEDVIDOVIC, N.; TAYLOR, N. T. **A classification and comparison framework for software architecture description languages**. Universidade da Califórnia, Irvine, 1996, 53p. Disponível em http://www.public.asu.edu/~gbpan/patternresearch/papers/A-Classification-and-Comparison-Framework-for-Software-Architecture-Description-Languages-TR96.pdf. Acesso em 29 dez. 2001

MORENO, M. F. **Um Framework para Provisão de QoS em Sistemas Operacionais**. 2002. 117p. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, 2002. Disponível em ftp://ftp.telemidia.puc-rio.br/pub/docs/theses/2002_08_moreno.pdf. Acesso em 31 jan. 2003.

MOTA, O. T. J. D. D. L. **Uma Arquitetura Adaptável para Provisão de QoS na Internet**. 2001. 113p. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, 2001. Disponível em ftp://ftp.telemidia.puc-rio.br/pub/docs/theses/2001_05_mota.zip. Acesso em 31 jan. 2003.

MUCHALUAT-SAADE, D. C. **Relações em Linguagens de Autoria Hipermídia: Aumentando Reuso e Expressividade**. 2003. 206p. Tese (Doutorado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, 2002. Disponível em ftp://ftp.telemidia.puc-rio.br/pub/docs/theses/2003_03_muchaluat.pdf. Acesso em 30 jul. 2003.

PERRY, D. E.; WOLF A.L. Foundations for the Study of Software Architecture. **ACM SIGSOFT Software Engineering Notes**, Nova Iorque, v. 17, n. 4, p. 40-52, Outubro de 1992.

PREE, W. **Framework Patterns**. SIGS Management Briefings: SIGS Books & Multimedia, 1996, 104p.

RANGEL, J. L. Departamento de Informática – PUC-Rio. **Material didático da disciplina de Compiladores**. Rio de Janeiro, Abril de 2000.

RATIONAL SOFTWARE CORPORATION. **Unified Modeling Language: Notation Guide**. Lexington, EUA, 1997, 148p. Disponível em http://iamwww.unibe.ch/~scg/Resources/UML/Mirror/PDF/notation11.pdf. Acesso em 24 jan. 2003.

RODRIGUES, R. F. **Formatação e Controle de Apresentações Hipermídia com Mecanismos de Adaptação Temporal**. 2003. 170p. Tese (Doutorado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, 2003.

SAIEDIAN, H. Towards More Formalism in Software Engineering Education. **ACM Special Interest Group on Computer Science Education (SIGCSE) Bulletin**, v. 25, n. 1, p. 193-197, Março de 1993.

SHAW, M.; DELINE, R.; KLEIN, D. V.; ROSS, T. L.; YOUNG, D. M.; ZELESNIK, G. Abstractions for software architectures and tools to support them. **IEEE Transactions on Software Engineering**, v. 21, n. 4, p. 314-335, Abril de 1995.

SZTAJNBERG, A.; LOQUES, O. **Bringing QoS to the Architectural Level**. Universidade Federal Fluminense, 2000, 2p. Disponível em http://www.ic.uff.br/~rrio/papers/Szt-QoSDOS2k.ps.gz. Acesso em 31 jan. 2003.

TAYLOR, R. N.; TRACZ, W.; COGLIANESE, L. Software development using domain-specific software architectures. **ACM SIGSOFT Software Engineering Notes**, Nova Iorque, v. 20, n. 5, p. 27–37, 1995.

Apêndice A
Arquitetura de Software e Descrição Formal de Sistemas

Este apêndice inicia introduzindo, na Seção A1, o conceito de arquitetura de sistemas, chamando a atenção para a inexistência de um consenso para uma definição mais precisa. A necessidade de uma descrição formal de sistemas e os vários mitos relacionados aos métodos formais são assuntos da Seção A2.

A1
Arquitetura de Software

Com o aumento da complexidade dos sistemas de software e a importância cada vez maior de aplicações distribuídas, o projeto de sistemas sofreu mudanças significativas na última década. Tornou-se necessária a existência de diferentes modelos em diferentes níveis de abstração e em diferentes fases de desenvolvimento (Hofmann et al., 1997).

A área de arquitetura de software é focalizada na identificação de importantes propriedades e relacionamentos, ou seja, restrições nos tipos de componentes que são necessários para a arquitetura, projeto e implementação de um sistema.

A existência de uma descrição de alto nível do sistema traz vantagens em todas as fases de seu ciclo de vida. Em especial, um maior reuso em componentes será experimentado quanto menos forem as restrições aplicadas a ele, por isso há um maior reuso no nível arquitetural em comparação com o de implementação, já que o nível de abstração é maior quando se descreve o comportamento e relacionamento abstrato de componentes do que quando se define seu código em uma linguagem de programação.

Como forma de comparação, (Perry & Wolf, 1992) mencionam que a arquitetura está interessada com a seleção de elementos arquiteturais, suas interações, e as restrições nesses elementos, provendo um framework no qual se satisfazem os requisitos e servindo como base efetiva para o reuso. Por sua vez, o referenciado trabalho cita que o projeto está interessado com a modularização e

interfaces detalhadas dos elementos de projeto, seus algoritmos e procedimentos, e os tipos de dados necessários para dar suporte à arquitetura e satisfazer os requisitos.

Em (Garlan & Shaw, 1993), a arquitetura de software é definida como "um nível de projeto que vai além dos algoritmos e estruturas de dados da computação". Segundo os autores, o projeto e especificação da estrutura geral do sistema surge como um novo tipo de problema. Questões estruturais incluem: organização geral e estrutura global de controle; protocolo para comunicação, sincronização e acesso de dados; atribuição de funcionalidade para elementos de projeto; escalonamento e performance; e seleção entre alternativas de projeto.

(Garlan & Shaw, 1996) citam que a "arquitetura de um sistema de software o define em termos de componentes computacionais e interações entre esses componentes (...) No mais, especificando a estrutura e topologia do sistema, a arquitetura mostra a correspondência entre os requisitos e elementos do sistema construído, provendo, por esse meio, princípios para justificar as decisões de projeto".

Finalmente, o IEEE (em inglês, *Institute of Electrical and Electronics Engineers*) definiu um padrão de recomendação sob sigla P1471, atualmente na versão 5.2, voltado para a descrição arquitetural. Em (IEEE P1471, 1999), a arquitetura é vista como "a organização fundamental de um sistema incorporada nos seus componentes, nos seus relacionamentos, entre si e para com o ambiente, e nos princípios guiando seu projeto e evolução".

Como se vê, ainda não há um consenso com relação a uma definição precisa para *arquitetura de sistemas*. Mais do que isso, o que se nota é uma grande dificuldade em definir qual o nível exato de abstração em que se deve definir a arquitetura de um sistema para ampliar ao máximo as vantagens dessa área. No mais, essas diferentes visões sobre o tema também servem para mostrar o alto grau de atividade de pesquisa na área. Em (Perry & Wolf, 1992) chega a ser mencionado que "[eles] acreditam que os anos 90 serão a década da arquitetura de software". Conforme foi observado, tal trabalho chega a contrastar os termos arquitetura e projeto, para evocar noções de codificação, abstração, padrões, treinamento formal (ou de arquitetos) e estilo.

A.2
Descrições formais de sistemas

O principal pré-requisito para que projetistas de software possam desenvolver sistemas que atendam às expectativas dos usuários é que os requisitos sejam não só identificados, como expressos precisamente. Isso requer o uso de alguma notação que defina, sem ambigüidades, os requisitos de software. Para evitar mal-entendidos entre as partes, é necessário, também, que essa notação possa ser entendida por ambos, para eventuais validações e correções.

Em fases mais avançadas do ciclo de vida do software, é necessário que o sistema seja especificado livre de ambigüidades para que, enfim, a implementação reflita as expectativas dos clientes. Uma abordagem útil para isso é disponibilizar ferramentas que automatizem a tarefa de traduzir a especificação formal em uma linguagem de programação.

Descrições formais possibilitam que sistemas sejam especificados livres de ambigüidade e com todos os requisitos mais facilmente identificados. Além disso, os sistemas passam a ser passíveis de teste e análise da correção e performance. Os benefícios de haver uma descrição de alto nível continua por todo o ciclo de vida, seja nas áreas de entendimento de software, gerenciamento de configuração, teste de regressão e no próprio desenvolvimento.

Segundo (Saiedian, 1993), há uma dificuldade natural no entendimento de descrições formais, chegando ao ponto de criar a falsa idéia que métodos formais são muito matemáticos e extremamente complicados. Tal trabalho aponta haver uma necessidade maior de enfatizar alguns cursos na formação em ciência da computação para promover uma base de conhecimento mais direcionada. Esses cursos são: matemática discreta; lógica matemática e especificação formal.

Para profissionais, esse estudo aponta como relevante o treinamento em matemática discreta através do estudo de teoria elementar de conjuntos e lógica. Depois, deve seguir um treinamento em um método formal específico e, finalmente, tutoriais e consultoria em projetos reais. É importante que estudos de caso sejam desenvolvidos para demonstrar a aplicabilidade de métodos formais, com a intenção de convencer os praticantes que os benefícios superam as dificuldades de transição e que os resultados são relevantes.

Em (Hall, 1990), métodos formais são tidos como controversos. De um lado, seus defensores afirmam que eles revolucionam o desenvolvimento, de outro, há aqueles que os acusam de serem impossivelmente difíceis. Em meio aos dois extremos, para a maioria das pessoas, os métodos formais são tão pouco familiares que ficaram sujeitos a certas crenças exageradas. Tais mitos são apontados em (Hall, 1990):

- *Métodos formais podem garantir que o software é perfeito.* A verdade é que eles são sujeitos a falha. Há um limite no que pode ser provado e erros podem ser cometidos nessas provas. No entanto, o uso de métodos formais permite que se demonstre haver a ausência de *bugs*, ao contrário dos testes convencionais em programas, que apenas os detectam. Além disso, é muito mais fácil encontrar erros usando métodos formais e ainda mais fácil demonstrá-los. Nesse aspecto, podem ser vistos como um enfoque científico para o desenvolvimento, já que suas especificações podem ser refutadas;
- *Todos os métodos formais têm a ver com provas em programas*. O fato é que todos eles têm a ver com especificações. Do ponto de vista econômico, a parte mais importante do desenvolvimento formal é a especificação do sistema, que na maioria dos projetos é a única parte formal. Isso porque a especificação formal do que um programa deverá fazer é pré-requisito para verificar se ele está correto;
- *Métodos formais são úteis apenas para sistemas de segurança crítica*. O fato é que especificações formais ajudam em qualquer sistema, como é confirmado em várias aplicações com sistemas não-críticos;
- *Métodos formais requerem matemáticos altamente treinados.* A verdade é que o conhecimento matemático necessário é fácil;
- *Métodos formais aumentam os custos de desenvolvimento.* Apesar de ser difícil comparar os custos com desenvolvimento de software usando métodos diferentes, o que se percebe é que a utilização de métodos formais apenas amplia a fase de especificação do sistema, mas a facilidade resultante na fase de implementação acaba reduzindo os custos;

- *Métodos formais são inaceitáveis a usuários*. Na prática, usuários e não-especialistas entendem e utilizam cada vez mais as notações formais para verificarem se o que elas representam estão de acordo com as necessidades reais;
- *Métodos formais não são usados em software real, de larga escala*. Atualmente se verifica a aplicação de métodos formais em muitos sistemas, o que derruba esse mito.

Complementando e revendo o trabalho anterior, em (Bowen & Hinchey, 1995) são apresentados mais outros sete mitos com relação ao uso de métodos formais:

- Eles retardam o processo de desenvolvimento.
- Faltam mais ferramentas para métodos formais.
- Métodos formais substituem os métodos tradicionais de engenharia de projeto.
- Métodos formais se aplicam apenas a software.
- Métodos formais são desnecessários.
- Métodos formais não têm suporte.
- Pessoas que conhecem métodos formais, os utilizam sempre.

O que se percebe é que há uma enorme confusão com relação à aplicação de métodos formais. Vários mal-entendidos são originados, em parte, pela falsa crença de leigos que notações formais são meros exercícios acadêmicos e não têm uso no mundo real. Quando se menciona métodos formais, eles são ou sujeitos a grande crítica ou embutidos de vantagens excessivas.

Apêndice B
Linguagens com suporte à QoS

Algumas ADLs oferecem suporte à QoS em ambientes e categorias de serviço específicas. Sua função é facilitar o desenvolvimento de um serviço com QoS, através de mecanismos para gerência de recursos embutidos na linguagem. Este Apêndice descreve dois exemplos de tais linguagens: Xelha, apresentada na Seção B1; e CBabel, na Seção B2.

B.1
Xelha

A ADL Xelha provê, segundo termos próprios, o suporte nas camadas de aplicação e *middleware* para a especificação do gerenciamento estático e do gerenciamento dinâmico de QoS (Duran-Limon & Blair, 2000). O primeiro está relacionado com especificação do nível desejado de QoS, enquanto o segundo equivale aos mecanismos de sintonização usados para manter esse nível.

O gerenciamento estático de QoS inclui a configuração inicial dos componentes e da quantidade de recursos necessários para garantir o nível desejado de QoS. Em Xelha, os aspectos do gerenciamento estático são definidos em termos de *grafos de tarefas* juntos com uma sentença de propriedades de QoS associada (como será visto em detalhes adiante). Uma *tarefa* modela interações entre componentes com um conjunto de recursos associados.

O gerenciamento de QoS dinâmico envolve a monitoração em tempo de execução e reconfiguração dinâmica de componentes e recursos. Seus aspectos são organizados através de estruturas constituídas de *componentes de gerenciamento*, que são responsáveis tanto pelo monitoramento da QoS, quanto pela seleção adequada de uma estratégia de adaptação no caso de uma violação de contrato de serviço.

Grafos de tarefas são definidos em termos de pontos de trocas de tarefas (*switching points*). Tais pontos denotam a operação em uma tarefa que dispara uma outra tarefa. Eles são expressos em conjunto com a interface e com o componente ao qual pertencem. Esse enfoque é suficiente para especificar o início

e fim de tarefas. Sentenças condicionais são opcionalmente definidas para determinar quando uma troca de tarefa deve ser efetuada, de acordo com a atual. Tarefas compostas são especificadas definindo a inclusão de outras tarefas como suas sub-tarefas. Sub-tarefas herdam a medida de importância (*importance*) de sua super-tarefa. Tal medida é usada para definir quão crítica é uma tarefa, de forma que, em caso de contenção de recursos, aquelas com um valor mais alto de importância tenham precedência sobre valores mais baixos.

A especificação de QoS permite que se definam as propriedades de QoS, que são associadas a um grafo de tarefas, em três principais categorias: *timeliness*; volume; e confiabilidade. Primeiramente, propriedades *timeliness* envolvem o retardo fim a fim de interações multimídia e variações de retardo, chamadas *jitter*, medidas em milissegundos. Propriedades de volume têm relação com o tráfego de dados e são medidas em quadros ou bytes por segundo. Finalmente, cada propriedade de confiabilidade é definida em termos dos *pontos finais das tarefas*. Um ponto final de tarefa é definido como uma tripla contendo um componente, uma interface e uma operação.

O papel dos *componentes de gerenciamento* envolve basicamente os aspectos de monitoramento e controle. O primeiro inclui coletores de eventos, que fazem interface com a implementação base, e monitores, que detectam violações de QoS. Aspectos de controle incluem seletores e ativadores de estratégia. Os seletores são responsáveis pela decisão de qual estratégia aplicar sobre uma ocorrência de violação de QoS. Os ativadores são, então, encarregados de pôr em prática a estratégia de adaptação definida por sua implementação.

Em Xelha, uma tarefa pode ter ou não uma estrutura de gerenciamento de QoS associada, de forma que se uma hierarquia é definida, uma estrutura associada a uma tarefa de nível superior pode ser suficiente para todos os níveis inferiores. Contudo, um maior refinamento do gerenciamento de QoS pode ser obtido ao se definir estruturas de adaptação de mais baixo nível a tarefas de mais baixo nível. Coletores de eventos são definidos pela interface de componente que se deseja observar. Monitores e seletores de estratégia são autômatos temporais, modelados separadamente e depois unidos em um único autômato, cujo comportamento é implementado por um componente pré-construído, que pode ser automaticamente gerado pelo uso de algumas ferramentas. O ativador de estratégia é implementado por um outro componente pré-construído e a

configuração de componentes da estrutura de gerenciamento de QoS é definida por um grafo.

A Figura B.1 a seguir, extraída de (Duran-Limon & Blair, 2000), mostra um exemplo de especificação de tarefa e estrutura de gerenciamento de QoS. Recursos são reservados de acordo com a especificação definida na seção de tarefas (*tasks*), em termos do nome da tarefa, número máximo de instâncias para ela, e a cápsula, onde executará. Para esse fim, as especificações de QoS são processadas por um interpretador, que as traduz em recursos específicos, levando em conta o número máximo de instanciações.

```
Def task transmitAu.marshall:
    switching points:
        srcStub:CTRL:start [if task x ]
    qos specifications:
        delay(srcStub:IN:read, streamConn:IN:put) = 5
        throughput(srcStub:OUT:put) = 64

Def task transmitAu includes transmitAu.marshall,
tansmitAu.unmarshall:
    importance: 5
    qos specifications:
        delay(streamConn:IN:put, streamConn:OUT:put) = 10
        packet_loss(streamConn:IN:put, streamConn:OUT:put) = 5
        delay(srcStub:IN:read, sinkStub:OUT:write) = 20
        jitter(srcStub:IN:read, sinkStub:OUT:write) = 1
    qos management structure:
        collector:
            COLLECT: (sinkStub, COLLECT)
        timed automaton:
            automaton: Tautomaton_3
        strategy activator:
            activator: Activator_V1
        qos management graph:
            interfaces:
                automatonIN: ( automaton, IN )
                automatonOUT: ( automaton, OUT )
                activatorIN: ( activator, IN )
                activatorOUT ( activator, OUT )
                streamConnCTRL: ( streamConn, CTRL)
            edges:
                ( COLLECT, automatonIN )
                ( automatonOUT, activatorIN )
                ( activatorOUT, streamConnCTRL )
```

Figura B.1 – Especificação de tarefa e estrutura de gerenciamento de QoS

Por fim, a reserva de recursos é definida na especificação dos componentes compostos de mais alto nível, que representam a estrutura mais alta do sistema, como mostrado na Figura B.2.

```
Def component YSystem:
   components: ...
   connectors: ...
   composition graph: ...
   tasks:
       sendData.marshall, 13, “capsule 1”
       sendData.unmarshall, 2, “capsule 2”
       sendVoice.marshall, 20, “capsule 3”
       sendVoice.unmarshall, 3, “capsule 4”
```

Figura B.2 – Especificação de componente

Como Xelha é *resource-aware*, os grafos de tarefas tornam suas especificações complexas, especialmente em função das descrições para a gerência de recursos. No entanto, o suporte fornecido pela plataforma OpenORB, como um todo, é eficiente (Duran-Limon & Blair, 2000).

B.2 CBabel

Cbabel é uma ADL em estágio de desenvolvimento que faz parte do projeto R-RIO (Reflective-Reconfigurable Interconnectable Objects), que se constitui em um ambiente de programação distribuído com integração de alguns conceitos de configuração e programação reflexiva. R-RIO é baseado em (Lobosco, 1999) (Sztajnberg & Loques, 2000):

- Uma metodologia de projeto com base em componentes que conta com a composição de módulos funcionais e conectores não estritamente funcionais para construir arquiteturas de aplicação;
- Uma linguagem de descrição de arquitetura, chamada CBabel, na qual são descritas as seleções de componentes, interconexões e elementos compostos que serão usados para estruturar as aplicações, e aspectos não necessariamente funcionais como coordenação, distribuição, tolerância a falha e QoS, permitindo a padronização dessas aplicações; e
- Um middleware de suporte reflexivo que permite se gerenciar a configuração de arquiteturas de aplicações.

A ADL CBabel está sendo estendida para permitir a especificação de contratos de QoS, tendo como base um subconjunto da linguagem QML (Frolund & Koistinen, 1998) e adaptando sua estrutura e terminologia para um escopo de descrição de arquitetura. Assim, os contratos de QoS são definidos separadamente dos componentes e uma cláusula QoS-profile os relaciona. Em (Sztajnberg & Loques, 2000), um contrato simples é descrito em um QoS-connector na Figura B.3.

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

Figura B.3 - Contrato de QoS em um QoS-connector

Os módulos Producer, Consumer e Buffer devem ter seus QoS-profiles já descritos quando forem associados a um contrato. Permite-se também uma política de tratamento de violação associada com um contrato, que é o caso de renegotiate e terminate.

Uma arquitetura de QoS descrita com CBabel, e com ADLs com suporte à QoS de uma forma geral, é passível de ser submetida a uma checagem de coerência e consistência, o que implica o projetista, ainda em tempo de projeto e não apenas na implementação, poder avaliar se a QoS desejada poderá ser suportada pelos recursos disponíveis. Posteriormente, em tempo de execução, a disponibilidade de recursos determina se, em um dado momento, é possível estabelecer um serviço com parâmetros específicos de QoS.

As descrições de contratos de QoS em CBabel são mapeadas em ações e dados que são armazenados como informações de meta-serviço. O middleware R-RIO interpreta essa informação para configurar os recursos adequados e iniciar os procedimentos apropriados para garantir a QoS. Dois módulos são usados para esse fim: QoS Experts e QoS Monitors. O primeiro interpreta as informações de QoS do meta-nível e sintetiza um QoS-connector que irá garantir a QoS especificada. O segundo, por sua vez, é composto de módulos especiais que irão continuamente medir os parâmetros de QoS de fato e checar se eles estão na faixa requisitada. De acordo com a política de tratamento de violações especificada no

contrato, uma ação é tomada, o que pode incluir a chamada ao módulo QoS Experts para reconfigurar o conector.

CBabel permite a especificação de contratos de QoS de forma bastante simples, tanto quanto em QML, o que é uma vantagem do ponto-de-vista de entendimento e treinamento. A técnica de seleção de conectores num repositório, para suportar a QoS requisitada, exige uma infra-estrutura diversa para disponibilizar diferentes categorias de QoS. É justamente esse mecanismo de mapeamento uma importante vantagem do ambiente R-RIO, já que novas categorias de QoS podem ser suportadas tantas quanto novos Experts e Monitors sejam criados para tal fim.