ALMIR ROGÉRIO CAMOLESI

Proposta de um Gerador de Ambientes para a Modelagem de Aplicações usando Tecnologia Adaptativa

Tese apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Doutor em Engenharia Elétrica

São Paulo
2007

ALMIR ROGÉRIO CAMOLESI

Proposta de um Gerador de Ambientes para a Modelagem de Aplicações usando Tecnologia Adaptativa

Tese apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Doutor em Engenharia Elétrica

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. João José Neto

São Paulo
2007

Dedico este trabalho
a Deus, que me deu força e inspiração;
e a meus pais, que nunca mediram esforços para que eu pudesse alcançar meus objetivos e realizar meus sonhos.

AGRADECIMENTOS

Ao Prof. Dr. João José Neto, pelo incentivo e apoio irrestrito prestado para a realização deste trabalho, por suas orientações técnicas e por compartilhar de suas experiências pessoais em nossas incontáveis reuniões;

Aos Prof. Dr. Ricardo Luis de Azevedo da Rocha e Prof. Dr. Paulo Sérgio Muniz Silva, que participaram da banca do exame de qualificação e, muito contribuíram com sugestões, na avaliação da proposta inicial deste trabalho;

Ao amigo Rogério José da Silva (Lélo), pela amizade, dedicação e apoio na leitura de meus textos e pelo incentivo no aperfeiçoamento de minha expressão escrita;

Aos meus familiares e amigos, pela compreensão dos momentos de ausência, os quais se justificam exatamente pela realização deste trabalho;

A minha irmã Mara, por estar sempre comigo, por me auxiliar quando possível nas suas discussões e sugestões sobre elaborações de trabalhos científicos e por permitir o intercâmbio de informações em outras áreas do conhecimento.

A minha namorada Daniela Antonio (Dany), também pela compreensão dos momentos de dedicação insuficientes ao amor que a mim dedica e, especialmente, pelo apoio nos maiores momentos de dificuldade encontrados na fase final de realização deste trabalho, justamente quando a encontrei;

À Fundação Educacional do Município de Assis (FEMA), pela oportunidade profissional concedida e solidificada com méritos ao longo dos últimos anos e, hoje, especialmente pelo apoio prestado para que este trabalho fosse realizado.

RESUMO

Este trabalho tem por objetivo propor um gerador de ambientes (metambiente) que possibilite a geração automática de ambientes para o projeto de aplicações adaptativas. Tal gerador fundamenta-se nos conceitos de Tecnologia Adaptativa e permite a definição de dispositivos adaptativos dirigidos por regras. No desenvolvimento deste trabalho foram propostos um método para definição de dispositivos adaptativos, a arquitetura geral de um ambiente para o projeto de aplicações adaptativas e a arquitetura para um gerador de ambientes para a modelagem de aplicações usando um dispositivo adaptativo específico. Com base nos conceitos propostos foram realizadas a implementação de algumas ferramentas para validar os mesmos e desenvolvidos alguns experimentos com o propósito de demonstrar a utilização de tais ferramentas e dos dispositivos adaptativos no projeto de aplicações.

Palavras-chave: Dispositivos Adaptativos. Meta-sistemas. Modelagem de aplicações.

ABSTRACT

This research aims at proposing an environment generator (meta-environment) which makes it possible the automatic environment generation for the project of adaptive applications. This generator establishes its concepts in Adaptive Technology and allows the definition of rule-driven adaptive devices. When developing the present study, we considered the general architecture of an environment for the project of adaptive applications and the architecture for an environment generator of modeling applications using a specific adaptive device as a method for definition of adaptive devices. Based on the concepts mentioned, the implementation of some tools were done to show such concepts and some experiments were made to demonstrate the use of such tools and adaptive devices in the project of applications.

Keywords: Adaptive devices. Meta-systems. Modeling applications.

SUMÁRIO

1. Introdução 17

1.1. Justificativa 18

1.2. Objetivos 20

1.3. Apresentação dos capítulos 20

2. Elementos Conceituais 22

2.1. Tecnologia Adaptativa 22

2.1.1. Formalização geral para dispositivos não-adaptativos. 27

2.1.2. Formalização geral para dispositivos adaptativos 28

2.2. Metadados 32

2.2.1. Extensibilidade e refinamento de metadados 34

2.2.2. Propriedades de metadados 35

3. Proposta de um gerador de ambientes para a modelagem de aplicações adaptativas 37

3.1. Método para a definição de dispositivos adaptativos 37

3.2. Proposta do Ambiente para o Projeto de Aplicações usando Tecnologia Adaptativa 40

3.2.1. Arquitetura geral do Ambiente para o Projeto de Aplicações Adaptativas 42

3.2.2. Ferramentas de Edição 44

3.2.3. Ferramenta para simulação e verificação 48

3.2.4. Ferramenta para a Produção de Representação Física 50

3.3. Proposta do gerador de ambientes 51

3.3.1. Definição de metambientes 52

3.3.2. Arquitetura do gerador de ambientes 54

3.3.3. Arquitetura do *framework* para um ambiente geral 56

3.3.4. Ferramenta para a definição dos elementos conceituais de dispositivos 58

3.3.5. Ferramenta para a definição dos objetos gráficos de dispositivos 58
3.3.6. Ferramenta para a definição da Forma de Operação de um Dispositivo 60
3.4. Modelo lógico para dispositivos adaptativos 61
3.4.1. Organização do Modelo Lógico.....62

4. Implementação do gerador de ambientes para a modelagem de aplicações adaptativas..... 66
4.1. Escopo do Ambiente para a Modelagem de Aplicações Adaptativas e Gerador de Ambientes.....66
4.2. Implementação da interface do ASPAA e do ELML.....68
4.2.1. Implementação física do ELML 71
4.3. Simulador de aplicações adaptativas 75
4.3.1. Implementação do SAA.....76

5. Experimentos realizados 85
5.1. Autômato de Estados Finitos Adaptativo.....85
5.1.1. Formalização de Autômato de Estados Finitos 85
5.1.2. Formalização de Autômato de Estados Finitos Adaptativo 87
5.1.3. Modelagem da aplicação *Cadent* usando AEF_{Adp} 90
5.1.1. Mapeamento do dispositivo AEF_{Adp} e da aplicação *CadEnt* para modelo lógico 93
5.2. Rede de Petri Adaptativa.....96
5.2.1. Formalização Rede de Petri.....97
5.2.2. Formalização Rede de Petri Adaptativa 99
5.2.3. Modelagem da aplicação AVSinc usando RP_{Adp}.....102
5.2.1. Mapeamento do dispositivo RP_{Adp} e da aplicação *AVSinc* para modelo lógico 104
5.3. ISDL Adaptativo.....108
5.3.1. Formalização ISDL.....110
5.3.2. Formalização $ISDL_{Adp}$ 114
5.3.3. Modelagem da aplicação *CpCl* usando $ISDL_{Adp}$.....119

5.3.1. Mapeamento do dispositivo $ISDL_{Adp}$ e da aplicação *CpCI* para modelo lógico. 126

6. Considerações finais 131

6.1. Avaliação do trabalho 131

6.2. Contribuições 134

6.3. Sugestões para trabalhos futuros 134

Referências bibliográficas 136

APÊNDICE 142

A. Descrição do Modelo Lógico 142

A.1. Descrição de classes do Pacote da Camada de Dispositivo 142

A.1.1. Classe Dispositivo 142

A.1.2. Classe Tipo de Componente 143

A.1.3. Classe Tipo de Conexão 144

A.1.4. Classe Tipo de Atributo 144

A.2. Descrição de classes do Pacote da Camada de Especificação Subjacente 145

A.2.1. Classe Projeto 145

A.2.2. Classe Componente 146

A.2.3. Classe Atributo de Componente 147

A.2.4. Classe Conexão 147

A.2.5. Classe Atributo de Conexão 148

A.3. Descrição de Classes do pacote Camada de Especificação Adaptativa 149

A.3.1. Classe Função Adaptativa 150

A.3.2. Classe Ação Adaptativa 150

A.3.3. Classe Acoplamento 151

A.3.4. Classe Gerador 152

A.3.5. Classe Parâmetros 152

A.3.6. Classe Variáveis 153

A.3.7. Classe Função Adaptativa de Ação Adaptativa 153

A.3.8. Classe Parâmetros de Ação Adaptativa 154
A.3.9. Classe Atributo de Conexão de Ação Adaptativa 154
A.3.10. Classe Atributo de Componente de Origem de Ação Adaptativa 155
A.3.11. Classe Atributo de Componente de Destino de Ação Adaptativa 155
A.4. Descrição de classes do Pacote da Camada de Memória 155
A.4.1. Classe Cadeia de Entrada 155
A.4.2. Classe Atributo de Cadeia de Entrada 156
A.4.3. Classe Componente Ativo 156

LISTA DE ILUSTRAÇÕES

Figura 1. Arquitetura do Processo Integrado para o projeto de aplicações adaptativas 19
Figura 2. Método para definição de dispositivos adaptativos 38
Figura 3. Ambiente para projeto de aplicações usando Tecnologia Adaptativa 41
Figura 4. Modelo de entidades do Ambiente para o Projeto de Aplicações Adaptativas 43
Figura 5. Representação da subentidade Ferramenta de Edição 44
Figura 6. Refinamento da subentidade Editor Textual. 45
Figura 7. Refinamento da subentidade Editor Gráfico 47
Figura 8. Refinamento subentidade Editor Linguagem Modelo Lógico 47
Figura 9. Representação da subentidade Ferramenta de Análise 48
Figura 10. Refinamento subentidade Simulador Aplicação Adaptativa 49
Figura 11. Refinamento da subentidade Núcleo Adaptativo 49
Figura 12. Refinamento da subentidade Verificador Aplicação Adaptativa 50
Figura 13. Refinamento subentidade Ferramenta para a Produção de Representação Física 51
Figura 14. Hierarquia de metamodelos 52
Figura 15. Arquitetura geral para geradores de ambientes 54
Figura 16. Arquitetura geral do gerador de ambientes 55
Figura 17. Definição de um F*ramework* para um Ambiente Geral 57
Figura 18. Arquitetura da Ferramenta para a Definição de Objetos Gráficos 60
Figura 19. Arquitetura da Ferramenta para a Definição da Forma de Operação de um Dispositivo 61
Figura 20. Organização de camadas do Modelo Lógico 63
Figura 21. Diagrama de classes do Modelo Lógico para Dispositivos Adaptativos 64
Figura 22. Arquitetura do APAA 67
Figura 23. Ambiente Simplificado para o Projeto de Aplicações Adaptativas 67
Figura 24. Interface da ferramenta para administração da base de dados do Modelo Lógico 72
Figura 25. Interface do ambiente Netbeans - Projeto ASPAA 73
Figura 26. Interface do menu de opções ASPAA 74

Figura 27. Interface de operação do ELML 74
Figura 28. Diagrama de atividades do SAA 76
Figura 29. Diagrama de classes da Interface do SAA 77
Figura 30. Descrição geral para um método Mostrar 81
Figura 31. Diagrama de atividades desempenhadas pelo método de simulação do SAA 82
Figura 32. Interface do Simulador de Aplicação Adaptativa 84
Figura 33. Dispositivo Autômato de Estados Finitos Adaptativo 87
Figura 34. Comportamento Inicial aplicação *CadEnt* 91
Figura 35. Comportamento aplicação *CadEnt* após reconhecer um símbolo *a* 91
Figura 36. Comportamento aplicação *CadEnt* após reconhecer um símbolo 92
Figura 37. Comportamento aplicação *CadEnt* após reconhecer um símbolo *b* 92
Figura 38. Comportamento aplicação *CadEnt* após reconhecer o segundo símbolo *b* 92
Figura 39. Representação gráfica de Rede de Petri 97
Figura 40. Dispositivo Rede de Petri Adaptativa 99
Figura 41. Comportamento inicial aplicação *AVSinc* 103
Figura 42. Função adaptativa F(..) 104
Figura 43. Comportamento aplicação *AVSinc* modificado 104
Figura 44. Modelo de entidades aplicação VoD 109
Figura 45. Algumas ações e relações de causalidade ISDL 113
Figura 46. Exemplo de estruturação de comportamento ISDL 114
Figura 47. Estrutura dispositivo $ISDL_{Adp}$ 115
Figura 48. Representação de ações adaptativas $ISDL_{Adp}$ 117
Figura 49. Representação de funções e ações elementares adaptativas $ISDL_{Adp}$ 118
Figura 50. Situação inicial MS-Word Office XP 2003 120
Figura 51. Habilitação da opção *"Copiar"* 120
Figura 52. Operações disponíveis para *"Colar Especial"* para objetos do tipo *texto* 121
Figura 53. Operações disponíveis para *"Colar Especial"* para objetos tipo *imagem* 122
Figura 54. Modelo entidades aplicação *CpCl* 122
Figura 55. Comportamento inicial aplicação *CpCl* 123
Figura 56. Subcomportamentos *C_Cop_Txt* e *C_Cop_Img* 124

Figura 57. Funções adaptativas *FCop(...)* 125
Figura 58. Funções adaptativas FReset(...) 125
Figura 59. Comportamento *Cp_Cl* após execução de funções adaptativas 126
Figura 60. Diagrama de classes da Camada de Dispositivo 142
Figura 61. Diagrama de classes da Camada de Especificação Subjacente 145
Figura 62 . Diagrama de classes da Camada de Especificação Adaptativa 149
Figura 63. Diagrama de classes da Camada de Memória 155

LISTA DE TABELAS

Tabela 1. Lista de Eventos do Editor que usa a Linguagem do Modelo Lógico 69
Tabela 2. Mapeamento de objetos da classe Dispositivo para o dispositivo AEF_{Adp}.93
Tabela 3. Mapeamento de objetos da classe Tipo de Componente para o dispositivo AEF_{Adp} 93
Tabela 4. Mapeamento de objetos da classe Tipo de Conexão para o dispositivo AEF_{Adp} 93
Tabela 5. Mapeamento de objetos da classe Tipo de Atributo para o dispositivo AEF_{Adp} 94
Tabela 6. Mapeamento de objetos da classe Projeto – Aplicação *CadEnt* 94
Tabela 7. Mapeamento de objetos da classe Componente – Aplicação *CadEnt* 94
Tabela 8. Mapeamento de objetos da classe Conexão – Aplicação *CadEnt* 94
Tabela 9. Mapeamento de objetos da classe Atributo Conexão – Aplicação *CadEnt* 95
Tabela 10. Mapeamento de objetos da classe Função Adaptativa – Aplicação *CadEnt* 95
Tabela 11. Mapeamento de objetos da classe Ação Adaptativa – Aplicação *CadEnt* 95
Tabela 12. Mapeamento de objetos da classe Atributo de Conexão de Ação Adaptativa – Aplicação *CadEnt* 95
Tabela 13. Mapeamento de objetos da classe Acoplamento – Aplicação *CadEnt*....95
Tabela 14. Mapeamento de objetos da classe Parâmetros – Aplicação *CadEnt* 96
Tabela 15. Mapeamento de objetos da classe Gerador – Aplicação *CadEnt* 96
Tabela 16. Mapeamento de objetos da classe Dispositivo – Dispositivo RP_{Adp} 105
Tabela 17. Mapeamento de objetos da classe Tipo de Componente – Dispositivo RP_{Adp} 105
Tabela 18. Mapeamento de objetos da classe Tipo de Conexão – Dispositivo RP_{Adp} 105
Tabela 19. Mapeamento de objetos da classe Tipo de Atributo – Dispositivo RP_{Adp} 105
Tabela 20. Mapeamento de objetos da classe Projeto – aplicação *AVSinc*.....106
Tabela 21. Mapeamento de objetos da classe Componente – aplicação *AVSinc*...106

Tabela 22. Mapeamento de objetos da classe Atributo de Componente – aplicação *AVSinc* 106
Tabela 23. Mapeamento de objetos da classe Conexão – aplicação *AVSinc* 106
Tabela 24. Mapeamento de objetos da classe Atributo de Conexão – aplicação *AVSinc* 107
Tabela 25. Mapeamento de objetos da classe Função Adaptativa – aplicação *AVSinc* 107
Tabela 26. Mapeamento de objetos da classe Ação Adaptativa – aplicação *AVSinc* 107
Tabela 27. Mapeamento de objetos da classe Acoplamento – aplicação *AVSinc* 107
Tabela 28. Mapeamento de objetos da classe Parâmetros – aplicação *AVSinc* 108
Tabela 29. Mapeamento de objetos da classe Atributo de Conexão de Ação Adaptativa – aplicação *AVSinc* 108
Tabela 30. Mapeamento de objetos da classe Atributo de Componente de Origem de Ação Adaptativa – aplicação *AVSinc* 108
Tabela 31. Mapeamento de objetos da classe Atributo de Componente de Destino de Ação Adaptativa – aplicação *AVSinc* 108
Tabela 32. Mapeamento de objetos da classe Dispositivo para o dispositivo $ISDL_{Adp}$ 126
Tabela 33. Mapeamento de objetos da classe Tipo de Componente para o dispositivo $ISDL_{Adp}$ 127
Tabela 34. Mapeamento de objetos da classe Tipo de Conexão para o dispositivo $ISDL_{Adp}$ 127
Tabela 35. Mapeamento de objetos da classe Tipo de Atributo para o dispositivo $ISDL_{Adp}$ 127
Tabela 36. Mapeamento de objetos da classe Projeto – Aplicação *CpCI* 127
Tabela 37. Mapeamento de objetos da classe Componente – Aplicação *CpCI* 128
Tabela 38. Mapeamento de objetos da classe Atributo de Componente – Aplicação *CpCI* 128
Tabela 39. Mapeamento de objetos da classe Conexão – Aplicação *CpCI* 129
Tabela 40. Mapeamento de objetos da classe Atributo de Conexão – Aplicação *CpCI* 129
Tabela 41. Mapeamento de objetos da classe Função Adaptativa – Aplicação *CpCI* 129

Tabela 42. Mapeamento de objetos da classe Ação Adaptativa – Aplicação *CpCI*. 130
Tabela 43. Mapeamento de objetos da classe Acoplamento – Aplicação *CpCI*..... 130
Tabela 44. Mapeamento de objetos da classe Parâmetros – Aplicação *CpCI*..... 130

1. Introdução

Aplicações complexas são caracterizadas por componentes e aspectos cuja estrutura e comportamento, comumente, não podem ser descritos por um único formalismo - devido à diferença de natureza existente entre os modelos – (LARA; VANGHELUWE; ALFONSECA, 2004).

Lara, Vangheluwe e Alfonseca (2004), citam, por exemplo, a modelagem de um controlador de temperatura e nível – volume - de um recipiente. O controlador de temperatura pode ser descrito por um formalismo discreto, como Rede de Petri (PETRI, 1962) ou Statecharts (HAREL et al., 1987). Considerando que o comportamento do controlador de líquido deve descrever a variação de volume, este deverá ser descrito com um formalismo contínuo (como, por exemplo, Equações Diferenciais Ordinárias).

Uma das técnicas utilizadas para modelar aplicações complexas é o uso de uma modelagem que utilize múltiplos formalismos. Utilizando-se desta técnica, as diferentes partes do sistema são modeladas servindo-se de mais de um formalismo. Para analisar uma aplicação expressa por multiformalismo não basta avaliar cada componente isolado. O projetista tem que considerar toda a aplicação. Por isto, a modelagem realizada com multiformalismo deve ser fundamentada na conversão de todos os componentes de cada formalismo em uma única representação comum. Desta forma, a aplicação será representada por diferentes formalismos para ser corretamente analisada ou simulada. No projeto AToM^3 (ATOM3, 2007), foi desenvolvido um ambiente multiparadigma para a modelagem e desenvolvimento de aplicações utilizando-se de multiformalismos. Tal ambiente fundamenta-se em uma representação intermediária comum definida para o mesmo e que permite descrever os diversos formalismos e aplicações representados.

Outra característica presente em aplicações complexas é a característica de que as mesmas possuem em relação à possibilidade de modificarem o seu comportamento em tempo de execução. Tais aplicações possuem um comportamento inicial definido por um conjunto de ações que desempenham suas funções elementares, e durante a execução podem ter o seu comportamento modificado para dar suporte a novas funcionalidades. Tais modificações são

decorrentes dos estímulos de entrada a que são submetidos no sistema e/ou da ocorrência de suas ações internas.

Uma técnica utilizada para auxiliar os projetistas na modelagem de aplicações com comportamento modificável é a tecnologia adaptativa (NETO, 1993). A tecnologia adaptativa envolve um dispositivo não-adaptativo (subjacente) já existente em uma camada adaptativa que permite realizar mudanças no comportamento da aplicação definida. É possível citar, por exemplo, trabalhos relacionados a reconhecedores sintáticos adaptativos (NETO, 1988), os *Statecharts* Adaptativos (ALMEIDA, 1995) - empregados na modelagem de sistemas reativos - e a modelagem de aplicações complexas com base no ISDL Adaptativo (CAMOLESI; NETO, 2004a), o qual está descrito no capítulo 5.3 deste trabalho.

O gerador de ambientes proposto neste trabalho utilizar-se-á dos conceitos de multiformalismo e de tecnologia adaptativa na busca de obtenção de ambientes para auxiliar os projetistas na modelagem de suas aplicações. Na seqüência, serão descritas as formas de processo de produção de aplicações complexas, suas vantagens, desvantagens e necessidades encontradas para o projeto e seus respectivos desenvolvimentos. Tais conceitos fundamentam as justificativas para a realização deste trabalho.

1.1. Justificativa

Quando um projetista vier a realizar o projeto de uma nova aplicação e executar as tarefas definidas para o ciclo de vida de desenvolvimento de aplicações utilizando tecnologia adaptativa e multiformalismo, poderá desempenhar o seu trabalho usando um ambiente que dê suporte às tarefas a serem desenvolvidas.

Com o objetivo de facilitar o trabalho do projetista, sugere-se o Processo Integrado para o projeto de aplicações utilizando tecnologia adaptativa, representado na Figura 1.

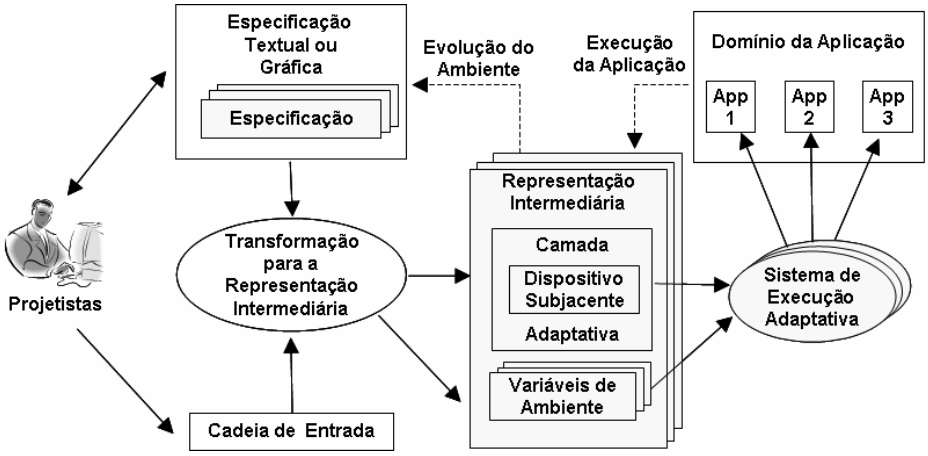


Figura 1. Arquitetura do Processo Integrado para o projeto de aplicações adaptativas

O Processo Integrado tem por objetivo tornar mais prática a realização do projeto de aplicações valendo-se de tecnologia adaptativa e constitui-se das seguintes fases: fase de especificação; fase de transformação da especificação; e, por fim, fase de validação, simulação e verificação da especificação. Na fase de especificação da aplicação, utilizando-se de uma ferramenta textual ou gráfica, será realizada a especificação da aplicação desejada. Em seguida, ocorrerá a transformação da especificação produzida para uma representação intermediária da ferramenta e, com base na representação obtida, o projetista poderá informar seqüências de entrada e avaliar o comportamento da aplicação que foi especificada. Ocorrendo inconsistências ou erros, o projetista poderá realizar as modificações na especificação e reiniciar o processo.

Para que o Processo Integrado seja viável, fazem-se necessárias modificações nos ambientes computacionais já existentes. O trabalho de modificação de tais ferramentas nem sempre pode ser efetuado pelo fato de as ferramentas existentes possuírem restrições de acesso por conta de normas de proteção à propriedade e por isso seus códigos-fontes não serem abertos. Isso também ocorre, mesmo na possibilidade de acesso ao código-fonte de uma ferramenta, se a arquitetura desta não permitir o acoplamento da camada adaptativa e de suas funcionalidades. Nestes casos, uma nova ferramenta deve ser desenvolvida e, para tal, deve ser empregada uma grande equipe de desenvolvimento e/ou também uma grande quantidade de tempo para a obtenção da mesma. Tem-se observado que a ausência de ferramentas de boa qualidade em um pequeno prazo tem dificultado a utilização dos formalismos adaptativos em ambientes reais no desenvolvimento de aplicações. Neste contexto, propõe-se este

trabalho que consiste na proposta de um gerador de ambientes para formalismos adaptativos que dê suporte a multiformalismos. Na próxima subseção serão descritos tanto os objetivos deste trabalho.

1.2. Objetivos

Neto (2001) apresentou uma formulação geral para dispositivos adaptativos. Tal formulação fundamenta-se em um núcleo constituído por um dispositivo não-adaptativo dirigido por regras, acrescido de uma camada adaptativa que provê os recursos de mecanismos adaptativos ao mesmo. Ao se desenvolver aplicações com base em um dispositivo adaptativo, faz-se necessária a utilização de um método para o projeto de tais aplicações. Neste trabalho é proposto um método para a definição de dispositivos adaptativos que traz facilidades aos especialistas na realização de seus trabalhos de extensão de seus dispositivos aos conceitos de Tecnologia Adaptativa. Para auxiliar o método para a definição de dispositivos adaptativos e a realização da implementação de ferramentas para dar suporte aos projetistas de aplicações durante a fase de modelagem encontra-se o objetivo deste trabalho que é a proposta de um gerador de ambientes para a modelagem de aplicações usando tecnologia adaptativa. Tal gerador deve permitir a um especialista em um determinado dispositivo não-adaptativo, depois de acrescentar a camada adaptativa ao formalismo, realizar o seu mapeamento para uma representação intermediária, obter de forma quase automática um ambiente para o projeto de aplicações. Tal ambiente deve permitir multiformalismos e a possibilidade de representações das características adaptativas das aplicações. Sua estrutura deve conter ferramentas que permitam a edição de especificações e, posteriormente, a simulação, a verificação e a implementação das aplicações projetadas.

1.3. Apresentação dos capítulos

Este trabalho foi organizado em nove capítulos. No Capítulo 1 são apresentados os motivacionais e os objetivos desse trabalho. No Capítulo 2, tem-se

uma revisão de literatura referente aos principais conceitos teóricos de Tecnologia Adaptativa e metadados.

No Capítulo 3 é apresentada a estrutura de um ambiente completo para o projeto de aplicações adaptativas e a arquitetura geral para o gerador destes ambientes. Finalizando a arquitetura proposta, no Capítulo 4 é descrita a arquitetura simplificada de um ambiente para a modelagem de aplicações adaptativas e realizada a implementação da mesma. O ambiente definido deverá implementar os conceitos teóricos apresentados neste trabalho e permitir a obtenção de ferramentas para a especificação e a simulação de aplicações.

O Capítulo 5 descreve os experimentos realizados com base nos conceitos e arquitetura definidos. Os experimentos foram realizados com foco na definição de novos dispositivos, projeto de aplicações com o uso dos dispositivos desenvolvidos, mapeamento dos novos dispositivos e aplicações para o modelo lógico e simulação das aplicações nas ferramentas desenvolvidas. Finalmente, no Capítulo 6, uma avaliação do trabalho é realizada, são descritas as principais contribuições e as perspectivas de novos trabalhos.

2. Elementos Conceituais

Neste capítulo, inicialmente serão definidos os elementos os conceitos de Tecnologia Adaptativa, uma visão geral sobre o estado da arte mesma e a representação geral para dispositivos adaptativos dirigidos por regras. Na seqüência, faz-se uma revisão dos principais elementos de metadados, que serão utilizados na definição do modelo lógico proposto.

2.1. Tecnologia Adaptativa

Um dos primeiros trabalhos relacionado ao estudo da Tecnologia Adaptativa é o ilustrado em (NETO e MAGALHÃES, 1981), que fornece uma visão de métodos de análise sintática e de geração de reconhecedores sintáticos. Este trabalho foi resultado de um primeiro esforço em busca da inclusão dos conceitos de mecanismos adaptativos em um sistema para apoio à construção de compiladores.

Neto (1983) elaborou uma extensão do modelo inicial que exibe a capacidade de incorporar funções de transdução sintática. Como continuação dos trabalhos, foi desenvolvida uma versão mais prática dos algoritmos de conversão de gramáticas em reconhecedores, que foi publicado como um livro introdutório sobre compilação (NETO, 1987).

No trabalho realizado por Neto e Komatsu (1988) foi apresentado um aperfeiçoamento dos algoritmos de geração dos reconhecedores sintáticos baseados em autômatos de pilha estruturados. Ao se definir compiladores por meio de formalismos clássicos como autômato de estados finitos e de pilhas, defrontar-se-á com problemas comuns e freqüentemente não considerados em níveis sintáticos. O tratamento destes problemas é, geralmente, excluído da análise sintática, migrando para seções impróprias do compilador, de cunho originalmente semântico. Neto (1988) apresenta o transdutor adaptativo que teve por objetivo facilitar a representação de tais problemas, que consiste numa classe de máquinas de estados finitos com memória organizada em pilha e que exibe recursos de aprendizado baseados na alteração dinâmica de sua configuração, com base nas transições efetuadas pelo transdutor.

Um dos principais resultados obtidos no desenvolvimento deste transdutor foi um aumento no poder de representação dos modelos matemáticos, empregados por meio da inclusão de recursos de aprendizado. Os transdutores adaptativos, assim idealizados, são capazes de se moldarem a cada sentença particular a ser reconhecida, proporcionando uma forma extremamente natural para o tratamento de diversos problemas sintáticos usualmente considerados como parte da manipulação semântica da linguagem.

Os resultados de uma investigação em que se procura obter, a partir de técnicas clássicas simples, mecanismos capazes de integrar em um único método de análise os elementos necessários à resolução plena e uniforme de grande parte dos problemas de interesse na análise de linguagens, referentes a aspectos puramente sintáticos podem ser encontrados em (NETO, 1993). Neste trabalho, foi apresentado o autômato e o transdutor adaptativo como dispositivos de reconhecimento e transdução sintática.

Com base no trabalho de Neto (1993), foram realizados outros trabalhos nos quais foi aplicada a tecnologia adaptativa no projeto de sistemas reativos. Almeida (1995) apresentou uma evolução da notação de *Statechart* (HAREL et al., 1987), na qual foram acrescentadas características provenientes da teoria de Autômatos Adaptativos.

O *Statechart* Adaptativo tem capacidade de modificar sua configuração em resposta às entradas impostas ao sistema por ele representado. O trabalho realizado por Almeida (1995) permitiu também a implementação de uma ferramenta de análise e desenvolvimento de aplicações valendo-se de *Statechart* Adaptativo. Essa ferramenta, intitulada STAD (STAD, 2005), constitui-se de um editor e de um simulador de sistemas que utiliza a notação desenvolvida. Além de ter propiciado uma visão prática da teoria do uso de tecnologia adaptativa, a ferramenta STAD comprovou a sua viabilidade, proporcionando assim uma forma bastante útil de difusão desses conceitos.

Um estudo que teve por objetivo melhorar a especificação de um conjunto de sistemas reativos complexos e sincronizados entre si pode ser encontrado em (NOVAES, 1997). Um dos resultados desse trabalho foi o desenvolvimento de um formalismo, o Stad-Sinc, que se fundamenta na junção das notações de Rede de Petri, de *Statechart* convencional e de *Statechart* Adaptativo. O novo formalismo

criado permite representar por meio de diagramas os mecanismos de sincronização existentes nas aplicações projetadas.

Neto, Almeida e Novaes (1998) apresentaram uma ferramenta para o desenvolvimento e análise, intitulada STAD-S. Tal ferramenta constitui-se de um editor de *Statechart* e de um simulador de *Statecharts* sincronizados. O Stad-S permite que um sistema seja considerado e diversos pontos de vista provenientes das características dos formalismos subjacentes ao formalismo desenvolvido. Sendo assim, os aspectos de hierarquia e concorrência fundamentam-se nas características do *Statechart*; os aspectos de sincronização são baseados no mecanismo de Rede de Petri; e os aspectos de aprendizagem utilizam-se dos conceitos de automodificação provenientes dos *Statecharts* Adaptativos. Desta forma, o Stad-S permite a representação de cada um desses aspectos isoladamente, bastando omiti-los, se forem desnecessários.

Um Ambiente de Desenvolvimento de Reconhecedores Sintáticos Baseado em Autômatos Adaptativos foi descrito em (PEREIRA, 1997). Este trabalho introduziu uma ferramenta de auxílio ao desenvolvimento de reconhecedores sintáticos denominada Reconhecedor Sintático para Window (RSW) (PEREIRA, NETO, 1999). A ferramenta RSW proporciona aos seus usuários um ambiente integrado, e os reconhecedores sintáticos reconhecidos e gerados pela ferramenta são baseados na teoria de autômato de estados finitos, autômatos de pilha estruturados e nos autômatos adaptativos. A possibilidade de implementação de reconhecedores baseados em Autômatos Adaptativos (NETO, 1993) constitui uma das importantes metas alcançadas pela ferramenta RSW, comprovando sua viabilidade prática.

Um formalismo dual ao autômato, o qual visava a facilitar o desenvolvimento de linguagens complexas ou outras aplicações que necessitassem especificar linguagens dependentes de contexto na forma de gramáticas foi denominado como Gramática Adaptativa (NETO, IWAI, 1998; IWAI 2000). Tal formalismo possui como característica principal a capacidade de se alterar a medida que vai sendo feita a geração da sentença pertencente à linguagem que é representada pela gramática adaptativa. Tal trabalho foi fruto da compilação de alguns trabalhos referentes às gramáticas adaptáveis (SHUTT, 1993, 1995; RUBINSTEIN, SHUTT, 1993, 1994, 1995), bem como de alguns formalismos correlatos dinâmicos que são utilizados na representação de linguagens, como por exemplo, os autômatos.

Rocha (2000) elaborou um estudo que visa o desenvolvimento de um método de construção de modelos e resolução de problemas complexos utilizando-se dispositivos adaptativos. Para tal, foram realizados estudos comparativos entre autômatos adaptativos, redes neurais, algoritmos genéticos e agentes, extraindo destes dispositivos, características que permitiram a composição do modelo denominado Busca de Soluções por Máquina Adaptável (BSMA) (ROCHA; NETO, 2000).

Com base no método BSMA foi construído um simulador para Autômatos Adaptativos para a escolha de solução de problemas e apresentada uma proposta para o uso da tecnologia adaptativa na simulação de Redes Neurais em ambientes computacionais (ROCHA, 2001; ROCHA, NETO, 2001). Tais estudos serviram para demonstrar a possibilidade de utilização da tecnologia adaptativa em aprendizagem computacional e, de maneira geral, em inteligência artificial.

Freitas (2000) realizou um estudo referente à aplicação da tecnologia adaptativa no desenvolvimento de ambientes que suportem multilinguagens de programação. Este trabalho apresentou uma proposta de implementação de um ambiente que viabilize o emprego da programação multilinguagem por meio do oferecimento de primitivas que facilitem a interface entre os diversos segmentos de linguagens que compõem a aplicação. Tais primitivas estabelecem um mecanismo de gerenciamento de nomes relativos às diferentes variáveis que são importadas ou exportadas entre os módulos componentes da aplicação multilinguagem. Desta forma, ocorre a necessidade de um mecanismo que gerencie o espaço de nomes do ambiente. Neste contexto, Freitas e Neto (2000a) empregaram o autômato adaptativo como coletor de nomes. Embora simples, esta estrutura representa uma alternativa eficiente e elegante para representar as estruturas de dados de armazenamento e busca de cadeias. Para validar a proposta de implementação do ambiente multilinguagem, foi desenvolvido um Ambiente Multilinguagem (AML), no qual as linguagens *C++*, *Prolog*, *Lisp* e *Java* podem ser utilizadas concomitantemente (FREITAS; NETO, 2000b, 2001).

Em relação ao projeto de linguagens, Neto e Silva (2005) apresentaram uma estrutura adaptativa para design de linguagens de especificação de software. Neste trabalho foram apresentadas algumas características adaptativas presentes em linguagens de especificação, e também foi descrita uma estratégia para estender a especificação de linguagens de programação. Um exemplo simples também foi

apresentado para demonstrar a estrutura proposta. Neto (2001) busca características comuns presentes nos dispositivos adaptativos dirigidos por regras, o que permite a um especialista estender um dispositivo dirigido por regras para suportar tecnologia adaptativa. Com base nisso Camolesi e Neto (2003) apresentaram uma proposta de extensão do dispositivo *Interaction System Design Language (ISDL)* (QUARTEL, 1997) definindo então o dispositivo $ISDL_{Adp}$. O estudo também demonstra a aplicação da tecnologia adaptativa na modelagem de aplicações hipermídia. Em estudo posterior (CAMOLESI; NETO, 2004a), foi apresentada uma formulação completa do $ISDL_{Adp}$ e a aplicação de sua linguagem na modelagem de aplicações complexas

No trabalho desenvolvido por Camolesi e Neto (2004b) foi descrita a extensão de Rede de Petri Adaptativa (RP_{Adp}) conforme a formulação de (NETO, 2001). Neste estudo, propunha-se a definição de uma estrutura de dados para a representação da Rede de Petri Adaptativa. As etapas de extensão de um dispositivo adaptativo e uma estrutura de dados geral para a representação de dispositivos adaptativos dirigidos por regras pode ser encontrado em (CAMOLESI, 2005).

Pistori e Neto (2002) apresentaram o AdapTree - um algoritmo para indução de árvores de decisão que usa técnicas adaptativas - e os primeiros resultados da aplicação de técnicas adaptativas na produção de algoritmos de aprendizagem eficientes. Um protótipo de um sistema cuja interface com o usuário era feita pela direção do olhar utiliza técnicas de baixo custo, fundamentadas em aprendizado computacional e que usa tecnologia adaptativa pode ser encontrado em (PISTORI et al., 2003).

Pistori (2003) apresentou o Autômato de Estados Finitos Adaptativo e descreveu uma complementação para a representação de funções e ações adaptativas, além de uma integração de dispositivos adaptativos, basicamente discretos, com mecanismos que manipulam informações não discretas. Neste trabalho, também foram desenvolvidos alguns exemplos que demonstram a utilização de tecnologia adaptativa no desenvolvimento de aplicações. Um outro estudo, que fundiu conceitos de Autômato de Estados Finitos Adaptativo e de algoritmos genéticos, criando um ambiente propício para explorar o impacto de adaptação individual durante toda a vida em evoluções de populações foi apresentado em (PISTORI et al., 2005).

Um estudo focalizando a aplicação de tecnologia adaptativa na otimização de código em compiladores foi apresentado em (LUZ; NETO, 2003). Luz (2004) introduziu o uso de uma ação adaptativa de forma que o algoritmo de otimização automodificasse o seu comportamento em resposta a uma condição de entrada específica e procurasse a seqüência de regras de otimização que melhor se aplicasse ao código-objeto entre as muitas seqüências possíveis resultantes da superposição de duas ou mais regras de otimização igualmente aplicáveis.

A utilização de autômatos adaptativos para o mapeamento de movimentos de robôs móveis autônomos pode ser apreciado em (SOUSA; HIRAKAWA; NETO, 2004a e 2004b). Inicialmente, o robô possuía um pequeno mapa do ambiente que era ampliado por meio dos caminhos percorridos por ele mesmo. Para tal, foi construído um algoritmo que utilizava técnicas adaptativas que inicialmente adicionavam algumas marcas livres que eram modificadas com informações obtidas pelos sensores. Sousa e Hirakawa (2005) demonstram o funcionamento da navegação do robô utilizando esta estrutura.

2.1.1. Formalização geral para dispositivos não-adaptativos.

Um dispositivo não-adaptativo, dirigido por regras, pode vir a ser qualquer máquina formal cujo comportamento dependa exclusivamente de um conjunto finito de regras, que determinem, para cada possível configuração corrente do dispositivo, a sua próxima configuração. O dispositivo é dito determinístico se, e somente se, dada uma configuração e qualquer cadeia de entrada, o mecanismo definir univocamente sua próxima configuração. Caso contrário, o mecanismo é dito não-determinístico. Os mecanismos não-determinísticos permitem mais de uma configuração seguinte, ou seja, admitem mais de um possível movimento em cada caso. Assim, o uso desses mecanismos requer que todas as possíveis configurações sejam tentadas como passos intermediários para alguma configuração final, na qual a cadeia de entrada será aceita. Ineficiências causadas por tal operação de tentativa e erro, normalmente tornam inadequada a utilização prática dos dispositivos não-determinísticos. Para alcançar eficiência, recomenda-se, sempre que possível, a utilização de mecanismos determinísticos equivalentes.

Um dispositivo não-adaptativo dirigido por regras é definido por ND = (C, NR, S, c_0, A, NA). Nessa formulação, C é o conjunto de todas as possíveis configurações e $c_0 \in C$ é a sua configuração inicial. S é o conjunto de todos os possíveis eventos que compõem a cadeia de entrada ND, o conjunto A representa as configurações de aceitação para ND e F = C – A corresponde ao conjunto de configurações que são rejeitadas. NR é o conjunto das regras, que definem ND, por meio de uma relação $NR \subseteq C \times S \times C \times NA$. As regras $r \in NR$ têm a forma $r = (c_i, s, c_j, z)$, significando que, em resposta a qualquer evento de entrada $s \in S$, a regra r modifica a corrente configuração de c_i para c_j, consome s e produz $z \in NA$ como saída.

Na prática, os símbolos de saída contidos em NA podem ser obtidos por meio de chamadas de procedimento. Assim, uma saída gerada por uma aplicação de qualquer regra pode ser interpretada como resultado de uma chamada a seu procedimento correspondente.

Uma regra $r = (c_i, s, c_j, z)$, com $r \in NR$; $c_i, c_j \in C$; $s \in S$; $z \in NA$, é dita compatível com a configuração corrente c se, e somente se, c_i=c e s ou é o evento vazio, ou é igual ao evento corrente. Neste caso, a aplicação de uma única regra compatível leva o dispositivo à configuração c_j (esse movimento é denotado por $c_i \Rightarrow sc_j$) e produz z como cadeia de saída. Observar que s, z ou ambos podem ser vazios, logo $c_i \Rightarrow^{\sim} c_m$, $m \geq 0$ abrevia $c_i \Rightarrow^{\varepsilon} c_1 \Rightarrow^{\varepsilon} c_2 \Rightarrow^{\varepsilon} \ldots \Rightarrow^{\varepsilon} c_m$, uma sucessão opcional de movimentos vazios. Se $c_i \Rightarrow^{\sim\kappa} c_j$, denota-se $c_i \Rightarrow^{\sim} c_m \Rightarrow^{\sim w_k} c_j$, uma sucessão opcional de movimentos vazios seguida por um símbolo não-vazio consumindo o símbolo $_k$. Uma cadeia de entrada $w = w_1 w_2 \ldots w_n$ é *aceita* por ND quando $c_0 \Rightarrow^{\sim w_1} c_1 \Rightarrow^{\sim w_2} c_2 \Rightarrow^{\sim w_n} c_n$ (por exemplo, $c_0 \Rightarrow^{w} c$, com $c \in A$). Finalizando, w é dito rejeitado por ND quando $c \in F$. A linguagem descrita por ND é $L(ND) = \{w \in S^* \mid c_0 \Rightarrow^{w} c, c \in A\}$ para toda cadeia de entrada $w \in S^*$ que é aceita por ND.

2.1.2. Formalização geral para dispositivos adaptativos

Seja T um contador automático de tempo cujo valor é iniciado em 0 e que é incrementado automaticamente sempre que uma ação adaptativa não-vazia é executada. Cada valor k assumido por T pode subscrever o nome de uma função

adaptativa no conjunto de tempo. Neste caso, são associadas operações adaptativas AD's ao passo k.

Um dispositivo AD=(ND_0, AM) é dito adaptativo sempre que para todo passo $k \geq 0$ o mecanismo adaptativo levar o comportamento de ND_k até a execução de alguma ação adaptativa não-vazia do passo k+1 e iniciar uma operação que altere o conjunto de regras.

Qualquer operação AD no passo $k \geq 0$ inicia o correspondente dispositivo ND_k subjacente para NR_k. A execução de alguma ação adaptativa não-vazia evolui ND_k para ND_{k+1}. Assim, AD inicia sua operação no passo k+1 criando o conjunto NR_{k+1} como uma versão editada NR_k. Depois disso, AD seguirá o comportamento de ND_{k+1} até uma ação adaptativa não-vazia iniciar outro passo. Este procedimento se repetirá até que a cadeia de entrada seja totalmente processada.

AD inicia sua operação na configuração c_0, em sua forma inicial AD_0= (C_0, AR_0, S, c_0, A, NA, BA, AA). No passo $k \geq 0$, um estímulo de entrada movimenta AD para uma próxima configuração e prossegue então sua operação no passo k+1 se, e somente se, uma ação adaptativa não-vazia é executada. Assim, estando AD em seu passo k, na forma AD_k=(C_k, AR_k, S, c_k, A, NA, BA, AA), a execução de uma ação adaptativa não-vazia o levará à forma AD_{k+1}=(C_{k+1}, AR_{k+1}, S, c_{k+1}, A, NA, BA, AA).

Nesta formulação, AD=(ND_0, AM) é algum dispositivo adaptativo formado por um dispositivo inicial subjacente ND_0 e um mecanismo adaptativo AM. ND_k é o dispositivo subjacente no passo k. ND_0 é o dispositivo subjacente, definido pelo conjunto inicial NR_0 de regras não-adaptativas. Por definição, qualquer regra não-adaptativa em NR_k tem sua regra correspondente adaptativa em NR_k. C_k é o conjunto de todas as possíveis configurações de ND no passo k, e $c_k \in C_k$ é a sua configuração inicial no passo k. Para k=0, tem-se, respectivamente, C_0, o conjunto inicial de configurações válidas e $c_0 \in C_0$, a configuração inicial de ND_0 e ND. A cadeia vazia ε denota a ausência de qualquer outro elemento válido do conjunto correspondente. S é o conjunto (finito, fixo) de todos os possíveis eventos (inclusive o evento vazio ε) de que se compõe a cadeia de entrada AD. $A \subseteq C$ é o subconjunto de configurações de aceitação. F = C - A é o conjunto das configurações de rejeição. BA e AA são conjuntos de ações adaptativas. Ambas incluem a ação vazia ($\varepsilon \in BA \cap AA$) e $w = w_1 w_2 ... w_n$, é a cadeia de entrada onde $w_k \in S - \{\varepsilon\}$, k =1,..., n com $n \geq 0$;

NA, com $\varepsilon \in$ NA, é um conjunto (finito, fixo) de todos os símbolos que podem ser gerados como saídas por AD, em resposta à aplicação de regras adaptativas. Analogamente ao que ocorre com os dispositivos não-adaptativos, a cadeia de saída assim obtida pode ser interpretada como uma sucessão de chamadas dos procedimentos correspondentes. AR_k é o conjunto das regras adaptativas que definem o comportamento adaptativo de AD no passo K e é dado por uma relação $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$.

Em particular, AR_0 define o comportamento inicial de AD. Ações adaptativas modificam o conjunto corrente de regras adaptativas AR_k de AD para um novo conjunto AR_{k+1}, adicionando e/ou eliminando regras adaptativas em AR_k. Regras $ar \in AR_k$ são da forma (ba, c_i, s, c_j, z, aa), significando que, em resposta a algum símbolo da cadeia de entrada s $\in$ S, ar inicialmente executa a ação adaptativa $ba \in BA$. Se a execução de ba eliminar ar de AR_k, a execução de ar é abortada; caso contrário, aplica-se a regra subjacente não-adaptativa nr = (c_i, s, c_j, z) $\in$ NR_k, conforme descrito anteriormente; e finalmente, executa-se a ação adaptativa aa $\in$ AA. Define-se AR como o conjunto de todas as possíveis regras adaptativas para AD. Define-se NR como o conjunto de todas as possíveis regras subjacentes não-adaptativas para AD. AM $\subseteq$ BA $\times$ NR $\times$ AA, definido para um dispositivo adaptativo particular AD, é um mecanismo adaptativo aplicado a todas as regras no passo k em $NR_k \subseteq$ NR, no qual AM deve ser interpretado da mesma forma como se fosse aplicada a qualquer subdomínio $NR_k \subseteq$ NR. Isso determinará um único par de ações adaptativas associadas a cada regra não-adaptativa.

O conjunto $AR_k \subseteq$ AR pode ser obtido colecionando-se todas as regras adaptativas construídas pela associação de cada par de ações adaptativas às correspondentes regras não-adaptativas em NR_k. Equivalentemente, pode-se construir NR_k removendo-se das regras AR_k todas as referências às ações adaptativas. O roteiro abaixo descreve de forma geral a operação, a partir do instante T=0, de qualquer dispositivo adaptativo dirigido por regras:

1. Leve o dispositivo adaptativo à sua configuração inicial;
2. Selecione, como evento corrente, o símbolo mais à esquerda da cadeia de entrada;

3. Caso não haja mais nenhum evento a ser processado, então desviar para o passo 8 deste roteiro, caso contrário, alimente o dispositivo fornecendo-lhe o próximo evento a ser processado;
4. Escolha a próxima regra adaptativa a ser aplicada: dada a configuração corrente $c_T \in C_T$ e um símbolo $s \in S$ extraído da parte ainda não processada da cadeia de entrada, pesquise-se o conjunto NR em busca de regras compatíveis, ou seja, determine se no conjunto de todas as regras existe uma regra que pode ser aplicada de acordo com as circunstâncias correntes $CR_T = \{ar \in AR_T \mid ar = (ba, c_T, s, c, z, aa), s \in \{s_T, \varepsilon\}; c_T \in C_T; ba \in BA; aa \in AA; z \in NA\}$

   Há três casos a considerar: se CR_T for vazio, nenhum movimento será possível para o dispositivo (por ser AR_T incompletamente especificado), então a cadeia de entrada será rejeitada, por definição; se $CR_T = \{ar_k\}$ para alguma $ar_k = (ba_k, c_T, s, c_k, z_k, aa_k) \in AR_T$, então existirá uma única regra disponível para aplicação determinística. Dessa forma, o dispositivo alcançará um estado pré-definido na próxima configuração $c_{T+1}=c$; se $CR_T = \{ar_k=(ba_k, c_T, s, c_k, z_k, aa_k) \mid k=1,2,..., m\}$, então todas estas m regras serão igualmente aplicáveis de forma não-determinística à configuração corrente, como é usual na operação de dispositivos não-determinísticos. Obviamente, em ambientes seqüenciais, o paralelismo deve ser exaustivamente simulado, por exemplo, aplicando-se algum tipo de estratégia de retrocesso à configuração anterior do dispositivo (*backtracking*).
5. Se uma ação adaptativa ba_k for a ação adaptativa ε (vazia) na regra que estiver sendo aplicada, então passar-se-á à execução do passo 6. Caso contrário, aplique-se a ação adaptativa ba_k ao conjunto de regras atuais, gerando uma nova configuração intermediária para o dispositivo AD. Note que, em alguns casos, até mesmo a regra adaptativa ar_k, eventualmente, pode ser removida em decorrência da aplicação de ba_k. Neste caso extremo, a aplicação de ar_k é abortada, retorne ao passo 4;

6. Aplique a regra $nr_k=(c, s, c_k, z_k)$, conforme definido para dispositivo não-adaptativo (subjacente), gerando uma configuração (intermediária) de AD;
7. Se a ação adaptativa executada aa_k, indicada na regra adaptativa ar_k, for a ação adaptativa vazia (ε), desvie para o passo 3; caso contrário, aplica-se aa_k ao conjunto corrente de regras adaptativas, gerando, finalmente, uma nova configuração do dispositivo. Da mesma forma que no passo 5, a execução da ação adaptativa também pode apagar ar_k. Neste caso, porém, não acarreta qualquer ação adicional;
8. Se a configuração corrente for uma configuração de aceitação, então a cadeia de entrada será considerada aceita, caso contrário será rejeitada. Em qualquer caso, a operação do dispositivo será finalizada.

2.2. Metadados

O texto apresentado nesta seção está fortemente fundamentado nos conceitos apresentados em Duval (2001) que define Metadados como um princípio fundamental de organização dos dados para ambientes caracterizados por fontes diversas de conteúdos, estilos diversos de gerenciamento de conteúdos e técnicas para descrição de recursos. Permite aos projetistas de esquemas de dados desenvolverem novas estruturas, fundamentados em metaestruturas previamente definidas e beneficiarem-se do reaproveitamento das estruturas existentes sem ter que redefini-las.

No mundo de metadados, dados elementares de diferentes esquemas, bem como seus significados e outros blocos de construção, podem ser combinados de forma sintática e semântica. Duval (2001) define que os projetistas de aplicações podem se beneficiar destes conceitos e desenvolver estruturas que permitam seu reuso, da mesma forma que ocorre no brinquedo *Lego* (LEGO, 2007; VON ATZINGEN, 2001) cujo conceito se baseia em partes que se encaixam permitindo inúmeras combinações. Usando metadados, o projetista pode juntar as partes de estruturas definidas e definir diferentes estruturas a partir de uma estrutura básica pré-existente, semelhante ao que ocorre no *Lego*, no qual a partir de diferentes estruturas podem ser engenhadas diferentes construções, que até são

semanticamente diferentes, mas preservam as mesmas estruturas dos blocos definidos para a arquitetura *Lego* (DUVAL et al., 2002).

Crianças, ao definirem seus brinquedos, usando *Lego*, não se preocupam em relação ao tema que estão criando. Podem desenvolver um prédio, algo que se assemelha a um estádio de futebol, ou mesmo um barco pirata. A semântica de tais combinações gera coisas distintas, que podem nem sempre serem óbvias para os adultos, mas o são para as crianças. Similar flexibilidade deveria ser observável em estruturas para metadados, permitindo aos projetistas de aplicações misturarem uma variedade de módulos semânticos comuns a uma mesma definição e obterem aplicações semanticamente diferentes. Por exemplo, um módulo de metadados de pesquisa XML e um módulo para gerenciamento instrucional – por exemplo, cursos a distância - escrito também em XML, possuem as mesmas características da linguagem sintática de definição XML e, desta forma, poderiam ser combinados formando um único esquema de definição que abrigasse ambas as definições.

Santos (2003) cita que a grande maioria dos usuários da Tecnologia da Informação já utilizou algum tipo de metadados, mesmo não tendo conhecimento de seu significado e nem mesmo de seu uso. Isso é absolutamente normal, até porque a sua própria definição não é um consenso.

Na literatura sobre metadados, a definição mais encontrada é a de que eles representam “dados sobre dados”. Barreto (1999) dá como exemplo, no contexto de banco de dados relacional, o modelo de dados básico, que consiste em tabelas, registros e atributos. No contexto da orientação a objetos, o modelo de dados básico consiste em classes e objetos. Em ambos os modelos, regras de composição distintas se aplicam. No modelo orientado a objetos, essas regras se referem à herança entre classes, polimorfismo, domínio de atributos e agregação; enquanto no modelo relacional essas regras especificam, por exemplo, que um registro deve conter um número qualquer de atributos e que cada atributo deve ser definido como termo de um tipo primitivo de dado.

De forma genérica, a complexidade do modelo de dados depende diretamente da complexidade estrutural da informação que se quer representar. Um modelo que consista num conjunto de pares (nome-atributo, valor-atributo) não possibilitará a representação dos tipos de relacionamentos encontrados em um sistema de arquivos, por exemplo. Neste caso, modelos de dados mais complexos (baseados em grafos, árvores, etc.) seriam mais apropriados.

Os metadados são de extrema importância, uma vez que, além de descreverem o conteúdo de determinado conjunto de dados, permitem uma redução do tamanho dos mesmos.

O fato de se utilizar informação padronizada faz com que os programas passem a ter sua utilização facilitada e os projetistas de aplicações possam intercambiar facilmente dados entre diferentes sistemas e plataformas computacionais. Ao se criar metadados, define-se uma padronização de nomenclatura, de definições, de catalogação e de operações para o tipo de informação que se esteja projetando.

2.2.1. Extensibilidade e refinamento de metadados

Sistemas de metadados têm que permitir a extensibilidade de estruturas, de forma que as necessidades particulares de uma determinada aplicação possam ser moldadas de acordo com seus requisitos. Arquiteturas de metadados devem possuir uma estrutura básica e receber facilmente elementos adicionais que sejam necessários a uma aplicação, de acordo com as necessidades específicas relacionadas ao domínio da aplicação em fase de projeto. Tal estrutura deve permitir que sejam adicionados novos componentes sem que ocorram problemas relacionados à interoperabilidade do esquema básico. Outra aplicação que encontre os novos esquemas de metadados definidos deverá ignorá-los caso estes não sejam inerentes a ela própria, e fazer uso dos elementos básicos previamente definidos ou utilizar outros recursos que tenham sido adicionados e que façam parte dela.

Domínios de aplicação devem ser definidos de acordo com o grau de detalhe necessário ou desejável. O design padrão de metadados deve permitir aos projetistas de esquemas escolherem o nível de detalhe apropriado para uma determinada aplicação. Determinadas áreas ou até mesmo termos no campo da informática podem ter interpretações diferentes, de acordo com o contexto em que estiverem inseridas. Neste contexto é que se fundamenta nesta seção o termo “semântica” (SHETH; RAMAKRISHNAN; THOMAS, 2005). Por exemplo, no domínio de bancos de dados, metadados é pensado como um esquema conceitual que descreve sua estrutura; já no domínio de recuperação de informação, poderíamos

considerar metadados como um conjunto de palavras-chave que descrevem o tema principal de um documento, ou como um registro para um esquema específico.

Sheth, RAMAKRISHNAN e Thomas (2005) descrevem estas diferentes representações de metadados, organizando-as em três tipos de semântica: implícita, formal e forte. A **semântica implícita** se parece com um texto não estruturado que define livremente sua estrutura e é menos formal (por exemplo, Recuperação de informação). A **semântica formal** é uma forma de representação dos dados de um modo mais rígido (por exemplo, Representação de Conhecimento). Finalmente, a **semântica forte** traz a combinação de estruturas sintáticas simples para representar o significado de estruturas mais complexas.

Al-Khalifa e Davis (2006) nos mostra que é importante manter a sintaxe e a semântica separadas até quando for possível. As mudanças rápidas da primeira década da Web ilustram bem isto. Várias versões de HTML foram definidas, o aparecimento de XML e o desenvolvimento de tecnologias derivadas destas duas, que incluem, atualmente, a definição de esquemas e os conceitos de *Resources Description Framework* – RDF – (RDF, 2007). A falta de estabilidade relacionada às definições das linguagens de marcação enfatiza a necessidade de manter independência entre a semântica de elementos de metadados e suas representações sintáticas. Porém, como a cada dia mais informações estão surgindo, espera-se que os metadados sejam essenciais na criação e na administração de recursos. Assim, não podem ser ignorados os assuntos relacionados à sintaxe, embora estejam mais próximos das relações semânticas de metadados e, particularmente, na forma como eles são relacionados.

2.2.2. Propriedades de metadados

Há uma espécie de quase absoluto consenso por parte de projetistas de metadados no sentido de utilizar todos os metadados definidos, ou seja, “preencher todos os espaços em branco”. Se um elemento estiver disponível, as pessoas querem/devem usá-lo em uma descrição. As aplicações poderiam ser projetadas para tornar evidente que nem todo elemento disponível é necessariamente apropriado para todo tipo de recurso. De forma semelhante, as aplicações deveriam prover ajuda, onde fosse possível, para a seleção de um valor apropriado para um

elemento particular. Para que as extensões de metadados criem facilidades na criação de aplicações que os utilizam como base, a aplicação poderia identificar os valores para alguns elementos, o que seria algo de maior confiabilidade em relação aos que são atribuídos pelo o usuário (DUVAL et al., 2002).

No final das contas, a riqueza de descrições de metadados será determinada pela experiência e pelo trabalho desenvolvido pelo projetista destas estruturas, guiados pelas exigências funcionais dos serviços ou aplicações. Metadados são amplamente definidos como dados estruturados sobre dados. Porém, o processo de criar metadados pode envolver contribuições subjetivas e objetivas. Alguns metadados são claramente objetivos: afirmação sobre autoria, data de criação, versão, e outros atributos, geralmente, podem ser determinados de um modo objetivo. Tais metadados podem ser gerados por máquinas, na maioria dos exemplos, como metadados de "propriedades" gerados a partir do momento em que se cria um arquivo em um processador de textos ou uma aplicação de planilha eletrônica.

Outros metadados podem ser subjetivos, pois tais elementos podem sujeitar-se a pontos de vista discrepantes (adjudicação de palavras-chave, sumarização de conteúdos abstratos), ou porque especificamente é pretendido que eles representem uma avaliação subjetiva (uma revisão de um livro ou uma apresentação). Até mesmo elementos de metadados mais formais ficam subjetivos quando usados dentro de um contexto de domínio que está sujeito a interpretações locais. Por exemplo, uma característica pedagógica que seja dependente de uma filosofia educacional particular pode ser importante num determinado contexto, mas não terá nenhum significado fora deste. A exigência para desenvolver metadados é, até onde possível, fazer com que contextos explícitos de aplicações possam reconhecer mais facilmente quando um determinado elemento revela-se restrito a tal contexto, o que evitaria ser erroneamente aplicável com amplitude.

3. Proposta de um gerador de ambientes para a modelagem de aplicações adaptativas

Para que um especialista em um determinado dispositivo não-adaptativo (subjacente) possa adicionar as funcionalidades providas por mecanismos adaptativos ao mesmo, devem ser realizadas algumas etapas. Inicialmente, o especialista necessita adquirir conhecimentos em tecnologia adaptativa — sua estrutura e funcionamento. Em seguida, o especialista acrescenta ao dispositivo subjacente a camada adaptativa e, por fim, disponibiliza o dispositivo obtido para o desenvolvimento de aplicações.

Neste capítulo é apresentado um método para definição de dispositivos adaptativos, um ambiente para o projeto de aplicações que use os dispositivos definidos e um gerador para tais ambientes. O método proposto permite a um especialista acrescentar a um dispositivo subjacente o mecanismo adaptativo e utilizar um gerador de ambientes para obter ferramentas que suportem o projeto de aplicações usando dispositivos adaptativos.

3.1. Método para a definição de dispositivos adaptativos

A utilização de um método adequado é importante para facilitar e tornar mais produtivo o trabalho do especialista na definição de um dispositivo adaptativo. O método proposto foi fundamentado na análise dos trabalhos de definição de dispositivos adaptativos realizados (NETO, 1993; ALMEIDA, 1995; IWAI,2000; PISTORI, 2003; CAMOLESI, 2004a; CAMOLESI, 2004b) para a definição de dispositivos adaptativos e nas experiências adquiridas durante a elaboração destes trabalhos. Tal método é constituído de três etapas: etapa de definição do modelo teórico (matemático); etapa de mapeamento para o modelo lógico (representação intermediária do modelo teórico) e etapa de definição do modelo físico (programa). A Figura 2 ilustra as etapas e as relações existentes entre si.

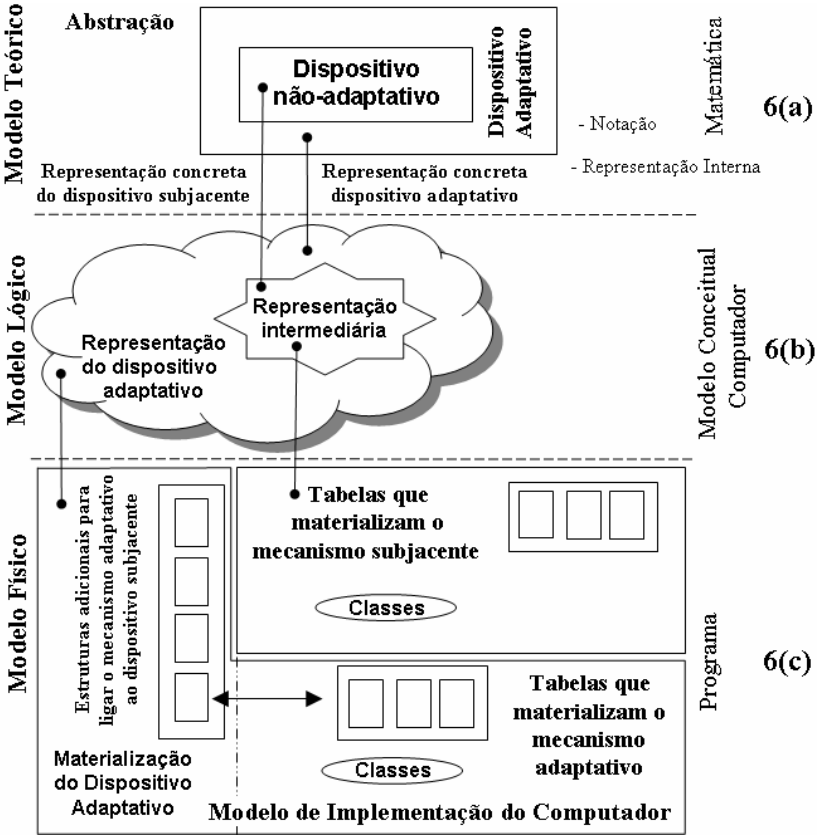


Figura 2. Método para definição de dispositivos adaptativos

Na etapa de definição do modelo teórico, representada na Figura 2(a), entende-se que um especialista com bons conhecimentos matemáticos sobre determinado dispositivo não-adaptativo possa acrescentar à definição formal do dispositivo subjacente as funcionalidades providas pelos mecanismos adaptativos. Em Neto (1993) e em Almeida (1995) são apresentadas extensões de dispositivos subjacentes pela inclusão de conceitos de dispositivos adaptativos. Nesta fase é realizada a união dos conceitos formais de ambos os dispositivos (não-adaptativo e adaptativo), pela qual se obtém o novo dispositivo adaptativo.

A etapa de definição do modelo teórico é a primeira etapa a ser realizada ao se definirem dispositivos adaptativos. Tal etapa fundamenta-se na estrutura para representação geral de dispositivos adaptativos proposta em (NETO, 2001). O trabalho aí desenvolvido, sintetizado também no Capítulo 2 deste trabalho, pode ser considerado como a forma mais abrangente para definição de um dispositivo adaptativo. Sua estrutura fundamenta-se na existência de um dispositivo não-adaptativo — por exemplo, Autômato de Estados Finitos, Autômato de Pilha, Gramática, Rede de Petri, ISDL, etc., — o qual é envolvido por um mecanismo adaptativo, responsável por permitir que a estrutura do mecanismo subjacente seja

dinamicamente modificada. Um autômato de estados finitos, por exemplo, ao ser envolvido por um mecanismo adaptativo, passa a poder sofrer em seu comportamento remoções ou inserções de transições enquanto realiza o processamento da cadeia de entrada, o que concede um aumento no seu poder de expressão. Uma característica fundamental no uso dos conceitos de tecnologia adaptativa é a possibilidade de utilização de dispositivos existentes com um aumento do seu poder de representação ao custo de um pequeno acréscimo na sua complexidade formal.

Após a obtenção do dispositivo adaptativo, faz-se necessário seu mapeamento para o modelo lógico, visando a representação de dispositivos adaptativos dirigido por regras no formato do modelo lógico, ilustrado pela Figura 2(b). Tal etapa consiste na definição da estrutura de dados (lógica) que representa os conceitos formais do dispositivo adaptativo. Esta é de fundamental importância, pois serve como base para a estrutura de armazenamento de informações necessárias para o desenvolvimento de ferramentas, propostas nas próximas seções deste capítulo.

Na etapa de definição física, representada pela Figura 2(c), um projetista com conhecimentos sobre o dispositivo adaptativo definido pode realizar a especificação de sua aplicação. Nessa etapa, são produzidas especificações de aplicações que serão, posteriormente, analisadas e implementadas. Esse trabalho deve ser realizado com o auxílio de um ambiente para modelagem de aplicações que use tecnologia adaptativa.

Ao realizar o seu trabalho, o projetista de aplicações instancia os objetos definidos na etapa lógica e define elementos físicos que representam o comportamento da aplicação desejada. Nesta fase, pode-se observar que são instanciados objetos pertencentes a duas classes distintas: os que representam o comportamento da aplicação projetada e os que representam as funções e ações adaptativas responsáveis por modificações no comportamento da aplicação em tempo de execução. Com base no conjunto de objetos definidos nesta fase, são permitidas a apresentação da especificação, a simulação do comportamento, a verificação de inconsistências e a execução da aplicação projetada. Durante o processo de simulação de comportamento e execução da especificação no núcleo adaptativo, ações adaptativas podem ser executadas e regras podem ser adicionadas e/ou removidas do comportamento, modificando-se sua estrutura.

Para finalizar, serão estabelecidos os seguintes passos:

- Adição do formalismo da camada adaptativa ao formalismo do dispositivo subjacente;
- Mapeamento do formalismo do dispositivo adaptativo para a representação do modelo lógico;
- Implementação de um tradutor para transformação da especificação de uma aplicação denotada na linguagem de especificação do dispositivo adaptativo em outra equivalente a representação do modelo lógico;
- Implementação de um tradutor para a transformação da representação de uma especificação no formato do modelo lógico em uma denotada no formato da linguagem de especificação do dispositivo adaptativo;
- Desenvolvimento de um ambiente que utilize o dispositivo adaptativo para o projeto de aplicações ou alteração do código-fonte de um ambiente já existente para o dispositivo subjacente suportar a camada adaptativa;

Os passos assim definidos consideram a inexistência de um gerador de ambientes. Será possível observar, mais tarde, que o gerador terá possibilitado uma implementação rápida e, uma vez existindo, poderiam ser suprimidas as últimas três fases definidas anteriormente e realizar-se apenas a fase de geração do ambiente desejado.

3.2. Proposta do Ambiente para o Projeto de Aplicações usando Tecnologia Adaptativa

O projeto de aplicações exige, em geral, soluções relativamente complexas, nas quais é imprescindível o uso de um método bem definido, que dê suporte à especificação de aplicações adaptativas e, posteriormente, à análise e à implementação das especificações produzidas. Fundamentado nos conceitos apresentados por Souza et al. (1997), é proposto um método para o projeto de aplicações adaptativas constituído de três fases: especificação, análise e implementação.

Na fase de especificação, um projetista utiliza uma linguagem de especificação e gera modelos da aplicação desejada. Com base nos modelos

obtidos, é realizada a fase de análise, que tem por objetivo verificar inconsistências na aplicação projetada. Nessa fase, o projetista submete estímulos ao modelo e obtém respostas que lhe permitem validar o modelo definido. Também nessa fase podem ser realizadas simulações do modelo, cujas respostas geradas auxiliarão o projetista na validação e análise da aplicação projetada. Ao serem detectadas inconsistências ou falhas, a especificação deve ser editada e, na seqüência, novamente analisada. Finalmente, depois de validada a aplicação, o modelo gerado poderá ser traduzido para uma linguagem de representação física. Para auxiliar o método proposto para o projeto de aplicações que use tecnologia adaptativa, faz-se necessária a utilização de um ambiente que integra um conjunto de ferramentas que dê suporte aos projetistas na realização de seu trabalho. A Figura 3 ilustra a arquitetura geral de tal ambiente, que é organizado em 3 (três) grupos de ferramentas que foram aglutinadas e definidas conforme as características inerentes ao método proposto: Ferramentas de Edição, Ferramentas de Análise e Ferramenta de Implementação.

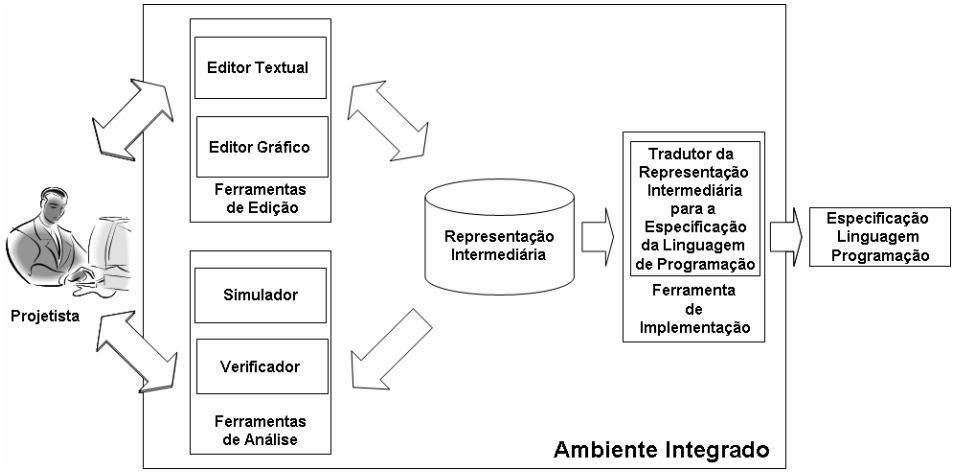


Figura 3. Ambiente para projeto de aplicações usando Tecnologia Adaptativa

Valendo-se das Ferramentas de Edição, um projetista de aplicações utilizando-se de um dispositivo adaptativo específico poderá realizar a especificação de sua aplicação com o auxilio de um editor de texto qualquer ou de um editor gráfico. Para possibilitar o intercâmbio das especificações produzidas, os editores devem gerar objetos no Modelo Lógico. Caso a especificação seja produzida em um editor textual, esta deverá ser compilada para transformar a codificação realizada no formato definido para o modelo lógico.

Depois de realizada a especificação, o projetista de aplicações poderá utilizar as Ferramentas de Análise. Tais ferramentas utilizam a codificação da especificação

(no formato do modelo lógico) como base e permitem a realização de uma análise do comportamento da aplicação adaptativa em desenvolvimento. Por fim, depois de especificada e analisada a representação de uma aplicação, o projetista pode se utilizar das Ferramentas de Produção de Representações Físicas e gerar uma representação de uma aplicação em um determinado padrão de linguagem de representação física e obter a aplicação desejada.

3.2.1. Arquitetura geral do Ambiente para o Projeto de Aplicações Adaptativas

Utilizando-se do modelo de entidades ISDL[1], é apresentada a arquitetura geral para a construção de um ambiente para o projeto de aplicações adaptativas. A Figura 4a, ilustrada na próxima página, descreve uma visão externa da arquitetura que é concebida pela estruturação de quatro entidades: *Projetista, Programa, Modelo Lógico e Ambiente para o Projeto de Aplicações Adaptativas* (*APAA*).

Na Figura 4b é apresentada a visão interna da arquitetura proposta. A entidade *Projetista* representa um projetista de aplicações que utiliza uma dada linguagem de especificação e um conjunto de ferramentas que dão suporte a esta linguagem para realizar o projeto de suas aplicações. A entidade *Programa* descreve a representação física de uma aplicação projetada no referido ambiente e traduzida para a especificação de uma linguagem de programação existente (eg. C/C++, Java, etc.). A entidade *APAA* é um módulo computacional que tem por objetivo gerenciar e dar suporte às funcionalidades necessárias ao projetista na realização de seu trabalho conforme foi definido no método adotado para o projeto de aplicações que usem tecnologia adaptativa. A entidade *APAA* tem a função de prover funcionalidades para a especificação de aplicações por meio da subentidade *Ferramentas de Edição (FE)*. As funcionalidades de simulação e verificação são representadas pela subentidade *Ferramentas de Análise (FA)* e, por fim, depois de validadas, as especificações podem ser traduzidas para uma linguagem de representação física, representada nesta arquitetura pela subentidade *Ferramenta para a Produção de Representação Física*.

[1] Para maiores informações sobre ISDL ver seção 5.3 deste trabalho.

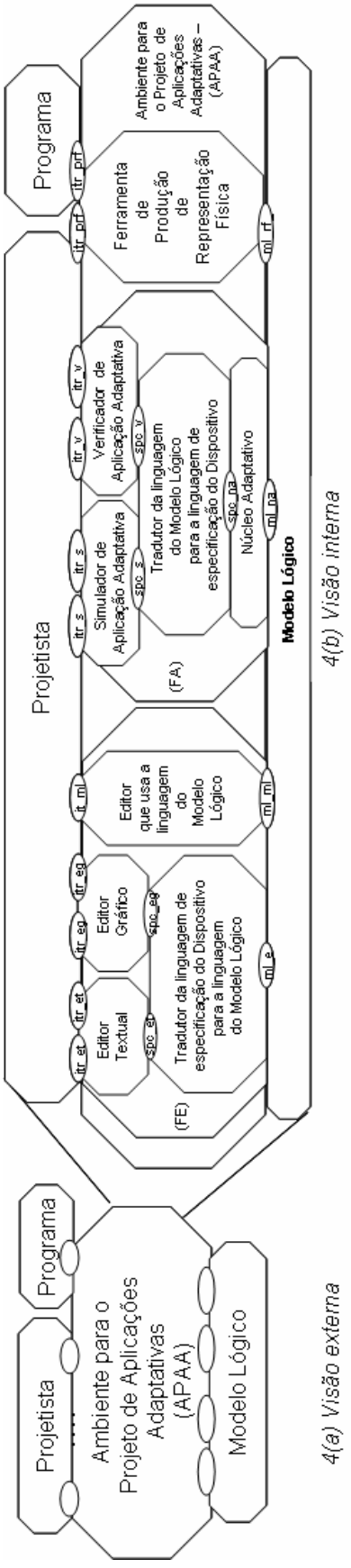


Figura 4. Modelo de entidades do Ambiente para o Projeto de Aplicações Adaptativas

A base da arquitetura proposta do Ambiente para o Projeto de Aplicações Adaptativas é o Modelo Lógico representado na Figura 4b pela subentidade *Modelo Lógico*. Tal modelo deve permitir a descrição dos dispositivos definidos, a representação das especificações das aplicações projetadas e servir para a integração das ferramentas que constituam o ambiente proposto.

3.2.2. Ferramentas de Edição

As Ferramentas de Edição são representadas pelo Editor Textual, pelo Editor Gráfico e pelo Editor que usa a Linguagem do Modelo Lógico. Tais ferramentas devem ser de fácil utilização e também devem permitir ao projetista definir os objetos necessários à especificação de uma aplicação. A estrutura dessas ferramentas deve possuir uma interface que possibilite a um projetista carregar ou definir uma nova aplicação em uma janela textual ou gráfica. Na Figura 5 é apresentada a subentidade *Ferramentas de Edição (FE)* que integra a arquitetura do APAA e é responsável por representar as ferramentas que realizam a definição de um dispositivo e a especificação de uma aplicação. Tal subentidade é refinada em quatro subentidades: *Editor Textual*, *Editor Gráfico*, *Editor que usa a Linguagem do Modelo Lógico* e *Tradutor da linguagem de especificação do Dispositivo para a linguagem do Modelo Lógico*.

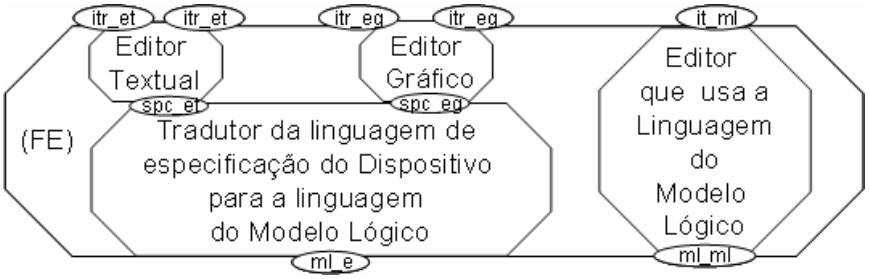


Figura 5. Representação da subentidade Ferramenta de Edição

O Editor Textual deve disponibilizar uma janela para o projetista realizar a especificação de sua aplicação com base na linguagem textual de um dispositivo específico. Tal ferramenta deve possuir um menu de opções que permita a escolha de funções como copiar/colar parte de uma especificação, buscar determinadas palavras e outras funcionalidades disponíveis nos editores de textos convencionais. Tal ferramenta deve também gerar a gravação da especificação no formato da

linguagem definida para o modelo lógico que é utilizada como base para as ferramentas do APAA.

A Figura 6 ilustra o refinamento da subentidade *Editor Textual* que é estruturada por quatro subentidades e seus pontos de interação. A subentidade *Interface do Editor Textual* é responsável por receber as interações dos projetistas de aplicações por meio do ponto de interação *itr_te* e disponibilizar-lhes os resultados no ponto *itr_ts*. Ao ser executada, a subentidade *Interface do Editor Textual* deve prover os recursos de menus, janela de especificação e barras de ferramentas, comunicar-se com o *Sistema de Ajuda do Editor Textual* por meio do ponto de interação *ig_sa* e fornecer a especificação realizada pelo projetista no ponto de interação *it_dec* para a subentidade *Representação de Elementos Conceituais de um Dispositivo*. Tal subentidade é responsável por informar à subentidade *Interface do Editor Textual* a forma de visualização textual dos elementos conceituais de um dispositivo e comunicar-se com o tradutor de especificação no formato da linguagem do dispositivo para o formato do modelo lógico por meio do ponto de interação *spc_et* para informar a especificação que deverá ser processada e armazenada no formato do Modelo Lógico.

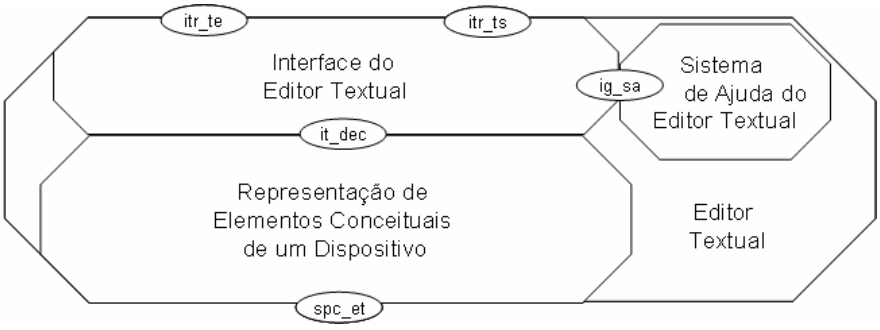


Figura 6. Refinamento da subentidade Editor Textual.

O editor gráfico deverá possuir um menu de opções e/ou caixa de ferramentas que permita ao projetista selecionar os objetos gráficos desejados correspondentes aos elementos conceituais do dispositivo que esteja em uso e possibilitar a sua inserção ou edição em uma janela gráfica. Por meio dos objetos inseridos e das interligações entre estes - conexões – obtém-se a especificação de uma aplicação. Ao ser selecionada a funcionalidade de gravação, a especificação da aplicação deve ser armazenada no formato do modelo lógico definido.

Alguns estudos referentes à concepção das ferramentas de edição gráfica existentes e de *frameworks* foram realizados para auxiliar no trabalho de

implementação de tais ferramentas. Entre as principais ferramentas e arquiteturas estudadas pode-se destacar as ferramentas produzidas no projeto *Design de Aplicações Multimídia (DAMD)* (SOUZA et al., 1997), o *framework Graphics Editing Framework (GEF)* (GEF, 2007) e a ferramenta *ArgoUML*. (ARGOUML, 2007).

A ferramenta *ArgoUML* tem por finalidade auxiliar os projetistas que utilizam a linguagem *UML* no projeto de suas aplicações. Tal ferramenta utilizou-se da linguagem Java (JAVA, 2007) para a sua implementação e foi concebida seguindo os conceitos de orientação a objetos. *ArgoUML* possui uma estrutura que permite aos desenvolvedores de aplicações o seu reaproveitamento. Dessa forma, não é necessário o desenvolvimento da estrutura geral da ferramenta: somente serão desenvolvidos os componentes necessários à implementação dos conceitos do dispositivo que estiver sendo utilizado. A ferramenta *ArgoUML* foi concebida com base no *framework GEF*. Este *framework*, orientado a objetos, foi desenvolvido na Universidade de Irvane (E.U.A.) e faz parte do núcleo da ferramenta *ArgoUML. GEF* é responsável pela definição dos elementos gráficos da caixa de ferramentas e dos menus, pelo gerenciamento da janela de edição de aplicações e pelo armazenamento e recuperação das aplicações produzidas. A sua arquitetura suporta a construção de aplicações de edição gráfica, por exemplo, aplicações que incluem a habilidade de desenhar diagramas estruturados e não-estruturados. *GEF* não é um programa de desenho, mas dá suporte à construção de programas customizados para domínios particulares.

Na Figura 7 é ilustrado o refinamento da subentidade *Editor Gráfico*. Na estruturação apresentada, a subentidade *Interface do Editor Gráfico* é responsável por disponibilizar os recursos para edição de objetos gráficos e demais funcionalidades aos projetistas de aplicações. Tal subentidade comunica-se com o projetista pelos pontos de interação *itr_ge* (requisição de funcionalidades) e *itr_gs* (apresentação de resultados); comunica-se ainda com a subentidade *Sistema de Ajuda do Editor Gráfico* por meio do ponto de interação *ig_sa* e com a subentidade *Representação de Elementos Conceituais e Gráficos de um Dispositivo* pelo ponto de interação *ig_decg*. Essa subentidade é responsável ainda, por fornecer as definições dos objetos gráficos para a subentidade *Interface do Editor Gráfico* e a especificação no formato da linguagem do dispositivo, no ponto de interação *spc_eg*, para ser traduzida para o formato suportado pelo modelo lógico.

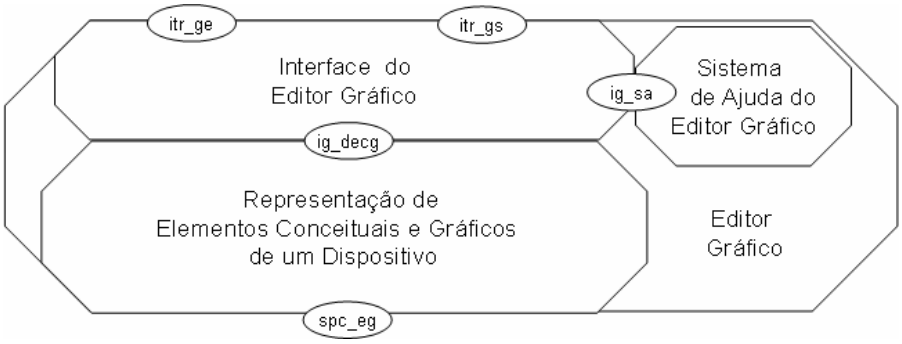


Figura 7. Refinamento da subentidade Editor Gráfico

Também foi proposto para as Ferramentas de Edição do APAA, o *Editor que usa a Linguagem do Modelo Lógico*. Tal ferramenta se apóia em uma interface semelhante à utilizada para aplicações comercias - caixas textos - e possibilita a um especialista definir um novo dispositivo. Esse editor também permite aos especialistas/projetistas realizarem a especificação de suas aplicações nessa mesma interface. Tanto os dispositivos definidos e as especificações realizadas são armazenados diretamente nas estruturas do modelo lógico utilizado. Na Figura 8 é apresentado o refinamento da subentidade *Editor Linguagem Modelo Lógico*. Tal subentidade é organizada em duas subentidades: *Interface do Editor que usa a linguagem do Modelo Lógico* e *Sistema de Ajuda do Editor que usa a linguagem do Modelo Lógico*. A subentidade *Interface do Editor que usa a linguagem do Modelo Lógico* recebe as informações vindas do projetista no ponto de interação *itr_emle* e apresenta as especificações e definições disponíveis no ponto de interação *itr_emls*. Tal subentidade fornece os recursos necessários para a definição de formalismos e a especificação de aplicações em uma interface de caixas textos e comunica-se com a subentidade *Sistema de Ajuda do Editor que usa a Linguagem do Modelo Lógico* por meio do ponto de interação *ig_sa*.

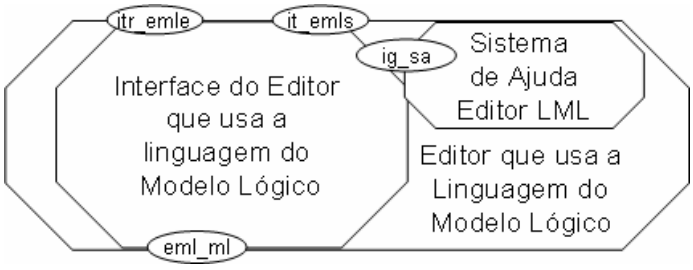


Figura 8. Refinamento subentidade Editor Linguagem Modelo Lógico

3.2.3. Ferramenta para simulação e verificação

As ferramentas de simulação e verificação devem facilitar o trabalho de análise das especificações produzidas. Na Figura 9 é descrito o refinamento da subentidade *Ferramenta de Análise (FA)* que é composta por um conjunto de ferramentas para a análise das especificações das aplicações. A subentidade *Ferramenta de Análise* é refinada nas seguintes subentidades: *Simulador de Aplicação Adaptativa*, *Verificador de Aplicação Adaptativa*, *Núcleo Adaptativo* e *Tradutor da linguagem do Modelo Lógico para a linguagem do Dispositivo*.

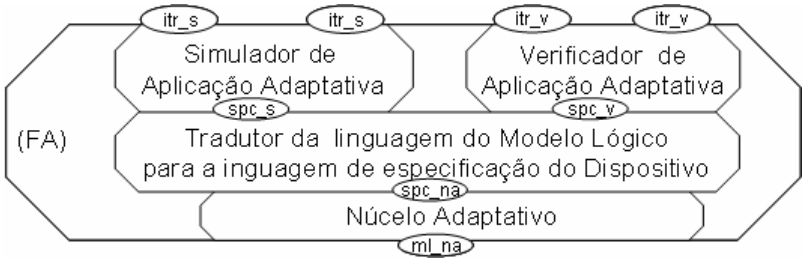


Figura 9. Representação da subentidade Ferramenta de Análise

O Simulador de Aplicação Adaptativa deverá permitir simular de forma interativa e automática as especificações elaboradas a partir de um dispositivo subjacente e/ou adaptativo. As opções disponibilizadas por tal ferramenta deverão ser: numa simulação passo a passo, fornecer um traço das ações ocorridas durante a passagem do tempo ou modificação dos valores das variáveis dos processos controlados pelo projetista; no caso de uma simulação automática, apresentar uma árvore - ou parte dela - de traços de execução contendo informes dos critérios de terminação da simulação e probabilidades de ocorrência de uma ação especificados pelo projetista.

Na Figura 10 é apresentado o refinamento do referido simulador. A subentidade *Simulador de Aplicação Adaptativa* tem por objetivo o recebimento de estímulos de entrada e de apresentação dos resultados de uma simulação realizada. Os estímulos fornecidos pelo projetista no ponto de interação *itr_s* são disponibilizados à subentidade *Tradutor da linguagem do Modelo Lógico para a linguagem do dispositivo* por meio do ponto de interação *spc_s*. A subentidade *Tradutor da linguagem do Modelo Lógico para a linguagem do dispositivo* converte os estímulos recebidos no formato do Modelo Lógico e fornece os mesmos para a entidade *Núcleo Adaptativo*. Essa subentidade ainda é responsável por receber os resultados referentes às simulações realizadas no Núcleo Adaptativo e convertê-los

para o formato suportado pelo Simulador de Aplicação Adaptativa para que possam ser apresentados para o projetista.

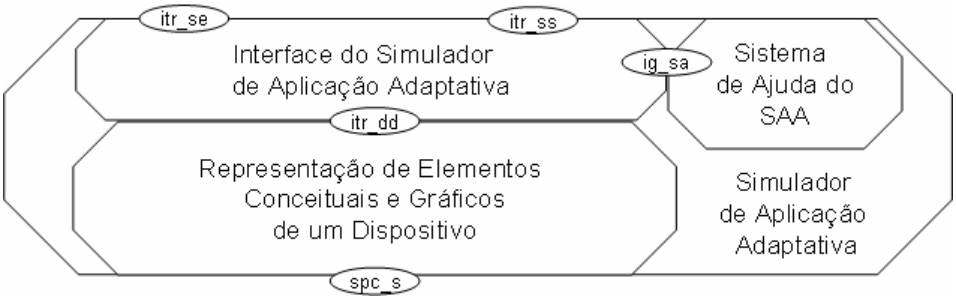


Figura 10. Refinamento subentidade Simulador Aplicação Adaptativa

Com o objetivo de atender às necessidades de interpretação e execução de especificações adaptativas, a subentidade *Núcleo Adaptativo,* cujo refinamento é ilustrado na Figura 11, funciona como um gerenciador de execução de especificações e se fundamenta nos conceitos de tecnologia adaptativa definidos anteriormente. A subentidade *Núcleo Adaptativo* é formada pela composição de três subentidades: *Função Adaptativa Anterior*, *Forma de Operação do Dispositivo* e *Função Adaptativa Posterior*. A subentidade *Núcleo Adaptativo* recebe os estímulos oriundos das chamadas realizadas pelo simulador ou verificador no ponto de interação *spc_na*, recebe também a especificação de uma aplicação vinda da subentidade *Modelo Lógico* no ponto de interação *ml_na* e, ainda, realiza as suas ações e fornece os resultados no ponto de interação *spc_na.* As subentidades *Função Adaptativa Anterior* e *Função Adaptativa Posterior* são responsáveis pela execução das funções e ações adaptativas – no caso de terem sido declaradas – e se comunicam, respectivamente, por meio dos pontos de interação *faa_f*o e *fap_f*o com a subentidade *Forma de Operação do Dispositivo* que deve realizar as ações específicas do dispositivo conforme a sua forma de operação.

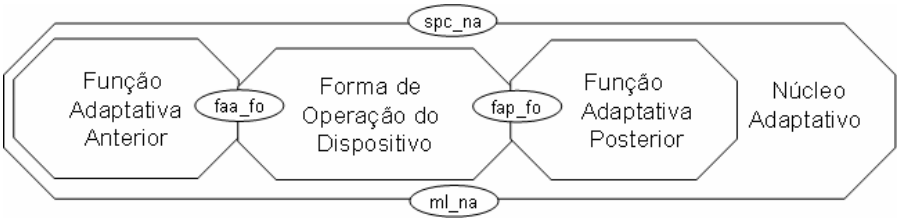


Figura 11. Refinamento da subentidade Núcleo Adaptativo

O Verificador de Aplicações Adaptativas deve permitir a realização de uma análise do comportamento lógico e temporal da aplicação adaptativa projetada. Tal ferramenta deverá basear-se no modelo lógico e gerar estruturas que permitirão

realizar verificações de propriedades, que são representadas por fórmulas lógicas, ou pode-se determinar regiões alcançáveis para certas condições.

Na Figura 12 é apresentado o refinamento da subentidade *Verificador de Aplicação Adaptativa*. Em tal refinamento observa-se que a subentidade *Verificador de Aplicação Adaptativa* recebe os estímulos oriundos dos projetistas pelo ponto de interação *itr_v,* realiza as tarefas de verificação – de forma semelhante ao simulador – e fornece os resultados aos projetistas no ponto de interação *itv_s*.

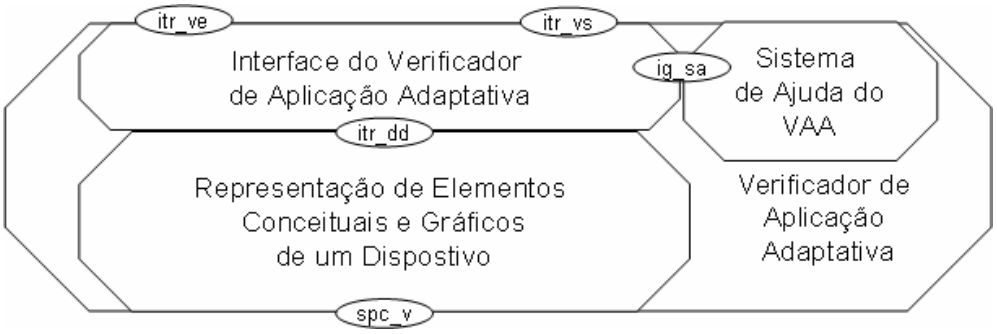


Figura 12. Refinamento da subentidade Verificador Aplicação Adaptativa

3.2.4. Ferramenta para a Produção de Representação Física

As especificações das aplicações realizadas e representadas no modelo lógico poderão ser traduzidas de forma automática para uma representação física (linguagem de programação). A ferramenta de tradução é de fundamental importância para os projetistas, pois evita o trabalho de implementação das aplicações projetadas, além de não permitir a geração de inconsistência das aplicações produzidas, uma vez que já foram analisadas. Sendo assim, as aplicações produzidas com o uso desta ferramenta estarão livres de inconsistência geradas durante a implementação e deverão produzir os resultados desejados pelo projetista da aplicação.

Para a construção desta ferramenta é sugerida como linguagem de saída – linguagem de representação física - a linguagem Java (JAVA, 2007). Java é uma linguagem de programação orientada a objetos e *multitarefa.* Uma aplicação Java pode ser interpretada por todo sistema operacional ou máquina que possua uma máquina virtual Java. Esta portabilidade, adicionada as suas bibliotecas gráficas e de rede, tornam a linguagem Java uma boa candidata ao projeto de aplicações adaptativas.

A produção de aplicações Java pode ser obtida diretamente a partir da representação de uma aplicação no formato do Modelo Lógico e essa tradução só

poderá ser habilitada após a validação da especificação. Também vale considerar que outras linguagens de representação física podem ser usadas como saída e, para tal, será necessário realizar a transformação da representação do Modelo Lógico para a linguagem desejada. Na Figura 13 é apresentado o refinamento da subentidade *Ferramenta de Produção de Representação Física*. Tal subentidade é constituída de uma interface para recebimento das requisições realizadas por um projetista no ponto de interação *itr_prf*. Ao ser solicitada a geração de uma representação física de uma aplicação, a subentidade *Gerador de Código de Representação Física* é acionada por meio do ponto de interação *ig_grf*. Essa subentidade comunica-se com a subentidade *Representação de Elementos Conceituais de um Dispositivo* no ponto de interação *grf_recf* e obtém a especificação de uma aplicação e as definições conceituais de um dispositivo. Tais informações são necessárias para a geração de código no formato da representação física. O código gerado será disponibilizado no ponto de interação *itr_prg* e ficará disponível para ser compilado e, conseqüentemente, um programa executável será obtido.

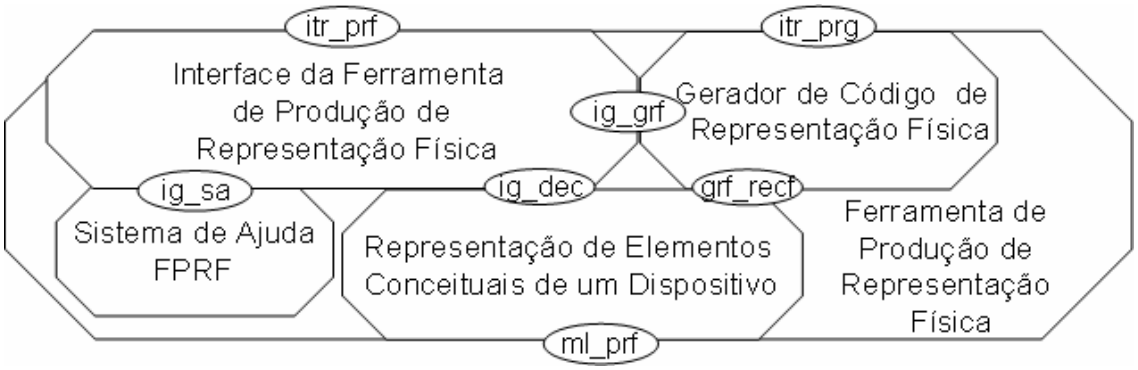


Figura 13. Refinamento subentidade Ferramenta para a Produção de Representação Física

3.3. Proposta do gerador de ambientes

Com o objetivo de atender às funcionalidades desejadas para o desenvolvimento de cada ambiente específico de cada novo dispositivo definido, justifica-se a proposição do gerador de ambientes para modelagem de aplicações usando tecnologia adaptativa.

3.3.1. Definição de metambientes

A estrutura do gerador de ambientes proposto fundamenta-se nas definições de metamodelos e metambientes. Na Figura 14, pode ser observada uma hierarquia organizada em níveis para a definição de modelos, metamodelos e os seus níveis superiores. No primeiro nível, tem-se a definição de modelo; no segundo nível, a definição de um metamodelo e, no terceiro nível, um metamodelo de nível 2, e assim sucessivamente.

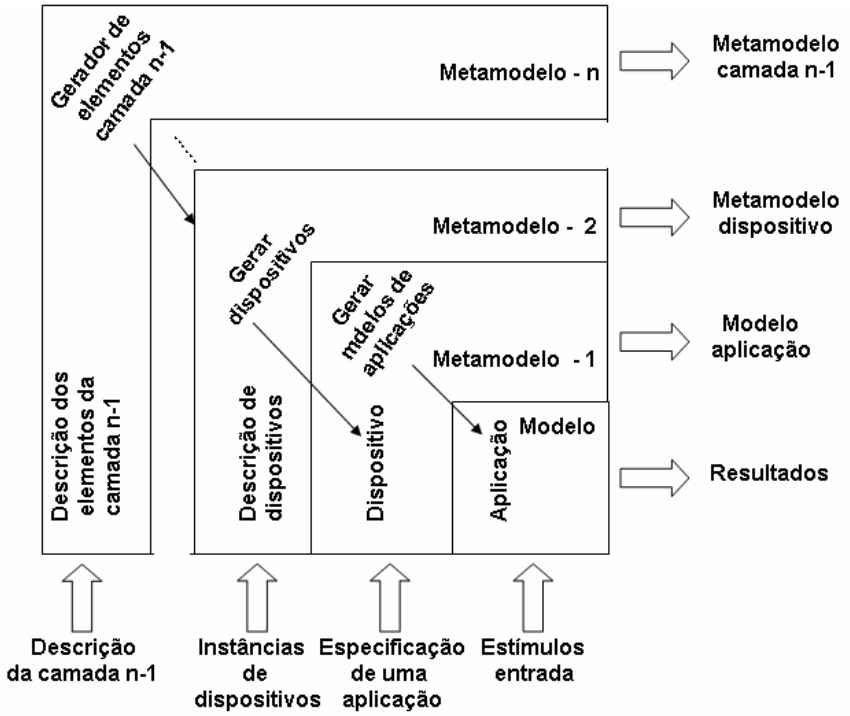


Figura 14. Hierarquia de metamodelos

A estrutura definida permite a definição de diversos níveis de metamodelos e permite que o nível superior defina (represente) os elementos (estruturas, objetos, etc.) do nível imediatamente inferior. Por exemplo, no nível de *Modelo* tem-se a representação de uma aplicação que foi especificada valendo-se de um determinado dispositivo (Autômatos, Rede de Petri, etc.). Nesse nível, a especificação produzida de uma aplicação está armazenada no formato definido para o nível *Metamodelo (por exemplo o Modelo Lógico)* e, injetando os e*stímulos de entrada em um modelo da aplicação* os *resultados* são obtidos por intermédio da execução de uma ferramenta de análise ou produção física representada na seção anterior. Para que um modelo seja definido, faz-se necessário a descrição dos elementos conceituais de seu dispositivo. Tais definições são representadas por uma estrutura de dados

(metaestrutura) definida para o nível *Metamodelo 1*. Tal estrutura permite a instanciação de objetos definidos para este nível e a obtenção de modelos de uma aplicação com base em um dispositivo específico. Num outro nível ainda, o *Metamodelo 2*, poderia ser definida uma metaestrutura que representasse os elementos conceituais de cada dispositivo e, conseqüentemente, ter-se-ia a definição de uma estrutura que representasse os elementos do *Metamodelo 1*, ou seja, um especialista poderia utilizar a metaestrutura definida no nível *Metamodelo 2* e instanciar objetos nela para obter um novo dispositivo. Com base no dispositivo obtido, novos modelos para as aplicações no nível *Modelo* poderiam ser gerados.

Fundamentado nos conceitos apresentados para os níveis de *Metamodelo 1* e *2*, podem ser realizadas definições nos níveis superiores e, ainda, criadas metaestruturas que permitam a representação dos níveis imediatamente inferiores, ou seja, no nível *Metamodelo 3* pode ser definida uma metaestrutura que descreva os elementos do nível *Metamodelo 2* e assim sucessivamente. Tal estrutura permite, por exemplo, a definição no nível *Metamodelo 3* de uma metaestrutura que represente formalismos de diversas características (formalismos discretos, formalismos contínuos). A metaestrutura gerada no nível *Metamodelo 3* poderá ser instanciada e uma metaestrutura (nível *Metamodelo 2)* ser obtida para representar os diversos dispositivos referentes a tal categoria.

De forma semelhante à realizada para a definição de modelos e metamodelos podem ser definidos ambientes e metambientes. Na Figura 15 é apresentada uma hierarquia de organização de ambientes e metambientes em níveis. O nível superior define automaticamente e gera os elementos do nível imediatamente inferior. No nível mais externo, observa-se a existência de um gerador de metambientes que, por meio da descrição de um metamodelo, realiza a definição dos elementos (modelo ou metamodelo) de nível imediatamente inferior. Ainda nesse nível, tem-se também a definição da forma de operação das ferramentas que operam no nível imediatamente inferior. Com base em tais descrições, o gerador permite a obtenção de ferramentas que possam realizar as funcionalidades definidas para o nível imediatamente inferior ao realizado.

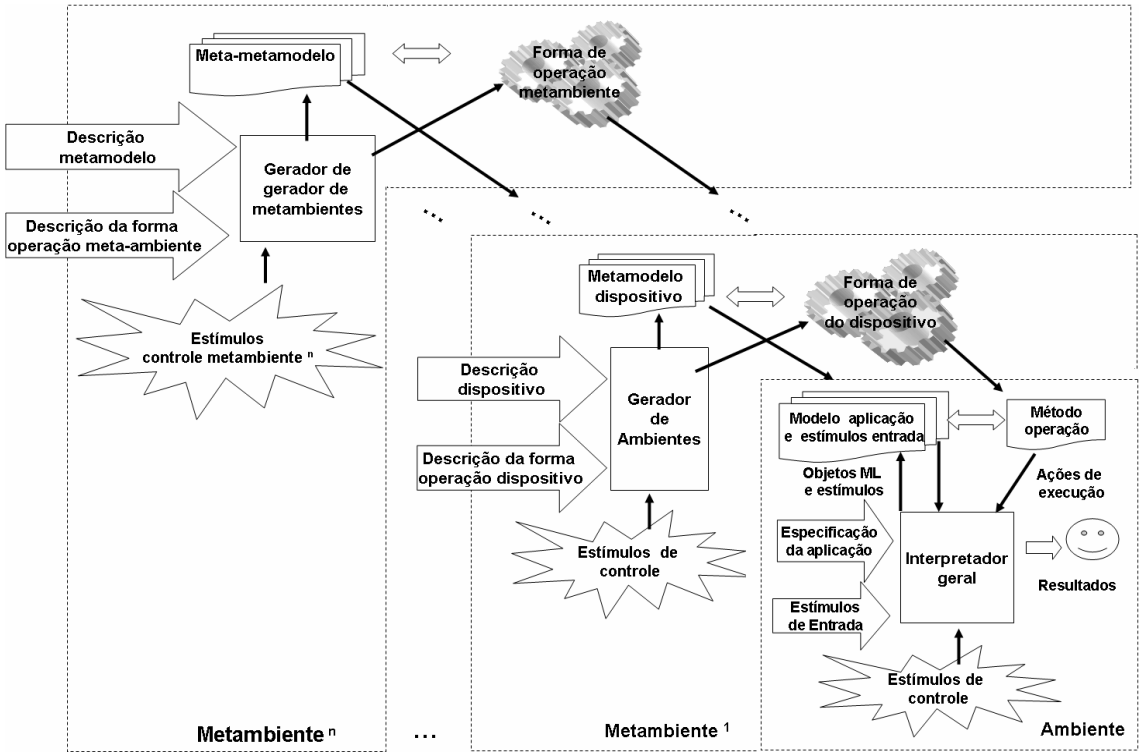


Figura 15. Arquitetura geral para geradores de ambientes

A arquitetura geral proposta para a definição de metambientes permite a existência de metambientes (geradores) e metaestruturas (modelos lógicos) que definem a geração das estruturas (metaestruturas) e ferramentas (metambientes) em diversos níveis, até serem obtidas as estruturas (modelo) e ferramentas (ambiente). Vale ressaltar que no escopo deste trabalho são definidas e utilizadas estruturas e ferramentas para o nível *Metamodelo 2* e os seus imediatamente inferiores.

3.3.2. Arquitetura do gerador de ambientes

O gerador de ambientes consiste em ferramentas que permitam a descrição de um dispositivo específico, suas definições gráficas e a forma de operação. Com base em tais definições, o gerador de ambientes é executado e obtém como resultado um ambiente para o dispositivo específico e que permite desempenhar as funcionalidades desejadas por um projetista. Na Figura 16 é ilustrada a arquitetura geral para o gerador de ambientes. Tal arquitetura define um metambiente composto por um *framework* de um ambiente geral – definição de uma estrutura que contém os elementos comuns a todos os ambientes e pode ser reaproveitada - e por quatro

ferramentas: Ferramenta para Definição dos Elementos Conceituais, Ferramenta para Definição dos Elementos Gráficos, Ferramenta para Definição da Forma de Operação de um Dispositivo e Ferramenta para a Geração de Ambientes.

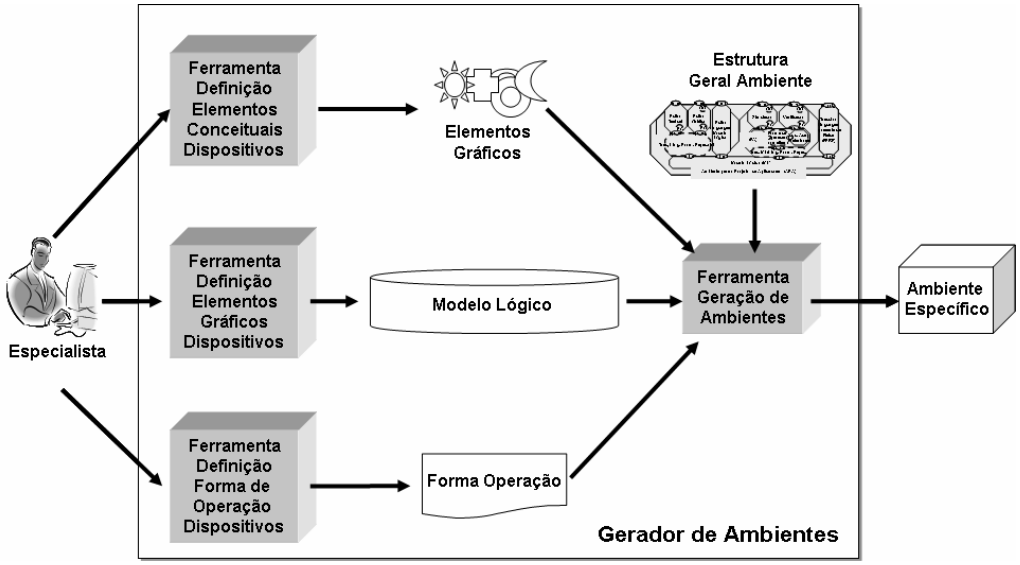


Figura 16. Arquitetura geral do gerador de ambientes

Na arquitetura proposta, o *Framework* de um Ambiente Geral é um conjunto de programas (classes) que descreve as funcionalidades comuns a todos os projetistas de aplicações adaptativas. Na subseção anterior deste capítulo, foi apresentado o Ambiente para o Projeto de Aplicações Adaptativas (APAA) que serve de base para a concepção do *framework* proposto. Tal *framework* permite a obtenção de um ambiente específico sem a necessidade de refazer a implementação de todos os componentes pertencentes ao mesmo. Um programador, com bons conhecimentos no *framework* e na linguagem para sua representação física, pode inserir os códigos necessários para que as funcionalidades específicas com base no dispositivo utilizado possam ser disponibilizadas no ambiente.

Com o objetivo de facilitar a implementação do ambiente e facilitar o trabalho de realização no *framework* é que se propõe a utilização da *Ferramenta de Definição dos Elementos Conceituais de um Dispositivo*, que permitirá a um especialista instanciar objetos no modelo lógico e definir a representação dos elementos conceituais do dispositivo desejado.

Os objetos gráficos associados aos elementos conceituais de um dispositivo também devem ser definidos para poderem ser utilizados pelas ferramentas de edição e análise. Para tal, é proposto a *Ferramenta para a Definição dos Elementos*

Gráficos de um Dispositivo, que possibilita a um especialista definir os objetos gráficos - no caso da existência de tal representação - dos elementos conceituais do novo dispositivo. Os novos elementos definidos comporão, posteriormente, a barra de ferramentas e os menus de opções - da ferramenta de edição do ambiente específico para modelagem -, responsáveis por permitir a inserção de tais objetos durante a edição da especificação de uma aplicação e que também servirão para apresentação do modelo da aplicação quando da sua simulação e verificação.

Além dos elementos descritos anteriormente, também se faz necessária a descrição da forma de operação de um dispositivo. Para esse trabalho, apresenta-se a *Ferramenta de Definição da Forma de Operação de um Dispositivo*. Tal ferramenta permite que um especialista descreva as ações que os simuladores e verificadores devem desempenhar sobre os elementos de especificação – armazenados no modelo lógico – de uma determinada aplicação durante a fase de análise.

Com base no *framework* para um ambiente geral e nos objetos definidos pelas ferramentas para definição de dispositivos, elementos gráficos e forma de operação, executa-se a *Ferramenta de Geração de Ambientes*. Tal ferramenta realiza de forma automática a definição de módulos que serão anexados ao ambiente geral e uma nova ferramenta para um dispositivo específico será obtida.

3.3.3. Arquitetura do *framework* para um ambiente geral

A arquitetura proposta para o *framework* fundamenta-se em um estilo de programação na qual se permite que a definição de um ambiente seja estruturada, possibilitando que novos módulos, também chamados *plugins*, sejam adicionados sem que haja necessidade de recompilação e reestruturação do ambiente já desenvolvido. Tal conceito foi empregado para a estruturação do ambiente para desenvolvimento de aplicações *Eclipse* (ECLIPSE, 2007). Esse ambiente possui um conjunto de operações básicas e permite que novos módulos sejam adicionados. O ambiente *Eclipse* foi desenvolvido pela IBM para a realização de programas utilizando a linguagem Java, porém adicionando-se novos módulos *(plugins)*, pode-se modificar a linguagem fonte, por exemplo, C/C++ ou ainda podem ser adicionada novas funcionalidades como, por exemplo, a possibilidade de realização de

modelagem de aplicações utilizando a linguagem de modelagem e ferramentas UML.

A possibilidade de definição de novos módulos permite uma maior facilidade e flexibilidade no desenvolvimento das aplicações e um melhor reaproveitamento dos códigos dos programas definidos para a estruturação do referido ambiente.

Com base nos conceitos de programação por módulos e na definição de um ambiente para o projeto de aplicações adaptativas especificado na subseção anterior é que se propõe a arquitetura para um *framework* de um Ambiente Geral. Na Figura 17 é apresentada a arquitetura do referido ambiente e seus refinamentos.

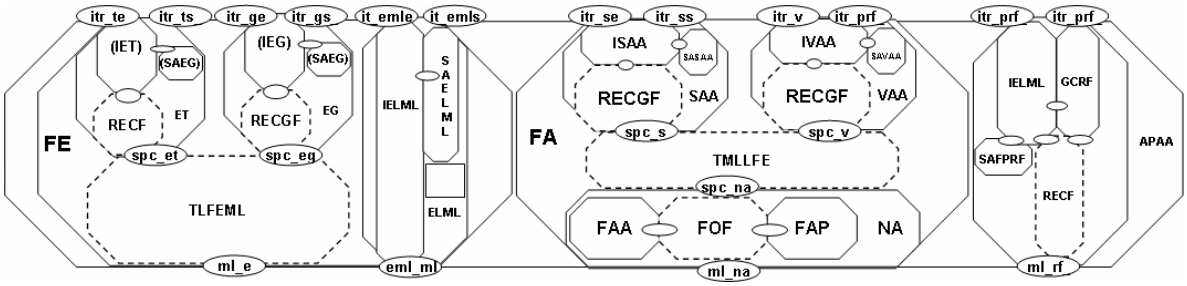


Figura 17. Definição de um F*ramework* para um Ambiente Geral

Na arquitetura apresentada pode ser observado que algumas (sub)entidades foram ilustradas em linhas sólidas, enquanto outras foram apresentadas em linhas pontilhadas. As (sub)entidades ilustradas com linhas sólidas representam os módulos que não necessitam ser modificados, enquanto os representados por linhas pontilhadas indicam os módulos que podem ser redesenvolvidos para que novos dispositivos possam ser definidos pela utilização da estrutura já definida para o ambiente para o projeto de aplicações adaptativas.

Observa-se que, após a construção do referido ambiente, um programador necessitará redesenvolver apenas as interfaces de tradução da linguagem de um dispositivo especifico para o modelo lógico e uma outra interface que permita a realização da operação inversa, tradução do modelo lógico para a linguagem de um dispositivo específico. O programador também deverá definir a forma de representação (textual ou gráfica) dos elementos conceituais de um dispositivo. Tais elementos são utilizados pelas ferramentas de edição e de análise. Além disso, o programador ainda deve realizar a programação da forma de operação do dispositivo, que será utilizada pelo núcleo adaptativo durante uma simulação ou verificação de uma aplicação.

3.3.4. Ferramenta para a definição dos elementos conceituais de dispositivos

A *Ferramenta para Definição dos Elementos Conceituais de Dispositivos* é uma ferramenta computacional textual ou gráfica que permite a um especialista definir os elementos conceituais do dispositivo desejado. No caso desta ferramenta ser textual, deverá possuir uma linguagem para a representação de dispositivos e para a realização de especificação utilizando os mesmos. Para que este tipo de ferramenta possa ser utilizado, faz-se necessária a utilização de um tradutor que permita aos elementos conceituais e elementos de especificação serem traduzidos para o formato suportado pelo modelo lógico.

No caso de a ferramenta ser gráfica, o projetista deverá definir os elementos em um editor gráfico e, por meio de uma caixa de propriedades, definir os elementos conceituais do dispositivo. As definições realizadas na caixa de propriedades são automaticamente armazenadas no formato do modelo lógico. Uma outra forma, semelhante à da caixa de propriedades, mas que não define diretamente os objetos gráficos, é a utilização de um ambiente que disponibilize caixas textos e permita ao especialista representar um dispositivo específico. Esta ferramenta gera automaticamente os elementos no modelo lógico. Tal ferramenta é representada no APAA por meio do Editor que usa a Linguagem do Modelo Lógico. A utilização deste tipo de ferramenta é a forma mais rápida para se obter ferramentas de definição de dispositivos – as interfaces se assemelham às desenvolvidas para aplicações comercias. São simples e de fácil implementação, mas estas interfaces contam com a inconveniência de não se serem tão intuitivas quanto outras interfaces e tampouco revelam-se amigáveis em sua utilização pelo especialista, ainda que este conte com uma considerável experiência neste tipo de trabalho, fatores estes que tornam o trabalho de definição de dispositivos algo mais tedioso e substancialmente demorado.

3.3.5. Ferramenta para a definição dos objetos gráficos de dispositivos

A *Ferramenta para a Definição dos Objetos Gráficos de dispositivos* permite a um especialista fornecer as características elementares dos objetos gráficos

associados aos elementos conceituais de um dispositivo. Tal ferramenta pode ter uma interface textual ou gráfica. A interface textual consiste em um editor associado a uma linguagem para a representação de interfaces e também numa associação dos objetos definidos aos elementos conceituais do dispositivo. A ferramenta Atom^3 (Atom3, 2007) utiliza-se deste conceito e permite que os objetos gráficos de um determinado dispositivo sejam representados diretamente na codificação física da ferramenta. Para tal, são utilizados os comandos e as funções das bibliotecas da linguagem de representação física Python (PYTHON, 2007). Este tipo de processo caracteriza-se pela facilidade e rapidez de sua utilização devido ao fato de não se necessitar realizar o desenvolvimento de novas ferramentas - editores e compiladores, os objetos são definidos diretamente no ambiente valendo-se da linguagem de representação física. A definição textual também poderia utilizar-se de um editor e de uma linguagem – para a representação de objetos gráficos – concebida para tal objetivo. Ao se utilizar este tipo de processo, faz-se necessária a utilização de um compilador que permita a transformação das especificações de objetos gráficos definidos para a linguagem utilizada pelo ambiente gráfico para edição de especificação e visualização de simulações e verificações. Este tipo de ferramenta é um pouco mais demorado para ser obtido, pois tornam-se necessários a definição de uma nova linguagem para definição de elementos gráficos e a construção das ferramentas associadas a esta linguagem. Porém este trabalho é compensado pelo fato de a interface do especialista ser um pouco mais amigável e prática. Finalmente, uma ferramenta mais adequada seria a utilização de uma ferramenta gráfica que permitisse a associação de ambas as ferramentas: definição conceitual e definição de elementos gráficos. Tal ferramenta deveria ter as mesmas características da ferramenta de definição conceitual gráfica, pois um especialista utiliza-se de um mesmo ambiente para definir os elementos gráficos e fazer a associação dos objetos gerados com os elementos conceituais do dispositivo. Esta ferramenta pode automatizar a produção e proporcionar uma boa produtividade para os especialistas ao realizar a definição dos elementos conceituais e gráficos de um dispositivo, porém o seu processo de desenvolvimento é um pouco mais demorado. Na Figura 18 é apresentada a arquitetura da *Ferramenta para a Definição de Objetos Gráficos de Dispositivos*. Tal ferramenta possui uma interface com o especialista que deverá permitir a este a definição dos objetos gráficos relacionados a um dispositivo. No ponto de entrada *itr_iogle*, a ferramenta recebe os estímulos

dos especialistas referentes à definição dos objetos gráficos e à associação dos mesmos aos respectivos elementos conceituais de um dispositivo. A subentidade *Ferramenta para a Definição de Objetos Gráficos de Dispositivos* comunica-se com a subentidade *Modelo Lógico* para obter as informações dos elementos conceituais de um dispositivo e se comunica com a subentidade *Objetos Gráficos*, responsável por armazenar as definições dos elementos gráficos realizadas.

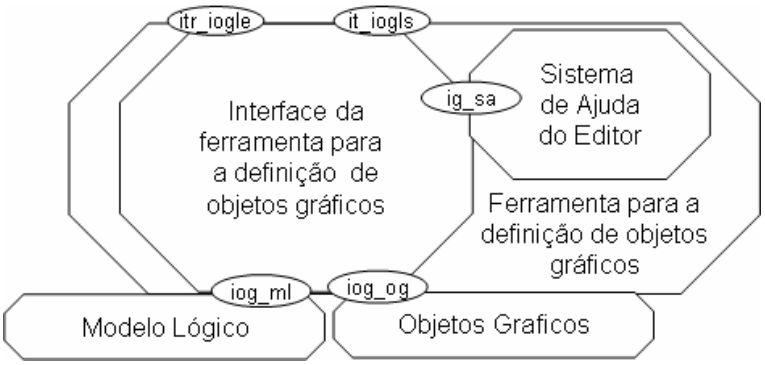


Figura 18. Arquitetura da Ferramenta para a Definição de Objetos Gráficos

3.3.6. Ferramenta para a definição da Forma de Operação de um Dispositivo

A *Ferramenta para definição da Forma de Operação de um Dispositivo* permite a um especialista, por intermédio de uma interface textual, definir a forma de operação do novo dispositivo. Ao realizar esta tarefa, o especialista pode desenvolvê-lo de forma direta, com a qual terá acesso a alguns métodos de um *framework* para implementação de ferramentas. Utilizando-se do *framework*, o especialista modifica apenas as partes essenciais e obtém as novas ferramentas desejadas. Uma outra maneira seria utilizar-se de uma ferramenta que permita a edição da forma de operação, com base em uma linguagem pré-definida. Depois de obtida a especificação da forma de operação, esta é compilada ou traduzida para a representação da linguagem física ou para a linguagem processada pelas ferramentas de análise e demais ferramentas que compõe o ambiente para modelagem de aplicações.

A forma de definição direta que utilize um *framework* é o modo mais rápido para se trabalhar com dispositivos, pois não é necessário realizar o desenvolvimento de linguagens para especificação, nem a construção de compiladores/tradutores para tais linguagens. O especialista realiza todo o processo utilizando a própria

linguagem de representação física e escreve a forma de operação de um dispositivo diretamente num código já existente. Tal tarefa exige que o especialista conheça os comandos da linguagem de representação física e a arquitetura do programa existente, o que o obriga, invariavelmente, a ter que estudar tais linguagens e codificações, que não são tão triviais e nem de tão fácil entendimento. A utilização de um ambiente específico tem por objetivo facilitar o serviço de definição da forma de operação e permite a utilização do novo dispositivo definido de uma forma mais rápida. Na Figura 19 é apresentada a arquitetura da *Ferramenta para a Definição da Forma de Operação de um Dispositivo*.

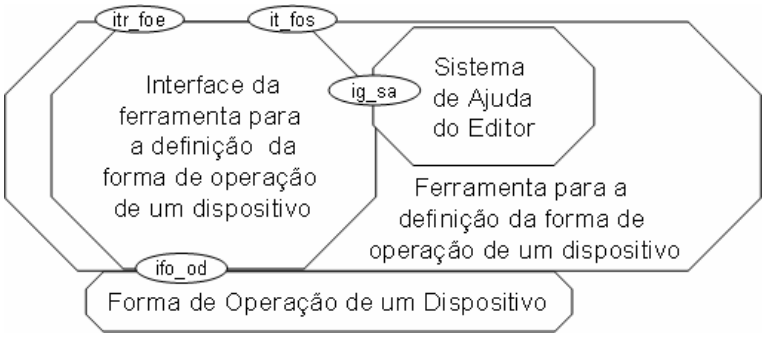


Figura 19. Arquitetura da Ferramenta para a Definição da Forma de Operação de um Dispositivo

A subentidade *Ferramenta para a Definição da Forma de Operação de um Dispositivo* recebe a forma de operação de um dispositivo por intermédio do ponto de *itr_foe*, realiza o seu processamento e disponibiliza para a entidade *Forma de Operação de um Dispositivo* uma especificação da operação de um dispositivo no ponto de interação *fo_od.*

3.4. Modelo lógico para dispositivos adaptativos

A definição de um modelo lógico para a representação de dispositivos adaptativos é de fundamental importância na concepção de ambientes para a modelagem de aplicações e na concepção de um gerador de tais ambientes.

Nessa seção será descrita uma metaestrutura (classes) que constitui a arquitetura do modelo lógico proposto. A metaestrutura permite o mapeamento dos elementos conceituais de um dispositivo e das aplicações que o utilizam como base.

3.4.1. Organização do Modelo Lógico

O Modelo Lógico deve exprimir os elementos conceituais de um dispositivo geral para permitir a representação das características específicas de cada um dos dispositivos dirigidos por regras que são representados no mesmo. Este modelo constitui-se em uma estrutura de dados que permite a instanciação de diversos dispositivos por meio dos metadados definidos. Desta forma, um especialista define a partir de uma metaestrutura existente objetos que comporão o metamodelo e definirão os elementos conceituais de um dispositivo. O dispositivo definido passa então a ser um metamodelo para definição das aplicações projetadas com base nele. Quando um projetista utilizar um dispositivo definido no modelo lógico ele estará instanciando objetos do metamodelo e definindo modelos de sua aplicação.

Para facilitar o entendimento e possibilitar uma melhor organização do modelo lógico, este foi concebido em camadas. As camadas foram organizadas conforme o nível de representação de cada entidade que constitui o modelo e as características dos especialistas/projetistas que terão acesso às informações armazenadas nas entidades de cada camada.

Outra característica também considerada ao se definir o modelo lógico para dispositivos adaptativos foi a definição de uma arquitetura que separe os objetos de especificações da camada adaptativa dos objetos de especificações da camada subjacente, conforme fora definida na representação geral para dispositivos adaptativos. Tal estruturação é de fundamental importância, pois as ferramentas, que forem criadas com base no modelo lógico definido, poderão realizar especificações, simulações, verificações e teste de aplicações adaptativas bem como especificações de aplicações que não possuem tais características. A definição do Modelo Lógico seguindo este princípio foi fundamental para permitir aos projetistas utilizarem a tecnologia adaptativa como um opcional, podendo usar ou não esta tecnologia conforme o seu interesse.

O Modelo Lógico para dispositivos adaptativos foi organizado em 4 (quatro) camadas: Camada de Dispositivo, Camada de Especificação Subjacente, Camada de Especificação Adaptativa e Camada de Memória. Tal arquitetura foi organizada de forma hierárquica e tem por objetivo atender as características definidas para a representação dos dispositivos adaptativos conforme ilustrado na Figura 20.

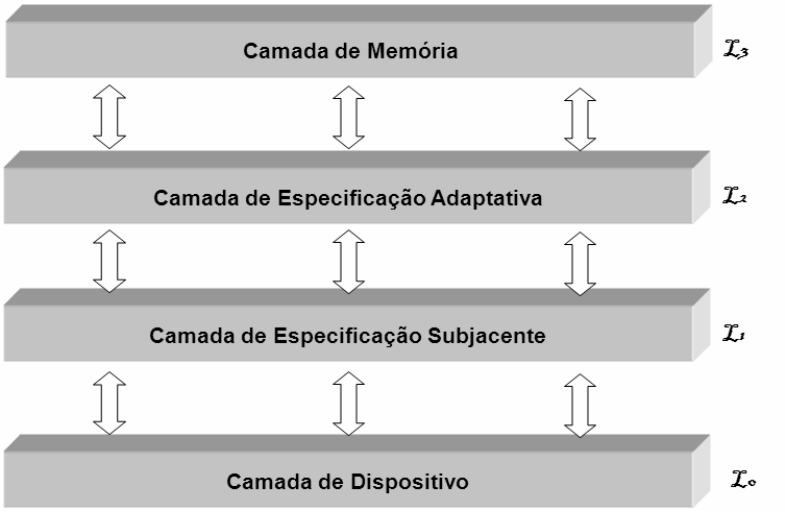


Figura 20. Organização de camadas do Modelo Lógico

Na arquitetura proposta, a camada $\mathcal{L}_0$ - Camada de Dispositivo serve para representar as propriedades e os elementos teóricos de um dispositivo subjacente. O especialista, ao desejar obter a representação lógica do dispositivo adaptativo desenvolvido, deve também mapear seus elementos formais para esta camada.

A camada, $\mathcal{L}_1$ - Camada de Especificação Subjacente tem por objetivo representar os elementos que compõem o comportamento não-adaptativo de uma aplicação. Ao realizar a modelagem de uma aplicação, um projetista instancia objetos da Camada de Dispositivo e produz elementos que representam a especificação não-adaptativa da mesma.

Depois de realizada a especificação subjacente, o projetista deve associar as funções e as ações adaptativas que fazem parte da especificação do comportamento de sua aplicação. Tais estruturas são representadas na arquitetura proposta por meio da camada $\mathcal{L}_2$ - Camada de Especificação Adaptativa. Esta camada representa os elementos conceituais para a concepção de dispositivos adaptativos definidos em (NETO, 2001).

Por fim, a camada $\mathcal{L}_3$, Camada de Memória, serve para representar os valores dos estímulos de entrada (cadeia de entrada, estímulos externos oriundos de outros agentes ou sistemas, etc.) e a situação atual do sistema (estado corrente ou transições habilitadas a ocorrer) em um determinado momento. Esta camada é de fundamental importância para se construir ferramentas para realizar simulações e verificações de aplicações utilizando os conceitos de tecnologia adaptativa.

Fundamentado na arquitetura proposta para representação de dispositivos adaptativos dirigidos por regras, foi desenvolvido um diagrama de classes,

representado em UML, para descrever as referidas classes de cada camada. A Figura 21 ilustra o referido diagrama e apresenta as classes, classificadas e organizadas em pacotes, conforme as camadas descritas anteriormente: Pacote da Camada de Dispositivo, Pacote da Camada de Especificação Subjacente, Pacote da Camada Adaptativa e Pacote da Camada de Memória, conforme as características as quais representam.

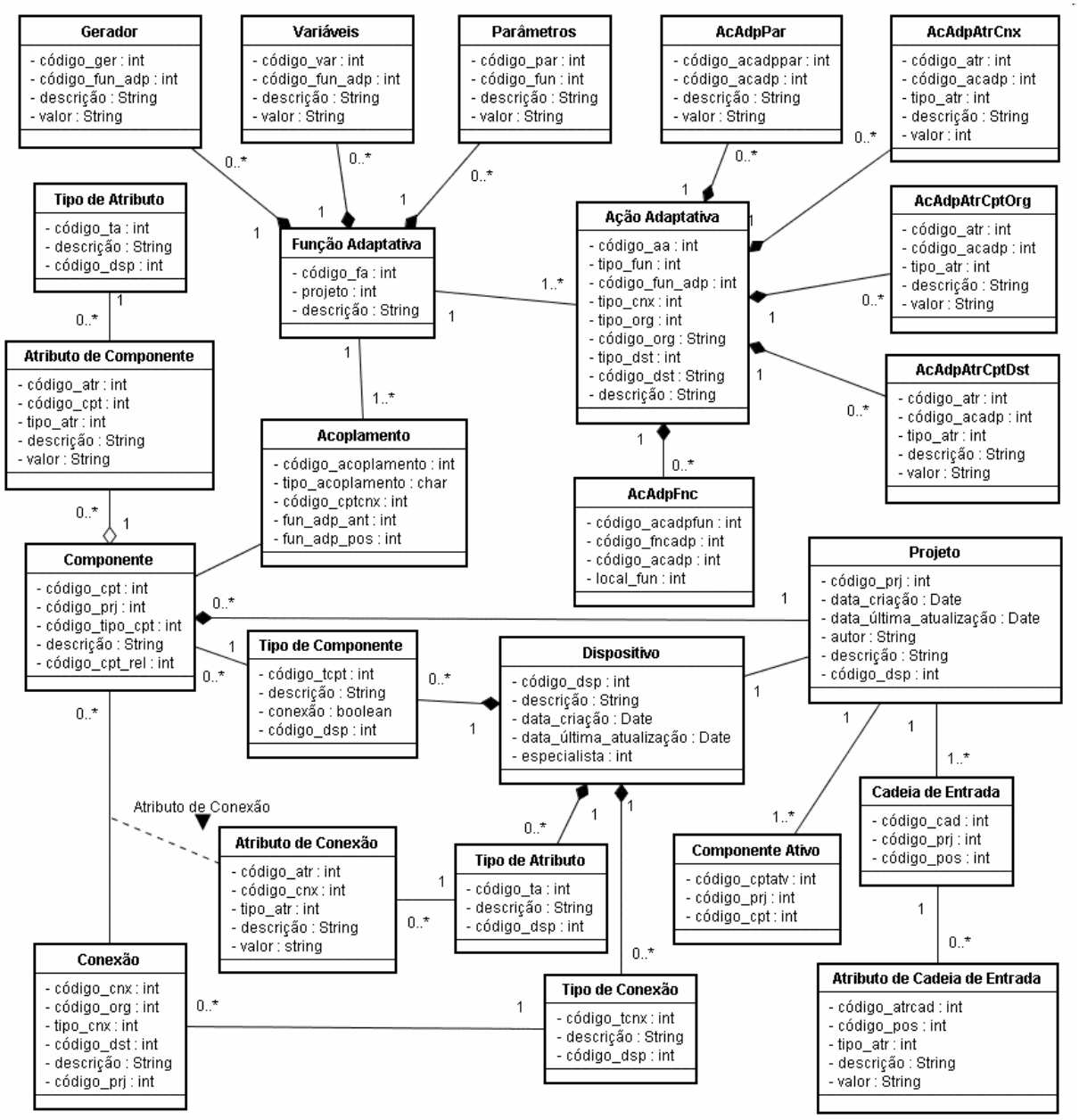


Figura 21. Diagrama de classes do Modelo Lógico para Dispositivos Adaptativos

As classes *Dispositivo, Tipo de Componente, Tipo de Conexão e Tipo de Atributo* pertencem ao Pacote da Camada de Dispositivo e representam as características formais do dispositivo adaptativo. As classes de *Projeto,*

Componente, Conexão, Atributo de Componente e Atributo de Conexão fazem parte do Pacote da Camada de Especificação Subjacente e têm por objetivo representar as especificações não-adaptativas dos modelos de aplicações desenvolvidos.

As classes *Função Adaptativa, Ação Adaptativa, AcAdpAtrCnx, AcAdpAtrCptOr, AcAdpAtrCptDs, AcAdpPar, AcAdpFnc, Gerador, Parâmetros, Variáveis e Acoplamento* pertencem ao Pacote da Camada de Especificação Adaptativa e têm por finalidade representar os elementos da camada adaptativa que envolve o núcleo subjacente. Por fim, as classes *Cadeia de Entrada, Atributo de Cadeia de Entrada e Componente Ativo* pertencem ao Pacote da Camada de Memória e representam a cadeia de entrada, seus atributos e os componentes ativos durante a execução de um modelo de uma aplicação.

Para maiores detalhes sobre cada uma das classes e seus respectivos atributos verifique o Apêndice A deste trabalho.

4. Implementação do gerador de ambientes para a modelagem de aplicações adaptativas

Para a realização do trabalho de implementação do gerador de ambientes foi utilizada a arquitetura proposta do ambiente geral para a modelagem de aplicações adaptativas e a arquitetura do gerador de ambientes propostos anteriormente neste trabalho. Também foi usada a metaestrutura definida para o Modelo Lógico na implementação do banco de dados para o armazenamento dos atributos dos dispositivos e das especificações de aplicações que os utilizam como base.

4.1. Escopo do Ambiente para a Modelagem de Aplicações Adaptativas e Gerador de Ambientes

Devido a grande complexidade inerente às arquiteturas propostas para o Ambiente para a Modelagem de Aplicações Adaptativas (APAA) e para o Gerador de Ambientes, foi selecionado um conjunto de ferramentas que seriam importantes para serem implementadas. As ferramentas foram escolhidas com o objetivo de permitir a validação dos conceitos apresentados neste trabalho, permitirem demonstrar a viabilidade e a utilização do ambiente e gerador propostos, possibilitar aos projetistas definirem dispositivos adaptativos e utilizá-los para o projeto de suas aplicações.

Neste contexto foi escolhido um conjunto de ferramentas *Editor* que usa a *Linguagem do Modelo Lógico* e o *Simulador de Aplicação Adaptativa.* Para a realização de tais ferramentas também são necessários a implementação dos módulos *Modelo Lógico*, *Tradutor da linguagem do Modelo Lógico para a linguagem do Dispositivo* e *Núcleo Adaptativo (NA).* A Figura 22 ilustra a arquitetura do APAA - definida no Capítulo 3 - e as subentidades representadas em linhas pontilhadas indicam os referidos módulos que devem ser implementados.

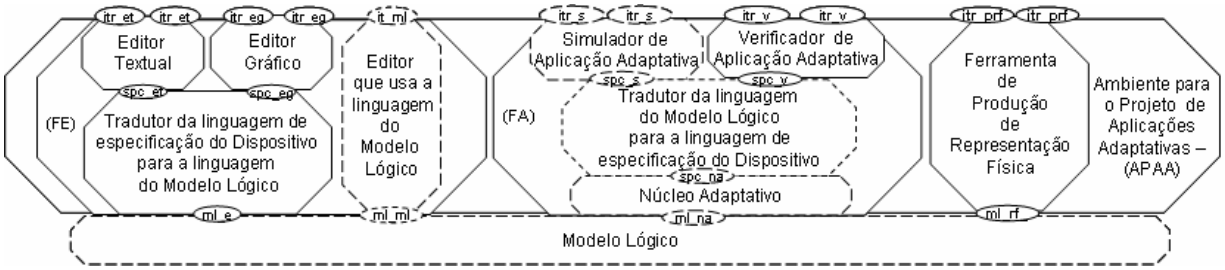


Figura 22. Arquitetura do APAA

Com base nas ferramentas escolhidas, foi realizado um estudo com o objetivo de obter uma arquitetura mais genérica e de rápida implementação. Neste sentido, foi definido um *Ambiente Simplificado para o Projeto de Aplicações Adaptativas. (ASPAA)*. O ambiente definido permitiu uma rápida implementação e ajudou na avaliação dos demais conceitos apresentados. Além disso, serviu como protótipo para auxiliar na definição e estruturação do ambiente APAA completo e gerador de ambientes. Para o ambiente ASPAA, foi definida uma ferramenta para definição de dispositivos e realização de modelagem usando a linguagem do Modelo Lógico – conforme definido anteriormente – e um Simulador de Aplicações Adaptativas que apresenta os resultados no formato da linguagem do Modelo Lógico e não no formato da linguagem de um dispositivo específico, como proposto para o ambiente APAA completo. Na Figura 23 é apresentada a arquitetura do ASPAA, da qual foram eliminados alguns módulos conforme dito anteriormente e seu ambiente foi definido com a subentidade *Editor Linguagem Modelo Lógico* para representar as *Ferramentas de Edição (FE)*. No contexto das *Ferramentas de Análise (FA)*, foi definido o *Simulador de Aplicação Adaptativa* juntamente com o seu respectivo *Núcleo Adaptativo (NA)*. Nota-se que foram abstraídas as traduções e representações de elementos textuais e gráficos de um dispositivo específico e os módulos referentes ao sistema de ajuda. Tais módulos podem ser desenvolvidos numa fase posterior a este trabalho, sugestão que, inclusive é apresentada nas considerações finais.

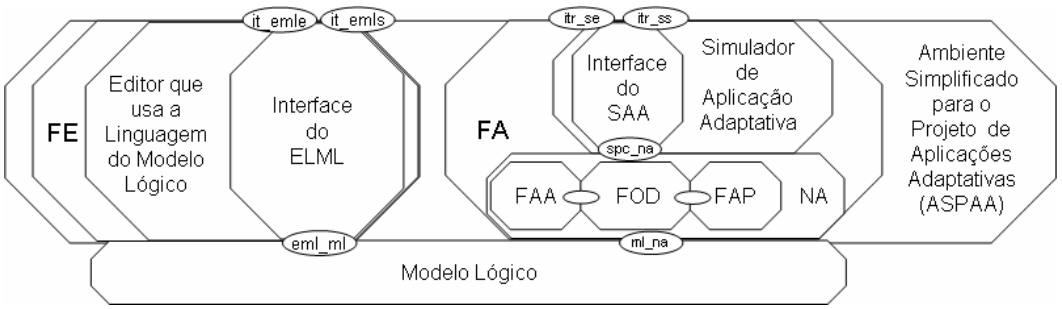


Figura 23. Ambiente Simplificado para o Projeto de Aplicações Adaptativas

4.2. Implementação da interface do ASPAA e do ELML

Um Editor que faça uso da Linguagem do Modelo Lógico (ELML) é uma ferramenta que permite a definição de dispositivos e a especificação de aplicações valendo-se dos dispositivos definidos e da linguagem do Modelo Lógico. Pedro (2005) define os principais requisitos para o referido editor:

- Permitir o acesso a dois tipos de usuários: o especialista em definição de dispositivos adaptativos e o projetista de aplicações. O especialista deverá possuir acesso completo à ferramenta tendo a possibilidade de realizar a definição de um novo dispositivo específico e também realizar a especificação de aplicações utilizando o dispositivo definido. Já o projetista, este poderá ter acesso somente aos recursos de especificação de aplicações com base nos dispositivos definidos por um especialista.
- Permitir a um especialista representar logicamente os elementos conceituais de um dispositivo específico e da camada adaptativa;
- Permitir a um projetista especificar suas aplicações por meio de um dispositivo adaptativo definido por um determinado especialista;
- Permitir a representação da especificação de aplicações em camadas, seguindo os conceitos apresentados para a definição do Modelo Lógico. Tal conceito é fundamental para permitir a separação dos conceitos relacionados à camada subjacente da camada adaptativa;
- Armazenar a representação dos elementos conceituais e de especificações de aplicações em um sistema de gerenciamento de banco de dados para que possam, posteriormente, ser recuperados pelas demais ferramentas que compõem o ASPAA e, futuramente, comporão o APAA.;
- Permitir que a representação de uma aplicação modelada com base em um dispositivo específico possa ser representada pela linguagem de outro dispositivo, uma vez que o Modelo Lógico é comum à definição dos elementos conceituais e de especificações para dispositivos dirigidos por regras.

Na Tabela 1, apresentada no trabalho realizado por Pedro (2005) são definidas as principais funcionalidades (eventos) para o ELML. As funcionalidades apresentadas fundamentam-se nos requisitos definidos anteriormente e no Modelo Lógico apresentado no Capítulo 6. Para facilitar a compreensão, os eventos foram agrupados e organizados seguindo a mesma estruturação para o modelo lógico, que se encontra em camadas. Na referida tabela são apresentadas três colunas: a primeira refere-se a uma codificação do evento; a segunda coluna representa a descrição do evento – estrutura de dados a qual ele tem acesso –; e, por fim, a terceira coluna índica qual o usuário que pode ter acesso ao referido evento, conforme os requisitos da aplicação. Um *E*, nesta coluna, indica que o evento somente será acessado por especialistas, enquanto um *P* indica que tanto os especialistas como os projetistas podem ter acesso ao referido evento.

Tabela 1. Lista de Eventos do Editor que usa a Linguagem do Modelo Lógico

Evento	Descrição do evento	Acesso
Eventos referentes às estruturas da Camada Dispositivos		
E1	Manter Usuário	E
E2	Manter Tipo de Componente	E
E3	Manter Tipo de Conexão	E
E4	Manter Tipo de Atributo	E
E5	Manter Dispositivo	E
Eventos referentes às estruturas da Camada Subjacente		
E6	Manter Projeto	P
E7	Manter Componente	P
E8	Manter Atributo de Componente	P
E9	Manter Conexão	P
E10	Manter Atributo de Conexão	P
E11	Manter Componente Inicial	P
Eventos referentes às estruturas da Camada Adaptativa		
E12	Manter Função Adaptativa	P
E13	Manter Ação Adaptativa	P
E14	Manter Atributo de Conexão de Ação Adaptativa	P
E15	Manter Atributo de Componente de Origem de Ação Adaptativa	P
E16	Manter Atributo de Componente de Destino de Ação Adaptativa	P
E17	Manter Função Adaptativa de Ação Adaptativa	P
E18	Manter Parâmetros de Ação Adaptativa	P
E19	Manter Acoplamento	P
Eventos referentes às estruturas da Camada de Memória		
E20	Manter Geradores	P
E21	Manter Variáveis	P
E22	Manter Cadeia de Entrada	P
E23	Manter Atributo de Cadeia de Entradas	P

As funcionalidades definidas para o Editor que usa a Linguagem do Modelo Lógico permitem ao especialista/projetista alimentar os dados necessários ao Modelo Lógico definido neste trabalho. A seguir, será descrito sucintamente o que faz cada uma das funcionalidades definidas para a referida ferramenta.

O evento Manter Usuário é responsável por permitir o cadastramento e a manutenção (consulta, alteração e exclusão) de usuários – administrador (especialista) e/ou projetista - que terão acesso à aplicação. Tal evento permite a um administrador cadastrar novos usuários para terem acesso ao ambiente de edição. A permissão de acesso aos novos usuários será conforme a sua categoria: se for do tipo especialista, terá acesso a todos os recursos da ferramenta; caso seja um projetista, terá acesso somente aos recursos de especificação de projetos utilizando os dispositivos já definidos.

Ao gerar um novo dispositivo, faz-se necessária a manutenção das mesmas características (especialista responsável, data de definição, etc.). Tal funcionalidade é fornecida pelo evento Manter Dispositivo. Ao definir um novo dispositivo, também se torna necessário definir a sua configuração: tipo de atributo, tipo de componente e tipo de conexão. Tais funcionalidades são representadas, respectivamente, pelos eventos: Manter Tipo de Atributo, Manter Tipo de Componente e Manter Tipo de Conexão.

Depois de definido um novo Dispositivo, este poderá ser utilizado por um projetista para realizar a especificação de suas aplicações. Inicialmente, um projetista deverá definir um novo projeto para armazenar a especificação de sua aplicação e, para tal, utilizará o evento Manter Projeto. Depois de criado um novo projeto, deve-se definir o comportamento subjacente da aplicação que vai ser projetada. Para realizar esta definição é preciso definir os seus componentes (Manter Componente) e respectivos atributos (Manter Atributo de Componente). Também devem ser definidas as suas conexões (Manter Conexão) e os seus respectivos atributos (Manter Atributo de Conexão). Ainda deve ser definido, ao ser realizada a especificação de uma aplicação, quais são seus componentes iniciais (Manter Componente Inicial).

Depois de realizada a especificação subjacente de uma aplicação, pode ser realizada a especificação, caso exista, da camada adaptativa. Para tal, inicialmente, devem ser definidas as funções adaptativas (Manter Função Adaptativa) e seus parâmetros de entrada (Manter Parâmetros).

Na seqüência, devem ser definidas as ações adaptativas pertencentes a cada função adaptativa (Manter Ação Adaptativa). Ao se definir uma ação adaptativa devem ser definidos os atributos dos componentes de origem (Manter Atributo de Componente Origem de Ação Adaptativa), os atributos dos componentes de destino (Manter Atributo de Componente de Destino de Ação Adaptativa) e os atributos de conexões (Manter Atributo de Conexão de Ação Adaptativa).

Depois de definidas as funções e as ações adaptativas, deve-se associar estas aos respectivos componentes e/ou conexões. Tal tarefa é desempenhada pelo evento Manter Acoplamento, que permite a associação de uma função adaptativa ao elemento componente/conexão a qual ela pertence. Destaque-se que este evento deve permitir a associação de uma ação a mais de um elemento, conforme a necessidade do projetista ao definir a especificação de uma aplicação.

Por fim, o Editor que usa a Linguagem do Modelo Lógico deve permitir a especificação dos elementos que serão utilizados na análise de uma aplicação — neste caso, para a ferramenta de simulação. Para tal, foram definidos os eventos Manter Cadeia de Entrada e Manter Atributos de Cadeia de Entrada para permitirem a um projetista informar os elementos que fazem parte dos estímulos submetidos a uma aplicação, quando da sua simulação.

4.2.1. Implementação física do ELML

Para a realização da implementação física do Editor que usa a Linguagem do Modelo Lógico foram necessárias a realização da definição de um banco de dados para armazenamento dos dados referentes ao Modelo Lógico e a codificação dos eventos que constituem o ELML.

Para a implementação do banco de dados, foi utilizada como base a metaestrutura definida para Modelo Lógico, descrita no Apêndice A. A materialização de tal estrutura foi realizada por meio do gerenciador de banco de dados Postgres (Postgres,2007). Postgres é um projeto de software livre coordenado pelo *PostgreSQL Global Development Group*. O banco de dados Postgres foi escolhido por ser uma ferramenta livre que pode ser utilizada sem ter a necessidade de custos financeiros como a compra de licenças para uso.

Utilizando-se da ferramenta pgAdmin, que permite a manutenção de bases de dados Postgres, foi criado um novo banco de dados - no Postgres - com o nome de *Modelo Lógico* e definidas as suas respectivas tabelas e relações com base na definição conceitual do referido modelo. Na Figura 24, é apresentada a interface da ferramenta pgAdmin, a qual contém, do lado esquerdo, uma árvore de banco de dados, suas tabelas e demais componentes. Do lado direito da mesma figura, pode ser observado a definição dos atributos de *Dispositivo* que representa, no modelo lógico, as informações de um dispositivo definido por um especialista.

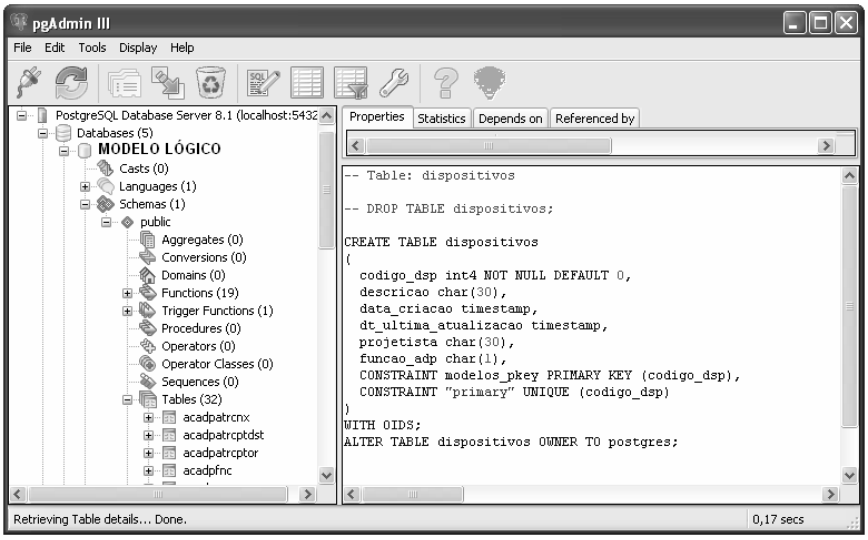


Figura 24. Interface da ferramenta para administração da base de dados do Modelo Lógico

O banco de dados Modelo Lógico implementado serve para o armazenamento das informações manipuladas pelo ELML e também é utilizada pela ferramenta de simulação definida para o ambiente ASPAA. Em uma segunda fase, o banco de dados definido poderá ser utilizado para o armazenamento das informações manipuladas pelo APAA completo.

Para a implementação do ELML foi utilizada a linguagem Java e o ambiente de desenvolvimento de aplicações Netbeans (NETBEANS, 2007). Netbeans é um ambiente de desenvolvimento - uma ferramenta para programadores, que permite a um projetista escrever, compilar, depurar e instalar programas. O ambiente é completamente escrito em Java, mas pode suportar qualquer linguagem de programação.

Utilizando-se do ambiente Netbeans, foi definido um novo projeto, *(ASPAA)* para armazenar as classes e formulários que constituem o ambiente ASPAA que é concebido pelas ferramentas Editor que usa a Linguagem do Modelo Lógico (ELML) e o Simulador de Aplicações Adaptativas (SAA). No projeto ASPAA, foram criadas

duas pastas: *interfaces* e *basedados*. Na pasta *interfaces* são armazenadas as classes responsáveis por representar as interfaces dos eventos definidos para o ELML e para o SAA. Na pasta *basedados* são armazenadas as classes responsáveis por realizar a conexão das tabelas - definidas no banco de dados Modelo Lógico – com as interfaces das ferramentas ELML e SAA. Na Figura 25, é apresentada a interface do ambiente Netbeans. Nesta figura pode ser observado, na janela projeto, a árvore de pastas para o projeto ASPAA, o qual contém as pastas *interfaces* e *basedados*. Também podem ser observados os menus de opções, barras de ferramentas e a definição da interface para a informação de *login* e *senha*. Tal interface será utilizada no ASPAA para permitir ao especialista/projetista informar os seus *login* e *senha*. Com base nestas informações, ocorre a validação do acesso e são definidas as suas permissões – especialista ou projetista - para o ambiente.

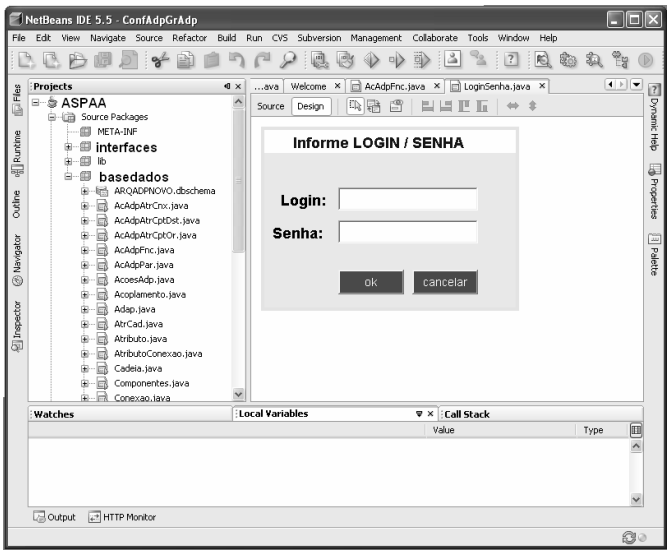


Figura 25. Interface do ambiente Netbeans - Projeto ASPAA

Depois, foi realizada a implementação da interface que apresenta um *Menu de Opções* e permite a escolha dos eventos referentes ao ELML ou chamada do SAA. Tal interface permite o acesso do especialista/projetista somente às funcionalidades a que este tenha acesso. Na Figura 26, nota-se que os menus de opções foram estruturados de forma a separar a definição de um formalismo e especificação subjacente numa determinada janela separada da especificação da camada adaptativa, conforme definido para os requisitos da aplicação. Nos menus *também* foram definidas as chamadas para a manutenção de usuários, execução do simulador *(SAA)* e tela de ajuda *(Sobre)*.

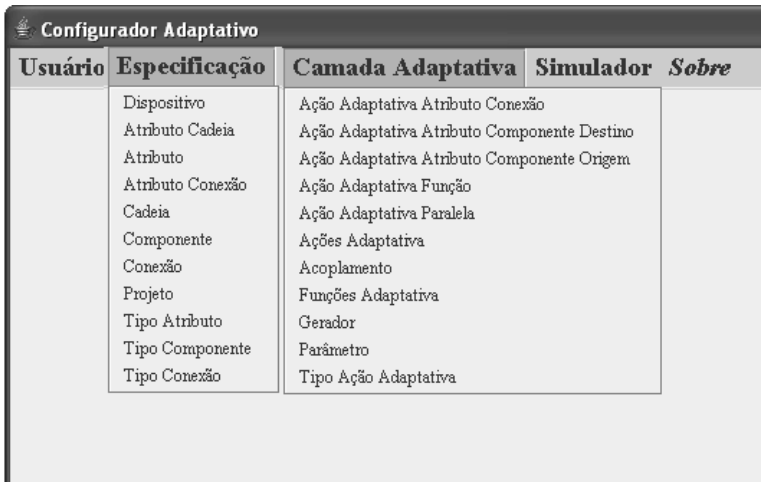


Figura 26. Interface do menu de opções ASPAA

Também foi realizada a implementação das interfaces para cada um dos eventos do ELML. Para as mesmas, foi definida uma interface bem simples, que se utiliza de caixas textos para a informação dos atributos relacionados a cada evento. Na Figura 27, é apresentada a interface para o evento responsável pela definição de *Tipo de Componente*. Observa-se que são disponibilizados botões para navegar (anterior, próximo, primeiro e último) e visualizar os objetos armazenados no modelo lógico, relacionados ao Tipo de Componente. São também disponibilizadas, por meio de botões, as funcionalidades para inserir, gravar, alterar e excluir objetos. Ainda outros dois botões permitem que sejam executadas as ações de cancelar a alteração de um objeto ou sair da referida interface.

Figura 27. Interface de operação do ELML

Maiores informações e detalhes sobre a implementação da interface do ASPAA e do ELML podem ser obtidos no trabalho apresentado em (PEDRO, 2005).

4.3. Simulador de aplicações adaptativas

Para a implementação do Simulador de Aplicações Adaptativas foi definido um *framework* que permite a instanciação do simulador para diversos dispositivos. Desta forma, utilizando-se do *framework*, podem ser desenvolvidos de forma rápida simuladores para os dispositivos adaptativos definidos e testá-los no projeto de aplicações. Para tal, foram definidos alguns requisitos conforme descritos abaixo:

- Disponibilizar uma interface para informar os valores da cadeia de entrada a serem testados em uma especificação e permitir que o projetista/especialista com ela interaja de forma a verificar se, ao executar sua especificação, esta atingirá os objetivos desejados;
- Permitir visualizar as definições dos elementos conceituais de um dispositivo;
- Permitir visualizar as especificações das aplicações que são simuladas usando o SAA;
- Permitir a visualização separada da especificação da camada subjacente da camada adaptativa;
- Permitir a visualização dos objetos referentes à Camada de Memória definida no Modelo Lógico;
- Permitir a execução da simulação de uma especificação passo a passo;
- Permitir a execução completa e direta (sem paradas) de uma especificação;
- Permitir visualizar as mudanças ocorridas numa especificação, quando da sua simulação (adição/remoção) de objetos da camada subjacente;
- Ter uma interface simples e intuitiva para o projetista;
- Servir como base para a implementação futura do simulador do APAA completo;
- Ter uma arquitetura que permita a um especialista mesmo sem muitos conhecimentos de programação obter um simulador para um novo dispositivo que ele venha a definir.

4.3.1. Implementação do SAA

Com base nos requisitos levantados e no Modelo Lógico, foi proposto um *framework* composto por uma interface que permita ao especialista/projetista visualizar os elementos conceituais de um dispositivo, de especificação de uma aplicação e de memória. Desta forma, ao acessar o ambiente de simulação, o projetista terá que informar, inicialmente, qual dispositivo será utilizado como base de simulação. Ao escolher um dispositivo, será apresentada a sua configuração e disponibilizada uma lista de aplicações modeladas que o utilizam como base. Ao ser escolhida uma aplicação, será apresentada a sua especificação (comportamento subjacente e adaptativo). Depois de carregada a especificação da aplicação desejada, o projetista deverá informar a sua cadeia de entrada e, na seqüência, iniciar a simulação. Depois de encerrada a simulação, o projetista pode reiniciar o processo ou escolher encerrar o simulador. Na Figura 28, é apresentada a hierarquia das atividades – diagrama de atividades UML – desempenhadas pelo SAA.

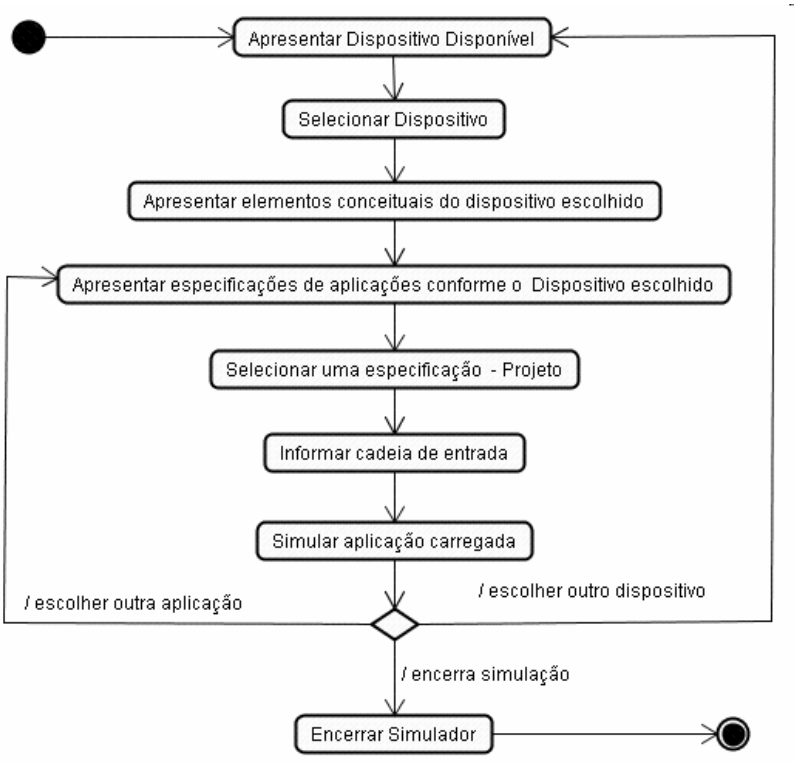


Figura 28. Diagrama de atividades do SAA

Fundamentado nas atividades a serem desempenhadas conforme apresentado acima e definidas nos requisitos para a aplicação, foi projetada uma

interface para o SAA também organizada em camadas, conforme definido para o modelo lógico e para o ELML. Neste contexto, é apresentado na Figura 29 o diagrama de classes referente à interface da aplicação. Nesse diagrama, a interface do SAA é representada por seis classes: *Interface de Dispositivo, Interface de Configuração de Dispositivo, Interface de Projeto, Interface de Comportamento da Camada Subjacente* e *Interface de Comportamento da Camada Adaptativa*.

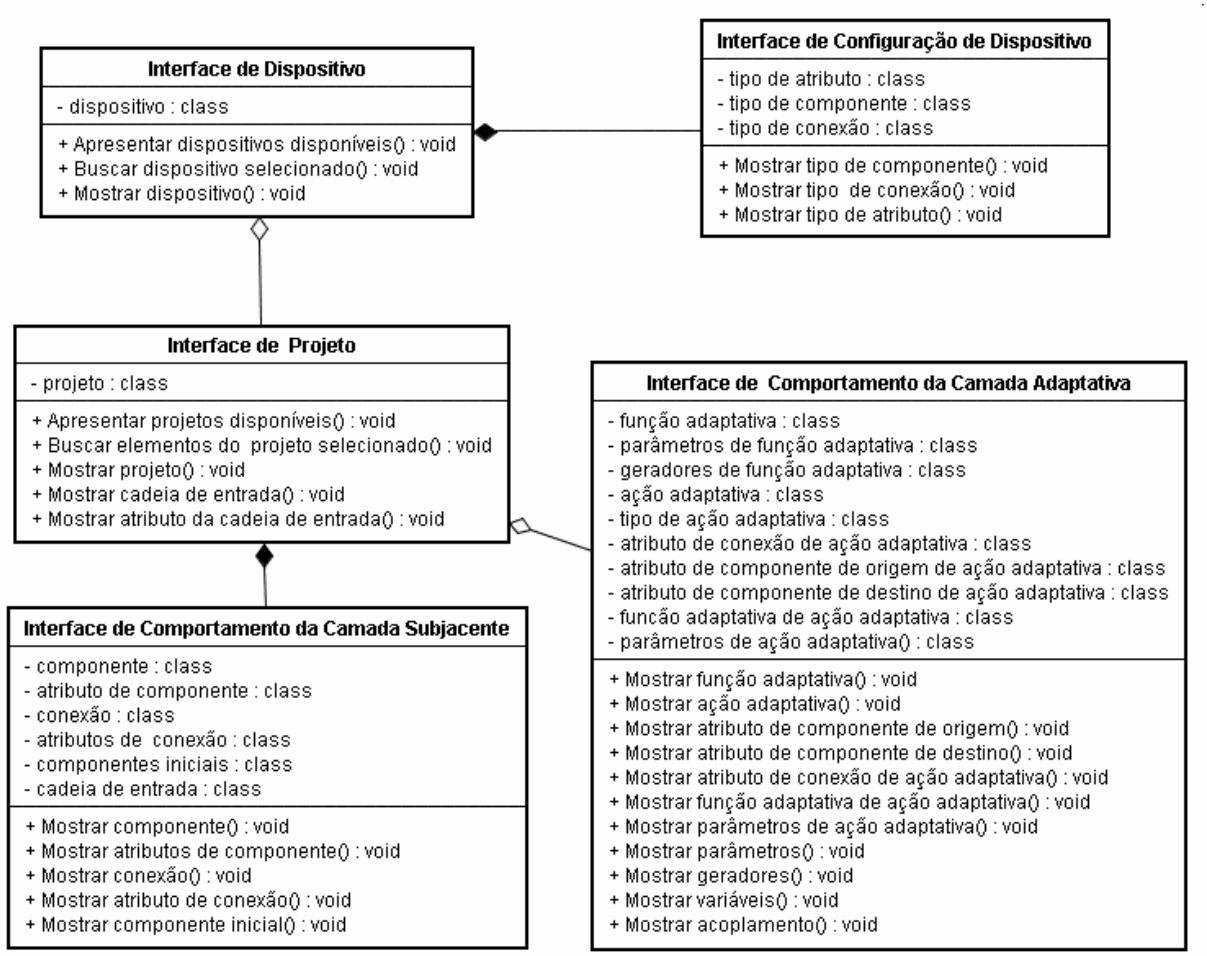


Figura 29. Diagrama de classes da Interface do SAA

A classe *Interface de Dispositivo* é responsável por gerenciar e apresentar as informações de um Dispositivo. Tal classe disponibiliza o método *Apresentar dispositivos disponíveis*, responsável por apresentar uma caixa *combo* que disponibiliza ao projetista uma lista de dispositivos. Ao ser escolhido um dos dispositivos na caixa *combo,* ocorre o método *Buscar dispositivo selecionado* que realiza uma busca no banco de dados *Modelo Lógico* e seleciona as informações de um dispositivo para serem apresentadas. Também é disponibilizado nesta classe o método *Mostrar dispositivo*. Tal método é responsável por apresentar as informações de um determinado dispositivo, recebe por parâmetros o campo chave,

pelo qual se deseja selecionar o dispositivo, neste caso o campo - *Código do Dispositivo* - e o seu respectivo valor.

A classe *Interface de Dispositivo* relaciona-se com a classe *Interface de Configuração de Dispositivo* que disponibiliza as funcionalidades para apresentação dos elementos conceituais de um dispositivo. Nesta classe são disponibilizados os seguintes métodos: *Mostrar tipo de componente*, *Mostrar tipo de conexão* e *Mostrar tipo de atributo*.

O método *Mostrar tipo de componente* tem por objetivo apresentar os tipos de componentes que representam os elementos conceituais de um dispositivo e, ao ser invocado tal método, devem ser passados por parâmetros o nome do campo chave que armazena o código de um dispositivo desejado e o seu respectivo valor para a busca a ser realizada. De forma semelhante ao método *Mostrar tipo de componente* é definido o método *Mostrar tipo de conexão*. Este método recebe os parâmetros referentes ao nome do campo que armazena as informações de um dispositivo, o código do dispositivo desejado, que realiza a busca das informações e as apresenta para o projetista. Os tipos de atributos que um dispositivo suporta também devem ser apresentados e, para tal, foi disponibilizado o método *Mostrar tipo de atributo* que recebe também os parâmetros que contêm o nome do campo que representa um dispositivo e o respectivo valor a ser consultado.

Depois de selecionados e apresentados os elementos conceituais de um dispositivo, devem ser apresentadas as informações de um projeto. Para tal, foi proposta a classe *Interface de Projeto*. Nesta classe é definido o método *Apresentar Projetos Disponíveis*, que realiza a apresentação – numa caixa *combo* – dos projetos de aplicações realizadas com base num dado dispositivo. Um projetista pode selecionar um dos projetos apresentados e, na seqüência, ver iniciar-se o método *Buscar elementos do projeto selecionado* que realiza a busca da descrição de um projeto e faz chamadas para o método *Mostrar projeto* que realiza a apresentação de algumas informações referentes *ao nome do projetista responsável, data de criação, etc*. Nesta classe também devem ser definidos métodos para visualizar a cadeia de entrada e os seus atributos. O método *Mostrar cadeia de entrada* permite apresentar as informações referentes aos estímulos da cadeia de entrada enquanto os seus atributos são visualizados pelo método *Mostrar atributos da cadeia de entrada*.

Após serem mostradas as informações de um projeto, deve ser disponibilizada a sua especificação (camada subjacente e adaptativa). Para tal, foram projetadas as classes *Interface de comportamento da camada subjacente* e *Interface de comportamento da camada adaptativa*. A classe *Interface de Comportamento da Camada Subjacente* é concebida por seus atributos e os seguintes métodos: *Mostrar componente, Mostrar atributo de componente, Mostrar conexão, Mostrar atributo de conexão* e *Mostrar componente inicial.*

Ao ser invocado o método *Mostrar Componente*, ele deve receber dois parâmetros para informar o nome do campo, *e.g. Código do Projeto* e o seu respectivo valor para ser selecionado. Quando forem apresentados os componentes, devem também ser apresentados os seus atributos. Tal funcionalidade é desempenhada pelo método *Mostrar atributo de componente.* Este método permite que sejam filtrados todos os atributos que fazem parte de algum componente de um projeto, isto quando o projetista informar o campo - *Código do Projeto* - e o seu valor. No caso do projetista desejar visualizar somente os atributos de um determinado componente, deverá informar como parâmetro o nome do campo - *Código do Componente* - e o valor do componente que deseje visualizar os atributos.

Para que seja possível a visualização da especificação de uma aplicação, devem ser apresentadas, além de seus componentes e atributos, as suas conexões. As conexões representam as relações existentes entre os componentes constituintes de uma especificação. O método *Mostrar conexão* é responsável por realizar tal funcionalidade. Ao ser acionado este método, devem ser informados os parâmetros que contenham o nome do campo que armazena o código do projeto e o valor do código do projeto desejado. Quando for apresentada uma conexão, também devem ser apresentados os seus atributos, caso estes existam. Para tal, foi estruturado o método *Mostrar atributo de conexão* que recebe, de forma semelhante à função *Mostrar atributo de componente*, os valores que contêm o nome do campo - *Código do Projeto* - e o seu respectivo valor, isto quando forem ser apresentados todos os atributos de todas as conexões que fazem parte de um dado projeto. Além desta possibilidade, o método pode receber os valores referentes ao campo - *Código da Conexão* - e o valor do código da conexão desejada, quando forem ser visualizados somente os atributos de uma determinada conexão. Também na classe *Interface de Comportamento da Camada Subjacente* é implementado o método

Mostrar componente inicial, que apresenta as informações dos componentes iniciais ativos para quando for iniciar uma simulação.

Caso seja realizada a especificação da camada adaptativa de uma aplicação, esta deverá ser visualizada no SAA. Para tal, foi estruturada a classe *Interface de Comportamento da Camada Adaptativa*, que agrega os seguintes métodos: *Mostrar função adaptativa, Mostrar ação adaptativa, Mostrar atributo de componente de origem, Mostrar atributo de componente de destino, Mostrar atributo de conexão de ação adaptativa, Mostrar função adaptativa de ação adaptativa, Mostrar parâmetros, Mostrar geradores, Mostrar variáveis* e *Mostrar acoplamento.*

O método *Mostrar função adaptativa* foi desenvolvido para filtrar e apresentar ao projetista as funções adaptativas que compõem a especificação de um projeto. Este método possui dois parâmetros que recebem os valores que permitem apresentar todas as funções adaptativas relacionadas a um projeto ou somente as funções adaptativas que estão relacionadas a um componente ou conexão. As funções adaptativas possuem um conjunto de ações adaptativas que são apresentadas pelo método *Mostrar ação adaptativa*. Tal método foi concebido para receber parâmetros que permitam a apresentação de todas as ações adaptativas de um projeto, ou somente as ações adaptativas que fazem parte de uma determinada função adaptativa.

Para cada uma das ações adaptativas devem ser apresentados os seus correspondentes atributos de origem, de destino e de suas conexões. Para executar tais funcionalidades foram definidos os respectivos métodos, *Mostrar atributo de componente de origem, Mostrar atributo de componente de destino* e *Mostrar atributo de conexão de ação adaptativa*. Tais métodos permitem a visualização de todos os atributos e de todas as ações pertencentes a um projeto ou somente dos atributos de uma ação adaptativa especifica.

As ações adaptativas podem descrever a definição de novas funções adaptativas para serem adicionadas. Tais funções e os seus parâmetros devem ser visualizados e, para isto, foram definidos os métodos *Mostrar função adaptativa de ação adaptativa* e *Mostrar parâmetros de ação adaptativa* que são responsáveis por apresentar tais funcionalidades.

As funções adaptativas possuem, além das ações adaptativas e seus elementos relacionados, os parâmetros, os geradores e as variáveis. Os métodos *Mostrar parâmetros*, *Mostrar geradores* e *Mostrar Variáveis* desempenham tais

funcionalidades e permitem a visualização de todos os elementos relacionados a um projeto ou a uma dada função adaptativa. A associação de uma função adaptativa a um componente ou conexão é apresentada pelo método *Mostrar Acoplamento*.

Os métodos responsáveis pela apresentação de informações possuem uma estrutura geral comum. Tais métodos têm, basicamente, a mesma funcionalidade, porém manipulam informações diferentes. A estrutura física destes métodos é padrão (comum), variando apenas a descrição dos campos de informação a serem apresentados, a origem dos dados a serem consultados e o local onde serão apresentados. A seguir, na Figura 30, é descrita a estrutura básica de um método *Mostrar* que é comum às diversas classes da interface do SAA.

```
<tipo retorno>  <descrição do método> ( <lista de parâmetros>) {
            <definição vetor de campos >
            <descrição campo visualizar 1>
            ...
            <descrição campo n>
        <objetobasedados>.manipulatable(<nometabela>, <nomecampo>, <valor>,
<vetor de campos>, <nomelocalapresentação> , "campoordemapresentação>);
    }
```

Figura 30. Descrição geral para um método Mostrar

Na estrutura definida para o método *Mostrar geral*, a sintaxe *<tipo retorno> <descrição do método> (<lista de parâmetros>)* representa a definição de um método específico. Nesta definição, *<tipo retorno>* representa qual o valor que o método deve retornar – neste caso, sem retorno de valor (void). A etiqueta *<nome do método>* indica o seu nome; e a etiqueta *<lista de parâmetros>* permite descrever os parâmetros associados a um método mostrar especifico. Na linha *<definição vetor de campos>*, é definido o nome do vetor responsável por descrever o nome dos campos que serão apresentados nas grades de visualização de valores, definida na interface do SAA. Após a definição do vetor de campos, deve ser definido cada um dos valores do referido vetor. Tal definição é representada pelas linhas *<definição vetor de campos 1..n>.* Depois de definidos o valor dos campos a serem apresentados, deve ser realizado a chamada do método de apresentação de dados definidos. A linha *<objetobasedados.manipulatable (<nomecampo>, <valor>, <vetordecampos>, <gradeapresentação>,<ordemapresentação>*) realiza a chamada do método *manipulatable* responsável por apresentar as informações para os projetistas. Nesta linha, a etiqueta *objetobasedados* indica o nome da tabela que contém os dados a serem visualizados; a etiqueta *<nomecampo>* indica o nome do

campo que vai ser filtrado; a etiqueta <*valor*> indica o valor que vai ser filtrado; a etiqueta <*vetordecampos*> recebe o nome do vetor de campos que contém a lista dos campos que serão visualizados e, por fim, as etiquetas <*gradeapresentação*> que indica o local – nome da grid de apresentação – em que os valores serão apresentados e<*ordemapresentação*> que recebe o nome do campo que indica por qual campo será ordenada as informações a serem apresentadas.

Depois de escolhida a aplicação a ser simulada e do projetista informar a cadeia de entrada, pode ser iniciada a simulação da aplicação. A simulação adaptativa deverá ser executada com base no algoritmo para execução do mecanismo adaptativo, desenvolvido por Neto (2001), já apresentado, anteriormente, no Capítulo 2 deste trabalho. Com base no referido algoritmo, é apresentado, na Figura 31, o diagrama de atividades que deverá ser desempenhado pelo simulador durante a simulação de uma dada aplicação.

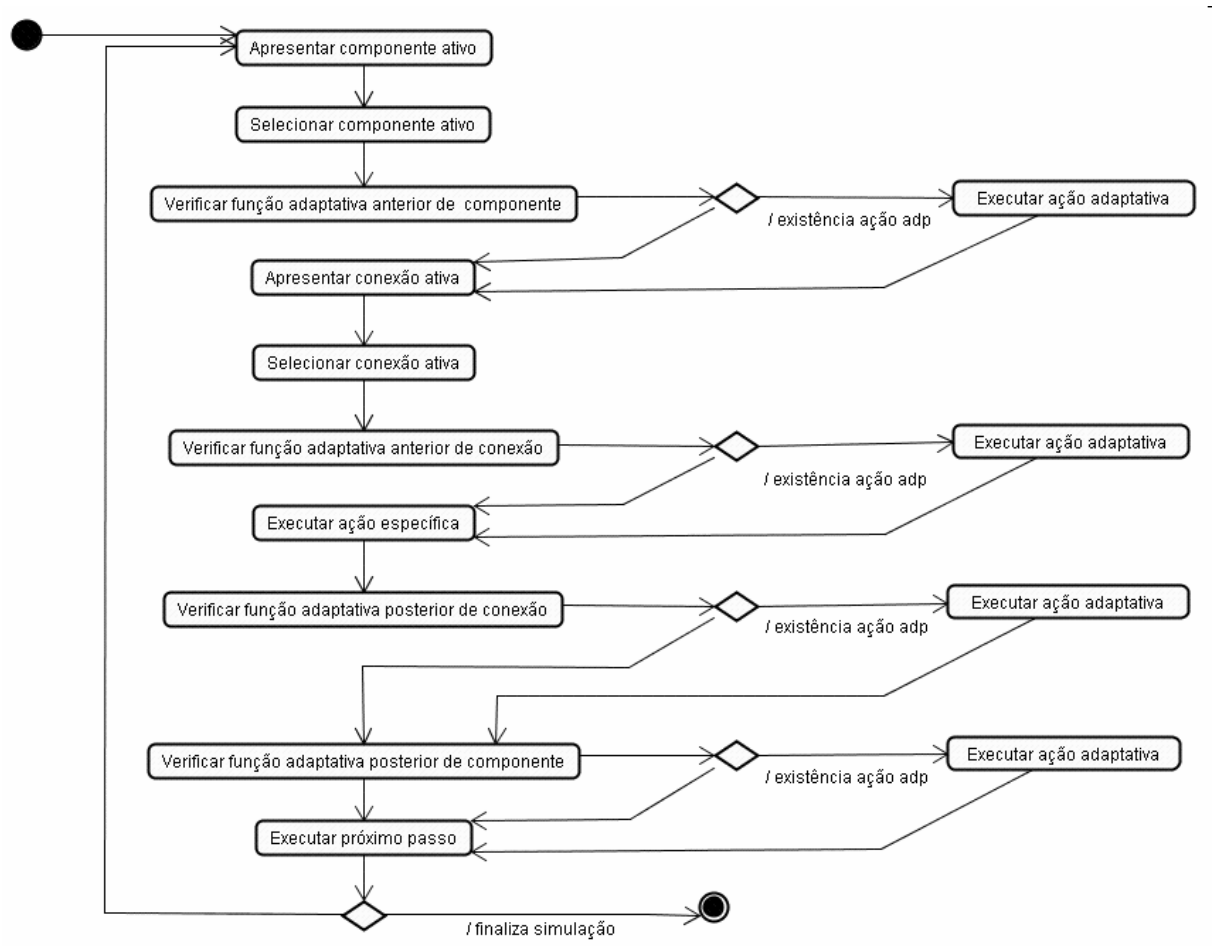


Figura 31. Diagrama de atividades desempenhadas pelo método de simulação do SAA

De acordo com o algoritmo proposto para a execução do mecanismo adaptativo e a estrutura definida para o SAA, inicialmente são apresentados ao

projetista quais são os componentes ativos. Para tal, utiliza-se o método *Apresentar componente ativo*. No caso da existência de mais de um componente ativo – indeterminismo – o projetista tem a possibilidade de escolher com qual componente deseja realizar a simulação utilizando-se dos recursos providos pelo método *Selecionar componente ativo*. Com base no componente ativo selecionado, chama-se o método *Verificar função adaptativa anterior de componente* que verifica a existência de tal função. No caso de existência de uma função adaptativa, será executado o método *Executar ação adaptativa*, responsável por executar as ações adaptativas definidas para uma dada função e, na seqüência, retorna-se ao curso normal de execução e são apresentadas as conexões ativas (método *Apresentar conexão ativa*). No caso de existência de mais de uma conexão ativa, o projetista tem a possibilidade de escolher qual conexão deseja simular por meio do método *Selecionar conexão ativa*.

Uma vez selecionada uma conexão ativa, é realizada a verificação da existência de funções adaptativas anteriores para conexão por meio do método *Verificar função adaptativa anterior de conexão*. Se existir uma função adaptativa, será executado o método *Executar ação adaptativa* e retornar-se-á ao processo de simulação.

Depois de selecionado um componente e uma conexão, executa-se a sua ação específica. Tal funcionalidade é provida pelo método *Executar ação específica*. Tal método é responsável por desempenhar as ações de um componente e/ou conexão, mover o ponteiro da cadeia de entrada e atualizar os componentes ativos de acordo com as ações realizadas. Na seqüência, verificam-se a existência de funções adaptativas posteriores de conexão (método *Verificar função adaptativa posterior de conexão*). Na existência de tais funções, é chamado o método *Executar ação adaptativa* e, em seguida, é verificado a existência de ações adaptativas posteriores de componentes por meio de o método *Verificar função adaptativa posterior de componente* e, por fim, é disponibilizado o método *Executar próximo passo* que permite ao projetista realizar mais um passo de simulação reiniciando o processo ou sair da simulação. No caso de fim da cadeia de entrada a simulação também será encerrada.

Na Figura 32, é apresentada a interface do SAA. Pode ser verificado que a referida interface foi organizada em camadas conforme definido nos requisitos, o que permite ao projetista visualizar todos os recursos necessários durante a

utilização. A interface utiliza-se do recurso de abas, que permite a visualização de mais de um agrupamento de dados em um mesmo painel de visualização (ex. Conexão e seus Atributos). Tal recurso pode ser visualizado na apresentação de componentes, conexões, ações adaptativas, etc.

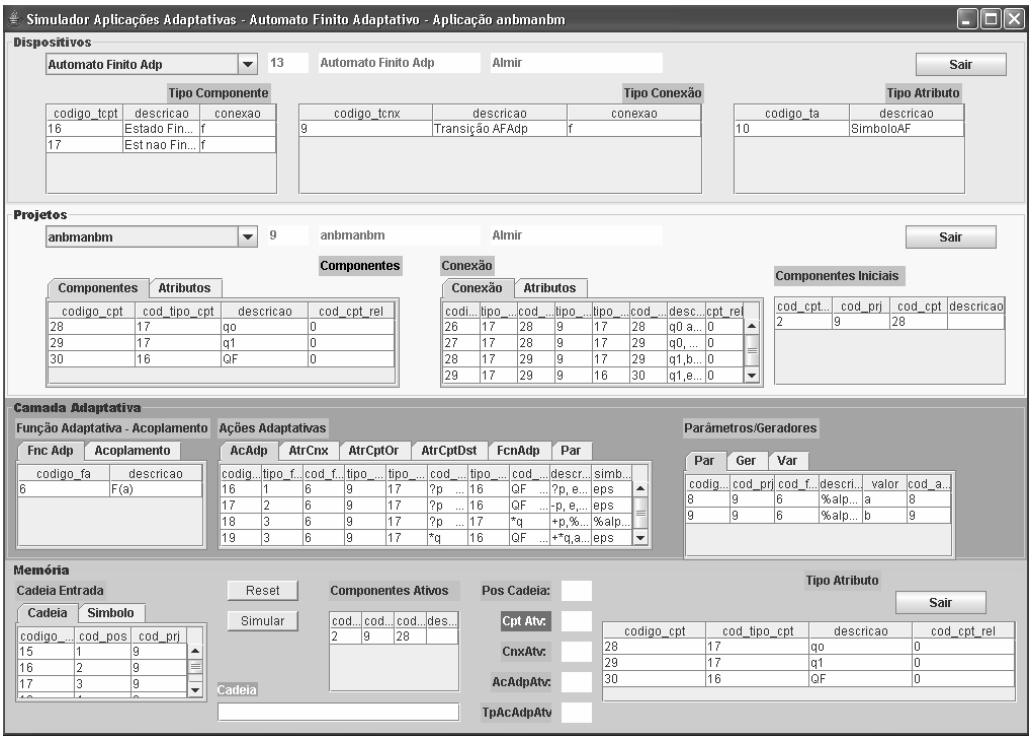


Figura 32. Interface do Simulador de Aplicação Adaptativa

Vale ressaltar que após serem implementados todos os métodos definidos para o SAA, ao definir um novo dispositivo, um especialista necessitará somente realizar a implementação do método *Executar ações específicas* e informar qual a forma de operação que o novo dispositivo deverá realizar. Verifica-se que tal estrutura permite a obtenção de uma ferramenta sem despender de um grande período de tempo para sua realização e, com isso, testar o novo dispositivo e aplicações desenvolvidas.

5. Experimentos realizados

Neste capítulo, inicialmente é apresentado um estudo de caso cujo desenvolvimento tem como base o dispositivo de Autômato de Estados Finitos (LEWIS; PAPADIMITRIOUS, 1998). Tal dispositivo fora escolhido por ser muito conhecido e relativamente simples. O segundo estudo de caso é fundamentado no dispositivo Rede de Petri (PETRI, 1962). Este dispositivo permite a representação de sistemas concorrentes e comunicantes. E por fim, o terceiro estudo de caso foi desenvolvido e teve como base o dispositivo *Interaction System Design Language (ISDL)* (QUARTEL, 1997). O dispositivo de ISDL foi escolhido por ser mais completo que os demais utilizados e por permitir, também, a especificação de sistemas distribuídos.

Inicialmente é realizada a definição do dispositivo adaptativo com base no dispositivo subjacente. Na seqüência, é realizado o mapeamento dos elementos conceituais do dispositivo adaptativo para o modelo lógico e por fim, é realizado o desenvolvimento de aplicações com base nos dispositivos obtidos, e finalmente, a utilização das ferramentas projetadas e automaticamente geradas neste trabalho.

5.1. Autômato de Estados Finitos Adaptativo

Um Autômato de Estados Finitos é um dispositivo simples e está intimamente relacionado com a classe das Linguagens Regulares, conforme denominado na hierarquia de Chomsky. Devido à sua simplicidade, tal dispositivo foi escolhido para ser representado utilizando a notação geral de dispositivos adaptativos. Em Lewis e Papadimitrious (1998) um Autômato de Estados Finitos (AEF) é constituído por um conjunto de estados e por uma função de transição que informa quais transições o mecanismo deve executar, a partir de um estímulo obtido na sua cadeia de entrada.

5.1.1. Formalização de Autômato de Estados Finitos

Com base nos conceitos apresentados em (NETO, 2001), a um AEF podem ser incorporados o conceito de dispositivo adaptativo para que ele possa oferecer os

recursos fornecidos por esses conceitos. Nesta formulação, um Autômato de Estados Finitos Adaptativo (AEF_{Adp}) é representado da seguinte forma AEF_{Adp} = (AEF_0, AM), onde, AEF_0 representa o comportamento não-adaptativo do dispositivo subjacente inicial e; AM representa o mecanismo adaptativo.

Formalmente, um Autômato de Estados Finitos pode ser representado por uma 6-upla, representada por AEF = (C, CR, Σ, c_0, A, NA) no qual:

- AEF é um mecanismo não-adaptativo de Autômato de Estados Finitos, cuja operação define o seu comportamento constituído por um Conjunto de Regras (CR);
- C é o conjunto de todas as possíveis configurações para um Autômato de Estados Finitos e $c_0 \in C$ determina a situação inicial do AEF.
- ε denota a cadeia vazia, que representa o elemento neutro do conjunto a que pertence, em relação à operação de concatenação;
- Σ é o conjunto finito de todos os possíveis eventos que formam as cadeias de entrada válidas para AEF. Este conjunto contém os estímulos que determinam a execução do AEF em cada passo de execução, com $\varepsilon \in \Sigma$, significando o símbolo de entrada vazio;
- $A \subseteq C$ é o conjunto de configurações de aceitação, ou seja, todas as possíveis ocorrências que são válidas para a aplicação modelada;
- F = C – A é o subconjunto das configurações que representam a não ocorrência de transições;
- $w = w_1 ... w_n$ é uma cadeia de entrada, onde $w_k \in \Sigma - \{\varepsilon\}$, k = 1,..., n, com $n \geq 0$;
- NA é representados por $\varepsilon \in$ NA uma vez que o AEF não gera saídas;
- CR é definido por um conjunto de transições. O estado do sistema é dado pelo estado corrente mais o valor da cadeia de entrada. A cada evento que ocorre, um símbolo é retirado da cadeia de entrada e uma transição é executada levando o sistema a um novo estado. O comportamento de um Autômato de Estados Finitos é definido formalmente por ($Q, \Sigma, c_0, F, \delta$) em que:
    - $Q \subseteq Q_\infty$ é um conjunto finito e não-vazio de estados;
    - Σ é o alfabeto de entrada também finito e não-vazio;
    - $c_0 \in Q$ é o estado inicial do autômato;

- $\delta \subseteq (Q \times(\Sigma \cup \{\varepsilon\}) \times Q_\infty)$ é a relação de transição, que inclui também transições em vazio (ε).

5.1.2. Formalização de Autômato de Estados Finitos Adaptativo

Um dispositivo torna-se adaptativo ao agregar uma camada adaptativa envolvendo o seu núcleo subjacente. Nesse contexto define-se um Autômato de Estados Finito Adaptativo (AEF_{Adp}) por meio do acréscimo de uma camada adaptativa envolvendo o seu núcleo subjacente. A Figura 33 ilustra a estrutura de um AEF_{Adp} constituída por uma camada adaptativa contendo funções anteriores e posteriores e suas respectivas ações.

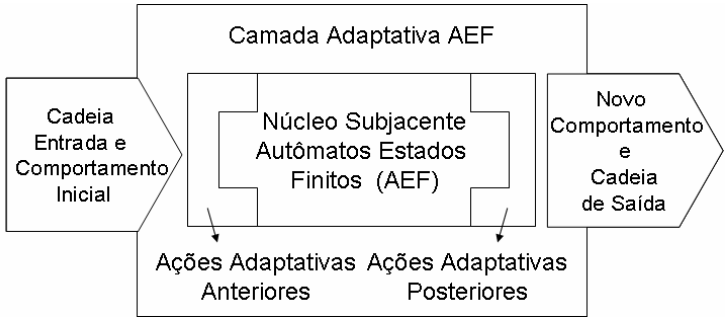


Figura 33. Dispositivo Autômato de Estados Finitos Adaptativo

Tomando-se como base a proposta apresentada por Neto (2001) para definição de dispositivos adaptativos, pode-se definir o dispositivo AEF_{Adp} iniciando sua operação na configuração inicial c_0 na forma $AEF_{Adp} = (C_0, AR_0, \Sigma, c_0, A, NA, BA, AA)$. No passo $k \geq 0$, um estímulo de entrada movimenta AEF_{Adp} para uma próxima configuração e, então prossegue sua operação no passo k+1, se e somente se, uma ação adaptativa não-vazia for executada. Assim, estando $AEF_{Adp,}$ em seu passo k, na forma $AEF_{Adp\ k} = (C_k, AR_k, \Sigma, c_k, A, NA, BA, AA)$, a execução de uma ação adaptativa não-vazia o leva à forma $AEF_{Adp\ k+1} = (C_{k+1}, AR_{k+1}, \Sigma, c_{k+1}, A, NA, BA, AA)$. Nesta formulação:

- $AEF_{Adp} = (AEF_0, AM)$ é um dispositivo Autômato de Estados Finitos Adaptativo formado por um dispositivo inicial subjacente Autômato de Estados Finitos (AEF_0) e um mecanismo adaptativo AM;
- AEF é o dispositivo de Autômato de Estados Finitos, cujo funcionamento foi descrito anteriormente, no passo k. AEF_0 é o

dispositivo subjacente inicial, e o conjunto CR_0 (representa o comportamento não-adaptativo inicial). Por definição, qualquer regra não-adaptativa em CR_k tem sua regra correspondente adaptativa em AR_k;

- C_k é o conjunto de todos os possíveis comportamentos de AEF no passo k, e $c_k \in C_k$ é o seu comportamento inicial no passo k. Para k=0, tem-se, respectivamente, C_0, o comportamento $c_0 \in C_0$, a configuração inicial de AEF_0 e de AEF_{Adp}.
- ε ("cadeia vazia") denota a ausência de qualquer outro elemento válido do conjunto correspondente;
- Σ é o conjunto (finito, fixo) de todos os possíveis eventos (inclusive o evento vazio ε) de que se compõem a cadeia de entrada em AEF_{Adp};
- $A \subseteq C$ é o subconjunto de comportamentos de aceitação de AEF;
- F = C - A é o conjunto das configurações de rejeição, ou seja, é o conjunto resultante da subtração das configurações de aceitação do conjunto de todas as possíveis configurações de comportamento;
- BA e AA são conjuntos de ações adaptativas. Ambas incluem a ação vazia ($\varepsilon \in BA \cap AA$)
- $w = w_1 w_2 ... w_n$, é a cadeia de entrada, onde $w_k \in \Sigma - \{\varepsilon\}$, k =1, ...,n com $n \geq 0$;
- NA, com $\varepsilon \in NA$, é um conjunto (finito, fixo) de todos os símbolos que podem ser gerados como saídas por AEF_{Adp}, em resposta à aplicação de regras adaptativas. Analogamente, ao que ocorre com os mecanismos não-adaptativos, a cadeia de saída assim obtida pode (se for conveniente) ser interpretada como uma sucessão de chamadas dos procedimentos correspondentes;
- AR_k é o conjunto das regras adaptativas que definem o comportamento adaptativo de AEF_{Adp} no passo K e é dado por uma relação $AEF_k \subseteq BA \times \Sigma \times C \times NA \times AA$;

Em particular, AR_0 define o comportamento inicial de AEF_{Adp}. Ações adaptativas modificam o comportamento adaptativo corrente AR_k de AEF_{Adp} para um novo comportamento AR_{k+1} adicionando e/ou eliminando locais e transições em AR_k. Regras $ar \in AR_k$ são da forma: (<ba>, $(Q \times (\Sigma \cup \{\varepsilon\} \times Q_\infty)$, <aa>), significando que,

em resposta a algum símbolo da cadeia de entrada $\sigma \in \Sigma$, ar inicialmente executa a ação adaptativa ba$\in$BA. Se a execução de ba eliminar ar de AR_k, a execução de ar é abortada; caso contrário, aplica-se a regra subjacente não-adaptativa ar = ($Q \times (\Sigma \cup \{\varepsilon\} \times Q_\infty)$, conforme descrito anteriormente; e finalmente, executa-se a ação adaptativa aa $\in$ AA.

- Define-se AR como o conjunto de todas as possíveis regras adaptativas para AEF_{Adp};
- Define-se CR como o conjunto de todos os possíveis comportamentos não-adaptativos para AEF_{Adp};
- AM $\subseteq$ BA$\times$CR$\times$AA, definido para um dispositivo adaptativo particular AEF_{Adp}, é um mecanismo adaptativo aplicado a todas as regras no passo k em $CR_k \subseteq$ CR. AM deve ser interpretado da mesma forma como se fosse aplicado a qualquer subdomínio $CR_{k.} \subseteq$ CR. Isso determinará um único par de ações adaptativas associadas a cada regra não-adaptativa.
- O conjunto $AR_k \subseteq$ AR pode ser obtido colecionando-se todas as regras adaptativas construídas pela associação de cada par de ações adaptativas às correspondentes regras de AEF em CR_k.

O comportamento não-adaptativo AEF_i, em cada passo de execução do mecanismo adaptativo, constitui o modelo AEF_{Adp} por meio de seus estados e corresondentes transições associados aos mesmos.

As ações adaptativas elementares AEF_{Adp} são definidas de forma semelhante às ações adaptativas elementares propostas em (NETO, 1993). As ações adaptativas AEF_{Adp} elementares assumem o seguinte formato: prefixo [padrão da produção], onde prefixo representa um dos três tipos de ações adaptativas elementares a serem executadas: “?” (ação de inspeção), “-“ (ação de eliminação) e “+” (ação de inserção), enquanto que o padrão da produção corresponde um nome da função que representa a ação AEF tradicional. Desta forma temos uma ação adaptativa elementar representada da seguinte forma [prefixo] ($Q, \Sigma, c_0, F, \delta$) onde ($Q, \Sigma, c_0, F, \delta$) faz parte do comportamento AEF definido para o dispositivo subjacente e o prefixo da ação adaptativa é definido conforme o tipo de uma ação adaptativa (prefixo), esta desempenha uma função que pode ter as seguintes características:

- ação adaptativa de inspeção: tem função de inspecionar/verificar se uma determinada regra AEF faz parte da especificação do comportamento da aplicação especificada;
- ação adaptativa de eliminação: é utilizada para eliminar regras AEF necessárias às modificações que devem ocorrer no comportamento da aplicação;
- ação adaptativa de inserção: serve para adicionar novas regras ao comportamento AEF.

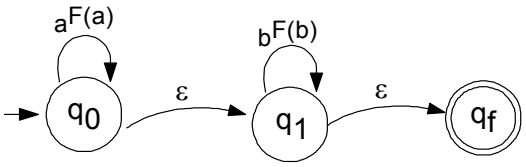
A operação do modelo AEF_{Adp} ocorre por meio da modificação do conjunto de regras de comportamento que definem uma aplicação. Ao serem executadas ações adaptativas novas regras (locais e transições) são adicionadas ou removidas.

Estando o AEF_{Adp} em sua situação inicial, recebe uma seqüência de eventos externos. As transições são executadas conforme essa seqüência, consumindo os estímulos da cadeia de entrada, repetindo-se o processo até o final da seqüência. Para cada evento recebido verifica-se qual ou quais são as produções a serem executadas. No que se refere à execução do AEF_{Adp}, se a ação adaptativa anterior B estiver presente, será executada em primeiro lugar, enquanto que se a função adaptativa posterior A estiver declarada, será executada por último, ou seja, após a execução de uma regra CR do dispositivo subjacente.

Na seção seguinte, será desenvolvida a modelagem de uma aplicação utilizando-se da estrutura definida para ilustrar a definição e o funcionamento do Autômato de Estados Finitos Adaptativo.

5.1.3. Modelagem da aplicação *Cadent* usando AEF_{Adp}

A primeira aplicação desenvolvida utiliza como base o dispositivo AEF_{Adp} e será fundamentado na linguagem $L = \{a^n b^m a^n b^m \mid n, m \in \text{naturais}\}$ que representa uma seqüência casada entrelaçada de *a´s* e *b´s*. A aplicação definida será denominada de aplicação *Cadeia Entrelaçada (CadEnt).* A Figura 34 ilustra a modelagem AEF_{Adp} do comportamento inicial da aplicação e de sua função adaptativa posterior *F()*.



a) Comportamento $a^n b^m a^n b^m$ inicial.

```
F(s)= { k*, :
        ? [?p, ε, qf],
        - [?p, ε, qf],
        + [?p, s, k*],
        + [ k*,ε, qf ],
      }
```

b) Função adaptativa F().

Figura 34. Comportamento Inicial aplicação *CadEnt*

Na Figura 34*a* pode ser observado que o comportamento inicial da aplicação *CadEnt* é constituído por três estados q_0, q_1 e q_f. O estado q_f representa o estado final de aceitação quando o AEF_{Adp} pára, reconhecendo toda a cadeia de entrada. Os estados q_0 e q_1 são responsáveis por reconhecer as seqüências iniciais de *a´s* e *b´s*. Cada um dos estados possui associada às suas transições, a função adaptativa F(s). Tal função tem por objetivo adaptar o referido autômato para que o mesmo possa reconhecer a linguagem definida. Nessa função *s* é um argumento que recebe por parâmetro o valor do símbolo reconhecido da cadeia de entrada, o * serve para representar os geradores que são utilizados pelas funções adaptativas para a geração de novos estados, (eg., k*) e o *?* serve para identificar as variáveis cujo valor é definido por uma ação adaptativa elementar de consulta e, posteriormente, utilizadas pelas demais ações adaptativas da respectiva função.

Tendo como exemplo, o reconhecimento da cadeia de entrada w=*aabbaabb* que pertence à linguagem definida. Inicialmente, ao ocorrer a transição *a* no estado q_0 e, consequentemente, a função *F(a)* e suas respectivas ações adaptativas é modificado o comportamento do autômato e uma nova transição (q_1, a, q_2) e o estado q_2 são adicionados, conforme ilustrado na Figura 35.

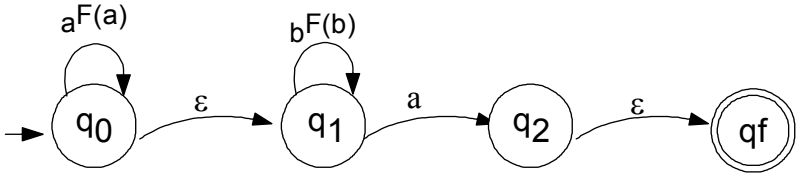


Figura 35. Comportamento aplicação *CadEnt* após reconhecer um símbolo *a*

Semelhante ao ocorrido no reconhecimento do primeiro símbolo *a*, ocorre no reconhecimento do segundo símbolo da cadeia de entrada (***aa****bbaabb*), são adicionados uma nova transição (q_2, a, q_3) e um novo estado q_3, necessários para o reconhecimento da linguagem da aplicação, conforme descrito na Figura 36.

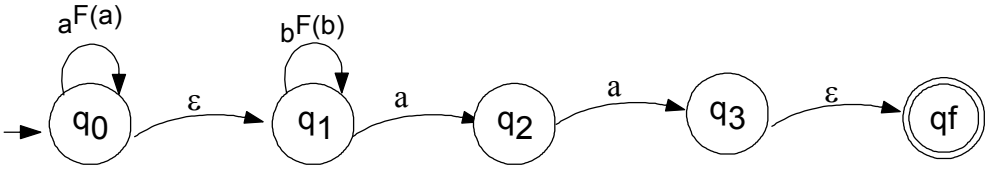


Figura 36. Comportamento aplicação *CadEnt* após reconhecer um símbolo

Continuando o reconhecimento, ocorre um *b* na cadeia de entrada (***aa<u>b</u>****baabb*) e o autômato evolui de forma não-determinística, por meio da transição ε, para o estado q_1, devido o autômato não possuir transições válidas no estado q_0. No estado q_1 o autômato consome o símbolo *b* e executa a função adaptativa $F(\alpha)$. Ao ocorrer tal função é adicionada uma nova transição (q_3, b, q_4) e um novo estado q_4, conforme apresentado na Figura 37.

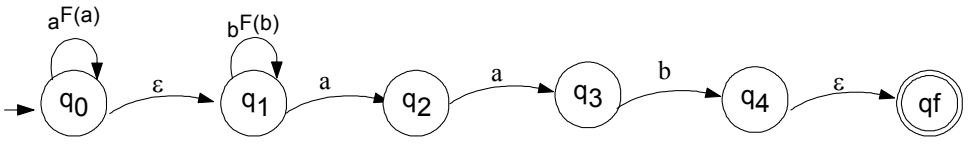


Figura 37. Comportamento aplicação *CadEnt* após reconhecer um símbolo *b*

Com base no comportamento e cadeia de entrada (***aab<u>b</u>****aabb*), é reconhecido um segundo símbolo *b* que ocasiona a chamada da função adaptativa $F(\alpha)$ e o comportamento da aplicação é modificado. Na Figura 38 é apresentado o comportamento modificado contendo uma nova transição (q_4, b, q_5) e um novo estado q_5.

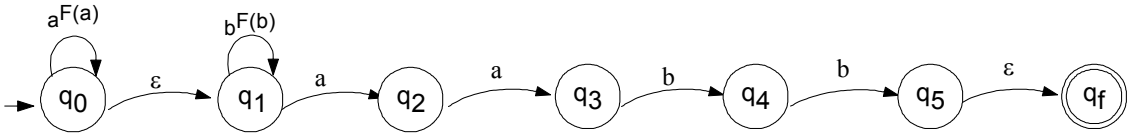


Figura 38. Comportamento aplicação *CadEnt* após reconhecer o segundo símbolo *b*

Como pode ser observado, na Figura 38 o comportamento da aplicação foi modificado e novos estados foram adicionados. O uso do AEF_{Adp} permitiu que na ocorrência dos demais símbolos da cadeia de entrada (***aabb<u>a</u>****abb*) estes possam ser reconhecidos por meio das transições e estados que foram adicionados no autômato. Por meio deste exemplo pode ser observado que o dispositivo AEF_{Adp} permitiu a modificação da estrutura do comportamento durante a execução e aquisição de informações necessárias para o reconhecimento da cadeia de entrada. Sem a utilização da estrutura adaptativa um Autômato de Estados Finitos não poderia reconhecer a linguagem base da aplicação *CadEnt*, uma vez que a mesma somente poderia ser reconhecida por um Autômato de Pilhas e não por um

Autômato de Estados Finitos que não possui pilhas. Para o reconhecimento dessa linguagem faz-se necessário o uso de uma pilha para armazenar os símbolos reconhecidos e compará-los com a cadeia de entrada conforme o procedimento de reconhecimento da cadeia de entrada definido para um Autômato de Pilhas.

5.1.1. Mapeamento do dispositivo AEF_{Adp} e da aplicação *CadEnt* para modelo lógico

Para realizar o mapeamento da especificação da aplicação *CadEnt*, faz-se necessário, inicialmente, o mapeamento do dispositivo AEF_{Adp} para o Modelo Lógico, descrita neste trabalho. Desta forma é definido um objeto na classe *Dispositivo*, ilustrado na Tabela 2 para armazenar as características do AEF_{Adp}.

Tabela 2. Mapeamento de objetos da classe Dispositivo para o dispositivo AEF_{Adp}

Atributo	Valor
código_dsp	2
descrição	Autômato de Estados Finitos Adaptativo (AEF_{Adp})
data_criação	15/05/2005
especialista	Almir Rogério Camolesi

Também é necessária, a definição dos elementos que constituem a estrutura básica do dispositivo que está sendo mapeado. Na Tabela 3 são descritos os possíveis tipos de componente permitidos para o dispositivo AEF_{Adp}.

Tabela 3. Mapeamento de objetos da classe Tipo de Componente para o dispositivo AEF_{Adp}

código_tcpt	código_dsp	descrição	conexão
15	5	Estado Final AEF_{Adp}	Não
16	5	Estado não Final AEF_{Adp}	Não

De forma semelhante, a definição de tipo de componente, devem ser mapeados os tipos de conexões que o dispositivo suporta. Na Tabela 4 é ilustrada a configuração do tipo de conexão disponível para o AEF_{Adp}.

Tabela 4. Mapeamento de objetos da classe Tipo de Conexão para o dispositivo AEF_{Adp}

código_tcnx	código_dsp	descrição
9	5	Transição AEF_{Adp}

Também devem ser mapeados os tipos de atributos que estão relacionados aos componentes e/ou conexões. Na Tabela 5, é apresentado o mapeamento de um objeto contendo a definição do atributo tipo *símbolo* que estará associado às transições AEF_{Adp}.

Tabela 5. Mapeamento de objetos da classe Tipo de Atributo para o dispositivo AEF_{Adp}

código_ta	código_dsp	descrição
9	5	Símbolo AEF_{Adp}

Depois de realizado o mapeamento dos elementos que fazem parte da definição estrutural do dispositivo AEF_{Adp}, deve ser desenvolvido o mapeamento da aplicação *CadEnt*. Ao realizar o mapeamento da aplicação, deve-se definir um novo projeto para representar a mesma, conforme descrito na Tabela 6.

Tabela 6. Mapeamento de objetos da classe Projeto – Aplicação *CadEnt*

Atributo	Valor
código_prj	10
código_dsp	5
descrição	Aplicação *CadEnt* - ($a^n b^m a^n b^m$)
data_criação	25/06/2005
data_última_atualização	22/07/2005
autor	Almir Rogério Camolesi

Após serem mapeadas as características elementares do projeto *CadEnt*, devem ser mapeados os componentes e as conexões que constituem a especificação subjacente da aplicação. Na Tabela 7 são descritos os objetos que representam os c*omponentes* que constituem o comportamento inicial da aplicação *CadEnt*.

Tabela 7. Mapeamento de objetos da classe Componente – Aplicação *CadEnt*

código_cpt	código_prj	código_tipo_cpt	descrição	código_cpt_rel
52	10	16	q_0	0
53	10	16	q_1	0
54	10	15	q_2	0

Conforme descrito anteriormente, o dispositivo de Autômato de Estados Finitos é um mecanismo simples e não possui atributos de componentes, sendo então necessário mapear os relacionamentos (conexões) existentes entre os componentes da aplicação. Na Tabela 8 é apresentado o mapeamento das conexões, do comportamento inicial, da aplicação *CadEnt*.

Tabela 8. Mapeamento de objetos da classe Conexão – Aplicação *CadEnt*

código_cnx	código_prj	tipo_cnx	código_org	código_dst	descrição
42	10	9	52	53	(q_0,ε,q_1)
43	10	9	53	54	(q_1,ε,q_f)
44	10	9	52	52	(q_0, a f(a), q0)
45	10	9	53	53	(q1, b f(b), q1)

Cada conexão AEF_{Adp} tem associado um atributo (*Símbolo*) que determina qual o valor da cadeia de entrada que é consumido a cada passo de execução. Na Tabela 9 é ilustrado o mapeamento dos atributos de conexões da aplicação *CadEnt*.

Tabela 9. Mapeamento de objetos da classe Atributo Conexão – Aplicação *CadEnt*

código_atr	código_cnx	tipo_atr	descrição	valor
10	42	9	Símbolo	ε
11	43	9	Símbolo	ε
12	44	9	Símbolo	a
13	45	9	Símbolo	b

Depois de realizado o mapeamento do comportamento não-adaptativo, faz-se necessário o mapeamento da especificação da camada adaptativa. Na Tabela 10 é apresentado o mapeamento da função adaptativa *F()* que compõem a especificação da aplicação *CadEnt*.

Tabela 10. Mapeamento de objetos da classe Função Adaptativa – Aplicação *CadEnt*

código_fa	projeto	descrição
7	10	F(α)

Para cada função adaptativa pertencente a uma determinada aplicação, devem ser mapeadas as suas correspondentes ações adaptativas. Na Tabela 11 é apresentado o mapeamento das ações adaptativas que constituem a função *F(α)* da aplicação *CadEnt*.

Tabela 11. Mapeamento de objetos da classe Ação Adaptativa – Aplicação *CadEnt*

código_aa	código_fun_adp	tipo_fun	tipo_cnx	tipo_org	código_org	tipo_dst	código_dst	descrição
26	7	1	9	16	p	16	qf	? [?p, ε, qf]
27	7	2	9	16	p	16	qf	- [?p, ε, qf]
28	7	3	9	16	p	16	q´	+ [?p, α, k*]
29	7	3	9	16	q´	16	qf	+ [k*, ε, qf]

Devido o AEF_{Adp} não possuir atributos de componentes, somente devem ser mapeados os atributos de conexões. Na Tabela 12 são apresentados os atributos de conexões da aplicação *CadEnt*.

Tabela 12. Mapeamento de objetos da classe Atributo de Conexão de Ação Adaptativa – Aplicação *CadEnt*

código_atr	código_acadp	tipo_atr	descrição	valor
14	26	9	Símbolo	ε
15	27	9	Símbolo	ε
16	28	9	Símbolo	α
17	29	9	Símbolo	ε

Depois de mapeadas as funções adaptativas e suas respectivas ações, deve ser realizada a associação das funções adaptativas com os respectivos elementos da camada subjacente. A referida associação é apresentada na Tabela 13.

Tabela 13. Mapeamento de objetos da classe Acoplamento – Aplicação *CadEnt*

código_acoplamento	tipo_acoplamento	código_cptcnx	fun_adp_ant	fun_adp_pos
9	X	44		7
10	X	45		7

Na Tabela 14 é apresentado o mapeamento dos parâmetros da função adaptativa *F(α)*. No exemplo apresentado é mapeado somente o parâmetro α que receberá durante a execução os valores *a* ou *b* conforme a chamada da função adaptativa. Se a função executada for invocada pela transição *(q0, aF(a), qo)*, o atributo receberá valor *a*, caso contrário, receberá valor *b* quando for executada a transição *(q1, bF(b), q1)*.

Tabela 14. Mapeamento de objetos da classe Parâmetros – Aplicação *CadEnt*

código_par	código_fun	descrição	valor
10	7	α	

E por fim, devem ser mapeados, na Classe *Gerador,* os valores referentes aos geradores que fazem parte das funções adaptativas. Na Tabela 15 é demonstrado o mapeamento do gerador *q´* utilizado na especificação da função adaptativa *F()*, da aplicação *CadEnt*.

Tabela 15. Mapeamento de objetos da classe Gerador – Aplicação *CadEnt*

código_ger	código_fun_adp	descrição	valor
3	7	q'	

5.2. Rede de Petri Adaptativa

O dispositivo não-adaptativo Rede de Petri (RP) desenvolvido por PETRI (1962) faz parte de um grupo de ferramentas matemáticas e gráficas que oferece um ambiente uniforme para a modelagem, análise formal e simulação de sistemas a eventos discretos, permitindo uma visualização simultânea da sua estrutura e comportamento. Mais especificamente, as RP modelam dois aspectos desses sistemas, eventos e condições, bem como, as relações entre eles. Segundo estas caracterizações, em cada estado do sistema verificam-se determinadas condições. Tais condições possibilitam a ocorrência de eventos que por sua vez podem ocasionar a mudança de estado do sistema.

Os elementos básicos que permitem a definição informal de uma Rede de Petri são polivalentes e em grande medida podem ser interpretados livremente. Informalmente uma RP constitui-se de três elementos básicos: local, transição e ficha.

O Local (representado por um círculo) pode ser interpretado como uma condição, um estado parcial, uma espera, um procedimento, um conjunto de

recursos, um estoque, uma posição geográfica num sistema de transporte, etc. Em geral, todo local tem um predicado associado, por exemplo, na Figura 39, os locais canal de comunicação livre e pacote de dados para transmitir.

A Transição (representada por barra ou retângulo) é associada a um evento que ocorre no sistema, como o evento iniciar a operação (transição *t* na Figura 39)

A Ficha (representada por um ponto num local) serve para indicar que a condição associada ao local é válida. Pode representar um objeto (recurso ou dados) numa certa posição geográfica (num determinado estado). Por exemplo, uma ficha no local canal de comunicação livre indica que o canal de comunicação está livre para a transmissão de dados (predicado verdadeiro). Se não tem fichas neste local, o predicado é falso, por conseguinte o canal não está livre. Se no local pacote de dados para transmitir houvesse cinco fichas, indicaria que existem cinco dados para serem transmitidos.

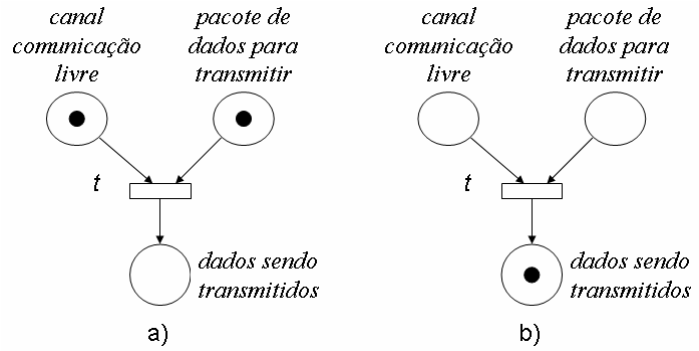


Figura 39. Representação gráfica de Rede de Petri

5.2.1. Formalização Rede de Petri

Utilizando-se dos conceitos apresentados por Neto (2001) pode-se incorporar aos conceitos do dispositivo de Rede de Petri o mecanismo adaptativo. Desta forma um RP_{Adp} = (RP_0, AM) é dito Rede de Petri Adaptativa (RP_{Adp}) sempre que, para toda operação $k \geq 0$, o mecanismo adaptativo levar o comportamento não-adaptativo RP_K até a execução de alguma ação adaptativa não-nula do passo k+1 e iniciar uma operação que altere o seu conjunto de regras.

Ao realizar o mapeamento de uma Rede de Petri para os conceitos de mecanismos adaptativos tem-se que inicialmente formalizar o dispositivo subjacente

(não-adaptativo). Desta forma tem-se uma Rede de Petri definida por um sêxtuplo, representado por RP = (C, CR, Σ, c_0, A, NA), onde:

- RP é um mecanismo não-adaptativo de Rede de Petri, cuja operação define seu comportamento constituído por um Conjunto de Regras CR;
- C é o conjunto de todas as possíveis configurações para uma Rede de Petri, e $c_0 \in$ C determina a situação inicial da RP.
- ε denota a cadeia vazia, que representa o elemento neutro do conjunto a que pertence, em relação à operação de concatenação;
- Σ é o conjunto finito de todos os possíveis eventos que formam as cadeias de entrada válidas para RP, contendo as fichas que permitem o disparo em cada passo de execução, com $\varepsilon \in \Sigma$;
- A $\subseteq$ C é o conjunto de configurações de aceitação, ou seja, todas as possíveis ocorrências de comportamentos que são válidos para a aplicação modelada;
- F = C - A é o subconjunto das configurações que representam a não ocorrência de transições, ou seja, comportamentos inválidos;
- w = w_1... w_n é uma cadeia de entrada, onde $w_k \in \Sigma - \{\varepsilon\}$, k = 1,..., n com n $\geq$ 0;
- NA é um conjunto finito com $\varepsilon \in$ NA sendo este o único símbolo pertencente ao conjunto devido o dispositivo de RP convencional não gerar cadeia de saída;
- CR é um comportamento definido por um conjunto de transições, locais e suas respectivas marcações. O estado do sistema é dado pela distribuição das fichas nos locais da Rede de Petri, cada local representando um estado parcial do sistema. A cada evento que ocorre no sistema, é associada uma transição no modelo RP. A ocorrência de um evento no sistema (que faz com que o mesmo passe do estado atual ao próximo estado) é representada, no modelo, pelo disparo da transição ao qual este está associado.

O disparo de uma transição consiste em dois passos: inicialmente as fichas são retiradas dos locais de entrada, indicando que esta condição não é mais verdadeira após a ocorrência do evento. Na seqüência, fichas são depositadas em

cada local de saída indicando que estas atividades estarão, após a ocorrência do evento, sendo executadas.

Em Cardoso e Vallete (1997) uma Rede de Petri é definida formalmente por um quádruplo CR = (P, T, Pre, Post) onde:

- P: {p1, p2,..., pn} é um conjunto de locais de dimensão n;
- T: {t1, t2,..., tm} é um conjunto de transições de dimensão m;
- Pre: P × T → N é a aplicação de entrada (locais procedentes ou incidência anterior), com N sendo o conjunto dos números naturais;
- Post: P × T → N é a aplicação de saída (locais seguintes ou incidência posterior)

5.2.2. Formalização Rede de Petri Adaptativa

Para que um dispositivo subjacente suporte às características adaptativas, faz-se necessário a adição de uma camada adaptativa para prover tais funcionalidades ao dispositivo definido. Neste contexto é apresentada a estrutura do dispositivo Redes de Petri Adaptativa definida por um núcleo subjacente RP envolvido por uma camada adaptativa. A Figura 40 ilustra a estrutura de RP_{Adp} constituída por uma camada adaptativa contendo funções e ações adaptativas anteriores e posteriores, que são responsáveis por realizar as mudanças no comportamento de especificações RP.

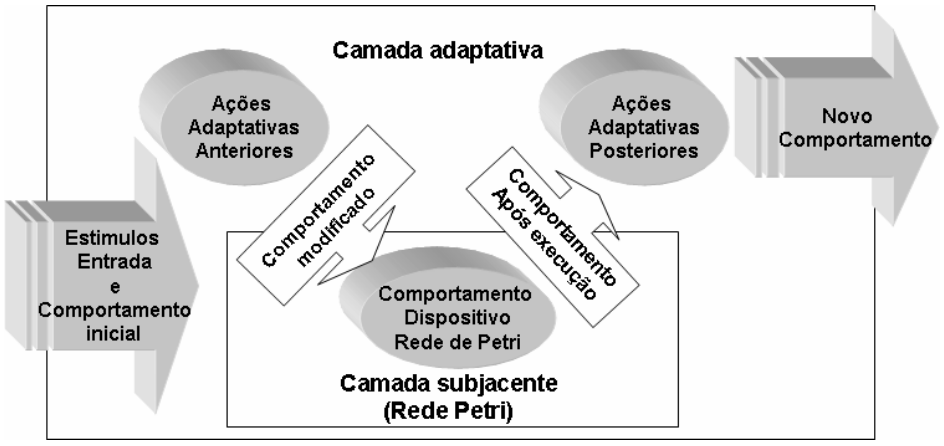


Figura 40. Dispositivo Rede de Petri Adaptativa

Tomando-se como base a proposta apresentada por NETO (2001) para definição de dispositivos adaptativos, pode-se definir o dispositivo RP_{Adp} iniciando

sua operação, na configuração inicial c_0, na forma RP_{Adp} = (C_0, AR_0, Σ, c_0, A, NA, BA, AA). No passo k≥0, um estímulo de entrada movimenta RP_{Adp} para uma próxima configuração e então prossegue sua operação no passo k+1 se, e somente se, uma ação adaptativa não-vazia for executada. Assim, estando RP_{Adp} em seu passo k, na forma RP_{Adpk}= (C_k, AR_k, Σ, c_k, A, NA, BA, AA), a execução de uma ação adaptativa não-vazia o leva à forma $RP_{Adp\ k+1}$ = (C_{k+1}, AR_{k+1}, Σ, c_{k+1}, A, NA, BA, AA). Nesta formulação:

- RP_{Adp} = (RP_0, AM) é um dispositivo Rede de Petri Adaptativa formado por um dispositivo inicial subjacente Rede de Petri (RP_0) e um mecanismo adaptativo AM;
- RP é o dispositivo de Rede de Petri, cujo funcionamento foi descrito anteriormente, no passo k. RP_0 é o dispositivo subjacente inicial, e o conjunto CR_0 (representa o comportamento não-adaptativo inicial). Por definição, qualquer regra não-adaptativa em CR_k, tem sua regra correspondente adaptativa em AR_k;
- C_k é o conjunto de todos os possíveis comportamentos de RP no passo k, e $c_k \in C_k$ é o seu comportamento inicial no passo k. Para k=0, tem-se, respectivamente, C_0, o comportamento $c_0 \in C_0$, a configuração inicial de RP_0 e de RP_{Adp}.
- ε (“cadeia vazia”) denota a ausência de qualquer outro elemento válido do conjunto correspondente;
- Σ é o conjunto (finito, fixo) de todos os possíveis eventos (inclusive o evento vazio ε) de que se compõem à cadeia de entrada em RP_{Adp};
- $A \subseteq C$ é o subconjunto de comportamentos de aceitação de RP;
- F = C - A é o conjunto das configurações de rejeição, ou seja, é o conjunto resultante da subtração das configurações de aceitação do conjunto de todas as possíveis configurações de comportamento;
- BA e AA são conjuntos de ações adaptativas. Ambas incluem a ação vazia ($\varepsilon \in BA \cap AA$)
- $w = w_1\ w_2\ ...\ w_n$, é a cadeia de entrada, onde $w_k \in \Sigma - \{\varepsilon\}$, k =1, ...,n com $n \geq 0$;
- NA, com $\varepsilon \in NA$, é um conjunto (finito, fixo) de todos os símbolos que podem ser gerados como saídas por RP_{Adp}, em resposta à aplicação

de regras adaptativas. Semelhante, ao que ocorre com os mecanismos não-adaptativos, a cadeia de saída assim obtida pode ser interpretada como uma chamada de procedimentos;

- AR_k é o conjunto das regras adaptativas que definem o comportamento adaptativo de RP_{Adp} no passo K e é dado por uma relação $AR_k \subseteq BA \times \Sigma \times C \times RP \times AA$;

Em particular, AR_0 define o comportamento inicial de RP_{Adp}. Ações adaptativas modificam o comportamento adaptativo corrente AR_k de RP_{Adp} para um novo comportamento AR_{k+1} adicionando e/ou eliminando locais e transições em AR_k. Regras ar $\in AR_k$ são da forma: (<ba>, (P, T, Pre, Post), .<aa>), significando que, em resposta a algum símbolo da cadeia de entrada $\sigma \in \Sigma$, ar inicialmente executa a ação adaptativa ba $\in$ BA. Se a execução de ba eliminar ar de AR_k, a execução de ar é abortada; caso contrário, aplica-se a regra subjacente não-adaptativa ar = (P, T, Pre, Post), conforme descrito anteriormente; e finalmente, executa-se a ação adaptativa aa$\in$AA.

- Define-se AR como o conjunto de todas as possíveis regras adaptativas para RP_{Adp};
- Define-se CR como o conjunto de todos os possíveis comportamentos não-adaptativos para RP_{Adp};
- AM $\subseteq$ BA $\times$ CR $\times$ AA, definido para um dispositivo adaptativo particular RP_{Adp}, é um mecanismo adaptativo aplicado a todas as regras no passo k em $CR_k \subseteq$ CR. AM deve ser interpretado da mesma forma como se fosse aplicado a qualquer subdomínio $CR_k \subseteq$ CR. Isso determinará um único par de ações adaptativas associadas a cada regra não-adaptativa.
- O conjunto $AR_k \subseteq$ AR pode ser obtido colecionando-se todas as regras adaptativas construídas pela associação de cada par de ações adaptativas às correspondentes regras de RP em CR_k.

O comportamento não-adaptativo RP_i, em cada passo de execução do mecanismo adaptativo, constitui o modelo RP_{Adp} por meio de seus locais, transições e fichas associadas aos mesmos.

As ações adaptativas elementares RP_{Adp} são definidas de forma semelhante às ações adaptativas proposta em (NETO, 1993). As ações adaptativas RP_{Adp}

elementares assumem o seguinte formato: prefixo [padrão da produção], onde prefixo representa um dos três tipos de ações adaptativas elementares a serem executadas: “?” (ação de inspeção), “-“ (ação de eliminação) e “+” (ação de inserção), enquanto que o padrão da produção corresponde um nome da função que representa a ação RP tradicional. Desta forma, temos uma ação adaptativa elementar representada da seguinte forma [prefixo] (P, T, Pre, Post) onde (P, T, Pre, Post) faz parte do comportamento RP definido para o dispositivo subjacente e o prefixo da ação adaptativa é definido conforme o tipo de uma ação adaptativa (prefixo), esta desempenha uma função que pode ter as seguintes características:

- ação adaptativa de inspeção: tem função de inspecionar/verificar se uma determinada regra RP faz parte da especificação do comportamento da aplicação especificada;
- ação adaptativa de eliminação: é utilizada para eliminar regras RP necessárias às modificações que devem ocorrer no comportamento da aplicação;
- ação adaptativa de inserção: serve para adicionar novas regras ao comportamento RP.

A operação do modelo RP_{Adp} ocorre por meio da modificação do conjunto de regras de comportamento que definem uma aplicação. Ao serem executadas ações adaptativas novas regras (locais e transições) são adicionadas ou removidas.

Estando o RP_{Adp} em sua situação inicial, recebe uma seqüência de eventos externos. As transições são executadas conforme essa seqüência, consumindo os eventos externos, repetindo-se o processo até o final da seqüência. Para cada evento recebido verifica-se qual ou quais são as produções a serem executadas. No que se refere à execução do RP_{Adp}, se a ação adaptativa anterior B estiver presente, será executada em primeiro lugar, enquanto que se a função adaptativa posterior A estiver declarada, será executada por último, ou seja, após a execução de uma regra CR do dispositivo subjacente.

5.2.3. Modelagem da aplicação AVSinc usando RP_{Adp}

O exemplo apresentado nesta seção utilizará o dispositivo de Rede de Petri Adaptativa (RP_{Adp}) para a modelagem de sistemas telemáticos. A Figura 41 ilustra

parte da modelagem Rede de Petri Adaptativa do comportamento que representa a aplicação que apresenta um áudio e um vídeo sincronizados *(AVSinc)*. Pode-se observar que no comportamento inicial um áudio e um vídeo estão armazenados em um servidor local e são representados, respectivamente, pelos locais, *AL* e *VL*. A existência de uma ficha no local *AL* e de uma ficha no local *VL* indica a possibilidade de ocorrência da transição *SINC*. A transição *SINC* possui associada à função adaptativa anterior que permite modificar o comportamento da aplicação em tempo de execução. Ao ser invocada a transição *SINC*, são executadas, anteriormente, suas funções adaptativas (representadas graficamente por meio de uma caixa texto ligada à transição).

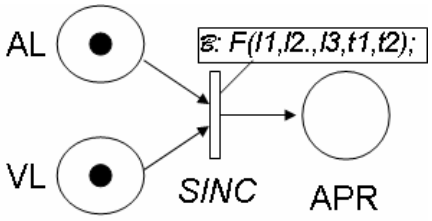


Figura 41. Comportamento inicial aplicação *AVSinc*

Com base no exemplo da Figura 42 supunha que o projetista deseja modificar a linguagem do áudio que está sendo reproduzido (eg., português para francês). Ao ser escolhido o novo tipo de áudio, verifica-se que este não está presente no servidor local e que ocorre a necessidade de transferência do referido áudio de um outro servidor e o armazenamento temporário do mesmo para posterior apresentação. Pode-se observar que deve ocorrer uma mudança no comportamento do sistema de forma a permitir o acesso ao novo áudio e posteriormente a sua apresentação. Tal modelagem pode ser representada pela definição da função adaptativa *F(..)* que executa suas ações adaptativas e modifica o referido comportamento. A função adaptativa *F(l1, l2, l3, t1, t2),* ilustrada pela Figura 42, recebe como parâmetros de entrada: o local *l1* (representa o áudio local que está sendo apresentado) que deverá ser consultado para verificar a existência de uma conexão entre *l1* e *t1* (responsável por representa a apresentação áudio e vídeo sincronizados). Caso existir está conexão, a mesma e o local *l1* deverão ser eliminados indicando a suspensão de apresentação de áudio e vídeo. Na seqüência, deverá ser adicionada a transição *t2* (responsável por modelar a ação de carregamento do novo áudio desejado). Tal função também recebe por parâmetros *l2* que corresponde à disponibilidade do áudio desejado em um servidor remoto e *l3*

que representa o áudio armazenado no servidor local após a sua transferência de um servidor remoto.

```
F (l1, l2,l3, t1, t2)
{
      ? l1 → t1
      - l1 → t1
      + l2 → t2
      + t2 → l3
      + l3 → t1
}
```

Figura 42. Função adaptativa F(..)

Ao ser executada a função adaptativa *F(...)* com seus respectivos parâmetros *F(AL, OP, BUF, SINC, DOWN)*; ocorre, inicialmente, a ação adaptativa de busca que verifica a existência da regra *AL → SINC*. Caso a regra seja encontrada, esta é eliminada e na seqüência são inseridas as regras *OP→ DOWN*, *DOWN→ BUF* e *BUF→ SINC*. A ocorrência de tais regras permite a mudança no comportamento da aplicação e, na seqüência, deve ser executada a transição *SINC*. A Figura 43 ilustra o comportamento da aplicação *AVSinc* após ter sido modificado por meio da execução da função adaptativa *F(...)*. Pode ser observado neste comportamento que foi removido o local *AL* e inserido os novos locais e transições conforme descritos anteriormente.

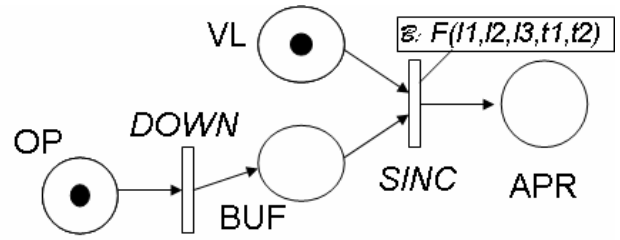


Figura 43. Comportamento aplicação *AVSinc* modificado

5.2.1. Mapeamento do dispositivo RP_{Adp} e da aplicação *AVSinc* para modelo lógico

Nesta seção será apresentado o mapeamento da estrutura de Rede de Petri Adaptativa e da aplicação *AVSinc* desenvolvida anteriormente.

Inicialmente são mapeados os conceitos teóricos definidos para o dispositivo de Rede de Petri Adaptativa para o Modelo Lógico. Na Tabela 16 é apresentado um objeto que representa as características do dispositivo RP_{Adp}.

Tabela 16. Mapeamento de objetos da classe Dispositivo – Dispositivo RP_{Adp}

Atributo	Valor
código_dsp	3
descrição	Rede de Petri Adaptativa (RP_{Adp})
data_criacão	15/01/2005
data_última_atualização	22/02/2005
especialista	Almir Rogério Camolesi

Na seqüência, são adicionados dois objetos na classe Tipo de Componentes para mapear os conceitos teóricos de Rede de Petri (local e transição). A Tabela 17 representa os elementos da respectiva classe, pode-se observar que o campo Conexão possui valor “False” devido a estrutura de Redes de Petri não permitir hierarquia de componentes.

Tabela 17. Mapeamento de objetos da classe Tipo de Componente – Dispositivo RP_{Adp}

código_tcpt	descrição	conexão	código_dsp
3	3	Local	False
4	3	Transição	False

Também, devem ser mapeados os conceitos teóricos referentes às conexões que interligam os diversos componentes da estrutura de uma RP_{Adp}. A Tabela 18 apresenta o mapeamento da “Ligação RP_{Adp}” que é o único elemento presente nesta estrutura. Também, pode ser observado na referida tabela que RP_{Adp} não possui hierarquia de conexões.

Tabela 18. Mapeamento de objetos da classe Tipo de Conexão – Dispositivo RP_{Adp}

código_tcnx	código_dsp	descrição
2	3	Ligação RP_{Adp}

Na Tabela 19 é apresentado o mapeamento do tipo de atributo relacionado à estrutura RP_{Adp}. Pode ser visto, na referida tabela que RP_{Adp} possuí apenas um atributo que indica a quantidade de fichas presentes em cada local de uma RP_{Adp}.

Tabela 19. Mapeamento de objetos da classe Tipo de Atributo – Dispositivo RP_{Adp}

código_ta	código_dsp	descrição
5	3	Fichas RP_{Adp}

Depois de mapeados, os conceitos referentes à configuração do dispositivo, deve ser realizado a especificação não-adaptativa da aplicação. Ao realizar a especificação da aplicação são gerados informações na estrutura lógica que permite representar a aplicação desenvolvida.

Inicialmente, é adicionado um objeto na classe *Projetos* que servirá para identificar o projeto que está sendo especificado, no caso, *Áudio* e Vídeo Sincronizados. A Tabela 20 apresenta o referido objeto e seus atributos.

Tabela 20. Mapeamento de objetos da classe Projeto – aplicação *AVSinc*

código_prj	código_dsp	descrição	autor	data_criação	data_última_atualização
9	3	AVSinc	Almir Camolesi	20/05/2006	25/05/2005

Utilizando-se dos tipos de componentes mapeados na fase anterior é realizada a especificação de componentes da aplicação *AVSinc*. A Tabela 21 ilustra o respectivo mapeamento e apresenta quatro novos objetos adicionados para representar, os locais *AL*, *VL* e *APR* e a transição *SINC*.

Tabela 21. Mapeamento de objetos da classe Componente – aplicação *AVSinc*

código_cpt	código_prj	código_tipo_cpt	descrição	código_cpt_rel
28	9	3	VL	
29	9	3	AL	
30	9	3	APR	
31	9	4	SINC	

Depois de mapeados os componentes, devem ser mapeados os seus respectivos atributos. A Tabela 22 apresenta a relação dos atributos, seus respectivos componentes e valores associados.

Tabela 22. Mapeamento de objetos da classe Atributo de Componente – aplicação *AVSinc*

código_atr	código_cpt	tipo_atr	descrição	valor
8	28	5	QFichas	1
9	29	5	QFichas	1
10	30	5	QFichas	0

Além dos componentes, devem ser mapeadas as conexões existentes entre os mesmos. A Tabela 23 ilustra as conexões e seus respectivos componentes de origem e destino.

Tabela 23. Mapeamento de objetos da classe Conexão – aplicação *AVSinc*

código_cnx	código_prj	tipo_cnx	código_org	código_dst	descrição
27	9	2	28	31	VL→SINC
28	9	2	29	31	AL→SINC
29	9	2	31	30	SINC → APR

Semelhante ao que ocorre para os componentes, deve ser mapeado os atributos das conexões. Neste caso, deve ser mapeada a quantidade de fichas (peso) que está relacionada com cada conexão (arcos), conforme a quantidade de fichas que são (pré ou pós) condições, para uma transição. Na Tabela 24 é

apresentada a quantidade de fichas relacionada com cada arco. Na aplicação *AVSinc* desenvolvida todos os arcos de pré e pós condições possuem peso 1.

Tabela 24. Mapeamento de objetos da classe Atributo de Conexão – aplicação *AVSinc*

código_atr	código_cnx	tipo_atr	descrição	valor
7	27	5	QFichas	1
8	28	5	QFichas	1
9	29	5	QFichas	1

Depois de realizada, a especificação do comportamento não-adaptativo, deve ser realizada a especificação das funções e ações adaptativas relacionadas a aplicação desenvolvida. Na Tabela 25 é inserido um objeto que contém informações da função adaptativa *F(...)* presente na aplicação *AVSinc*.

Tabela 25. Mapeamento de objetos da classe Função Adaptativa – aplicação *AVSinc*

código_fa	projeto	descrição
6	9	F(i1,i2,i3,t1,t2)

Para cada função adaptativa deve ser associado um conjunto de ações que determinam a estrutura da função e permitem que a mesma realize modificações no comportamento não-adaptativo da especificação desenvolvida. Na Tabela 26 é ilustrado as ações adaptativas referentes à função *F(...)*.

Tabela 26. Mapeamento de objetos da classe Ação Adaptativa – aplicação *AVSinc*

código_aa	código_fun_adp	tipo_fun	tipo_cnx	tipo_org	código_org	tipo_dst	código_dst	descricao
16	6	1	2	3	l1	4	t1	? l1→ t1
17	6	2	2	3	l1	4	t1	- l1 → t1
18	6	3	2	3	l2	4	t2	+ l2 → t2
19	6	3	2	4	t2	3	l2	+ t2 → l2
20	6	3	2	3	l3	4	t1	+ l3 → t1

Para que seja realizada a ligação da função adaptativa, faz-se necessário a associação desta com um componente ou conexão. Na tabela 27 é demonstrada a associação da função adaptativa anterior *F(..)* ao componente *Sinc*.

Tabela 27. Mapeamento de objetos da classe Acoplamento – aplicação *AVSinc*

código_acoplamento	tipo_acoplamento	código_cptcnx	fun_adp_ant	fun_adp_pos
5	C	31	6	

Ao ser invocada uma função adaptativa, deve ocorrer a passagem de seu parâmetros. A Tabela 28 representa a estrutura de armazenamento dos parâmetros da função *F(...)*. Os valores dos argumentos *I1*, *I2*, *I3*, *t1* e *t2* são armazenados nesta estrutura.

Tabela 28. Mapeamento de objetos da classe Parâmetros – aplicação *AVSinc*

código_par	código_fun	descrição	valor
10	6	l1	VL
11	6	l2	OP
12	6	l3	BUF
13	6	t1	SINC
14	6	t2	DOWN

Ao se definir uma ação adaptativa, deve-se também definir os atributos das conexões envolvidas. Neste caso, cada arco constituinte da aplicação *AVSinc* tem peso 1 conforme apresentado na Tabela 29.

Tabela 29. Mapeamento de objetos da classe Atributo de Conexão de Ação Adaptativa – aplicação *AVSinc*

código_atr	código_acadp	tipo_atr	descrição	valor
9	16	5	QFichas	1
10	17	5	QFichas	1
11	18	5	QFichas	1
12	19	5	QFichas	1
13	20	5	QFichas	1

Também devem ser definidos os atributos referentes aos componentes de origem. Tais atributos são estruturados conforme apresenta a Tabela 30.

Tabela 30. Mapeamento de objetos da classe Atributo de Componente de Origem de Ação Adaptativa – aplicação *AVSinc*

código_atr	código_acadp	tipo_atr	descrição	valor
4	16	5	QFichas	1
5	17	5	QFichas	1
6	18	5	QFichas	1
7	20	5	QFichas	0

Na Tabela 31 são relacionados os atributos relacionados aos componentes de destino de cada ação adaptativa.

Tabela 31. Mapeamento de objetos da classe Atributo de Componente de Destino de Ação Adaptativa – aplicação *AVSinc*

código_atr	código_acadp	tipo_atr	descrição	valor
4	16	5	QFichas	1
5	17	5	QFichas	1
6	18	5	QFichas	1
7	20	5	QFichas	0

5.3. ISDL Adaptativo

O *Interaction System Design Language* (*ISDL*) descrito por Quartel (1997) é um dispositivo que foi desenvolvido para a modelagem de sistemas distribuídos, processos de negócios, aplicações telemáticas e redes de comunicação. Tal

dispositivo fundamenta-se na busca de minimização de limitações existentes em métodos formais (SINDEREN et al., 1995). A modelagem de uma aplicação em ISDL consiste basicamente de dois modelos: um de entidades e um de comportamentos.

O modelo de entidades representa quais partes do sistema são consideradas e como elas são interconectadas. Dois conceitos são usados em um modelo de entidade: entidade e pontos de interação. Uma entidade representa um sistema (parte) que desempenha alguma função ou comportamento, como, por exemplo, um componente de software ou departamento de uma empresa. Um ponto de interação representa um mecanismo que possibilita a interação de uma entidade com outras entidades, como, por exemplo, um cliente de correio eletrônico.

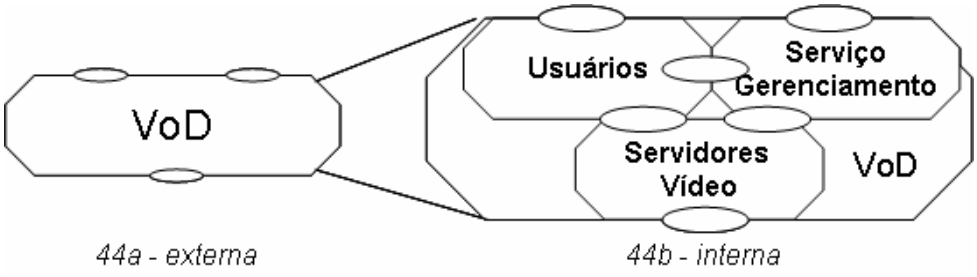


Figura 44. Modelo de entidades aplicação VoD

Pela perspectiva externa, um sistema é modelado por uma simples entidade, tendo um ou mais pontos de interação. A Figura 44a, por exemplo, representa um modelo de entidade de uma aplicação de Vídeo sob Demanda (VoD) elaborado por Camolesi (2000) no qual as interações ocorrem com o sistema por meio de três pontos de interação. Estes pontos de interação poderiam, por exemplo, representar as requisições de serviços realizadas por um projetista, as interações realizadas pelo administrador do sistema e os serviços de manutenção de mídias. Uma entidade é graficamente representada por um retângulo com os cantos entrecortados. Um ponto de interação é graficamente representado por uma elipse sobreposta às entidades que compartilham (e.g., via de comunicação) os pontos de interação.

Por outra, a perspectiva interna perspectiva interna, um sistema é modelado com uma composição de partes funcionais. Estas partes, por exemplo, podem representar subcomponentes ou departamentos de uma empresa. A perspectiva interna do VoD é apresentada na Figura 44b. Esta figura ilustra o VoD dividido em três partes: um serviço genérico do usuário, um provedor que notifica os usuários

sobre os serviços que estão disponíveis — autenticação, mídias disponíveis, etc. —, e um serviço provedor de mídias que são requisitadas pelos usuários.

O modelo de comportamento representa, literalmente, um comportamento, ou funcionalidade, de cada entidade em um correspondente modelo de entidade. Três conceitos são usados: ação, interação e relações de causalidade. Uma ação representa alguma atividade executada por uma entidade simples. Considere, por exemplo, a ação do serviço provedor de mídias que é responsável pela entrega das mídias aos usuários do serviço. Uma interação representa uma atividade executada por duas (ou mais) entidades. Uma contribuição de interação representa a participação de uma entidade individual em uma atividade comum. Considere também, ainda como exemplo, a interação entre o serviço de requisição de usuários e a apresentação de uma lista de filmes (mídias) disponíveis para apresentação.

5.3.1. Formalização ISDL

Fundamentado nos conceitos apresentados por Neto (2001), o conjunto de conceitos arquitetônicos do dispositivo ISDL constitui um mecanismo não-adaptativo dirigido por regras, cujo comportamento depende exclusivamente de um conjunto finito de regras que determinam, para cada possível configuração corrente do mecanismo, a sua próxima configuração. Desta forma, o dispositivo não-adaptativo ISDL é descrito como um sêxtuplo, e é representado formalmente por ISDL = (Ac, RC, Σ, c_0, A, NA), no qual:

- ISDL é um dispositivo não-adaptativo dirigido por regras cuja operação define um comportamento constituído de um conjunto de Relações de Causalidades (RC);
- Ac é o conjunto de todas as possíveis ocorrências de ações, e c0 $\in$ Ac determina o conjunto de ações do comportamento inicial. Os possíveis comportamentos alcançados são representados por intermédio da conjunção cruzada de suas execuções parciais. A conjunção cruzada e as execuções parciais são denotadas formalmente por $\otimes$ e eχ. Suponha a conjunção cruzada de dois comportamentos parciais EE1 e EE2, que constituem o comportamento EE. EE consiste de todas as possíveis execuções eχ, sendo eχ a conjunção de duas execuções

compatíveis $e\chi1$ e $e\chi2$, com $e\chi1 \in$ EE1 e $e\chi2 \in$ EE2. Conseqüentemente, o comportamento EE consiste em todas as possíveis construções alternativas que satisfaçam a um só tempo uma (sub)construção de EE1 e uma (sub)construção de EE2.

- ε denota a cadeia vazia, que representa o elemento neutro do conjunto ao qual pertence, em relação à operação de concatenação;
- Σ é o conjunto finito de todos os possíveis eventos que formam as cadeias de entrada válidas para ISDL, com $\varepsilon \in \Sigma$;
- A $\subseteq$ Ac é o conjunto de ocorrências de ações;
- F = Ac – A é o subconjunto das não ocorrências de ações;
- w = w_1... w_n é uma cadeia de entrada, onde $w_k \in \Sigma - \{\varepsilon\}$, k = 1,..., n com n$\geq$0;
- NA é um conjunto finito (com $\varepsilon \in$ NA) de todos os possíveis símbolos gerados como saída do mecanismo ISDL;
- RC é um comportamento definido por um conjunto de ações e interações de comportamentos e suas relações de causalidades. Um comportamento representa um conjunto de atividades que a entidade (conceito abstrato que modela um sistema, ou parte de um sistema) pode executar. O conceito de ação foi introduzido para representar uma atividade executada por um único sistema em um dado nível de abstração.

Atributos de informação, tempo e localização podem ser adicionados a uma (inter)ação para modelar, respectivamente, os resultados estabelecidos por alguma atividade, num determinado instante de tempo, em uma determinada localização. A ocorrência de uma (inter)ação representa o término com sucesso de uma atividade. Uma inter(ação) é atômica ao ocorrer e um resultado é estabelecido tornando-se disponível num determinado tempo e numa determinada localidade para todas as entidades envolvidas na atividade. Caso contrário, nenhum resultado é estabelecido e nenhuma atividade poderá se referir a qualquer resultado intermediário de uma atividade.

Uma ação é graficamente ilustrada por um círculo (ou elipse). Uma contribuição de interação é graficamente representada por um segmento de círculo (ou elipse), o que reflete o fato de que múltiplas entidades contribuem para a

interação. Os atributo de informação (ι), de tempo (τ) e de localização (λ) são representados dentro de caixas-texto anexadas às (inter)ações. Construções podem ser definidas nas possíveis saídas dos valores de ι, τ e λ .

No caso de uma interação, cada contribuição de interação define a construção da correspondente entidade, cada uma com os respectivos valores de ι, τ e λ devem satisfazer todas as construções das entidades envolvidas, caso contrário, a interação não ocorrerá. Também são permitidos múltiplos valores para algum atributo, e uma escolha não-determinística entre estes valores é assumida.

Desta forma, o conjunto de ações e interações que definem um comportamento ISDL é dado por uma relação de causalidade,

$$\langle\langle a, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma\rangle, \Gamma, \upsilon, \langle \textit{I-Refs}, \textit{T-Refs}, \textit{L-Refs}, \textit{ITLRefs}\rangle, \langle \textit{I-Caus}, \textit{T-Caus}, \textit{L-Caus}, \textit{ITL-Caus}\rangle\rangle$$

na qual:

- $a \in$ A é o nome da ação, que a identifica unicamente no sistema, e faz parte de A (conjunto de ocorrências de ações);
- $\mathrm{I}, \mathrm{T}, \Lambda$ são, respectivamente, os valores dos domínios de informação, tempo e localização da ação a;
- $\varsigma \subseteq \mathrm{I} \times \mathrm{T} \times \Lambda$ é a combinação dos valores dos domínios da ação a;
- $\Gamma \in$ CC é a condição de causalidade de a e faz parte de CC (conjunto de condições de causalidades disjuntivas);
- I-Refs, T-Refs, L-Refs, ITLRefs são, respectivamente, os conjuntos dos atributos de informação, tempo, localização e de combinação destes atributos;
- I-Caus, T-Caus, L-Caus, ITL-Caus são, respectivamente, os conjuntos de relações de causalidade de informação, de tempo, de localização e de combinação destes atributos;

Uma relação de causalidade é associada a cada (inter)ação, modelando a condição para esta inter(ação) ocorrer.

Três condições básicas para a ocorrência de uma ação a são identificadas: b → a; a ação b deve ocorrer antes da ação a; ¬ b → a; a ação b não deve ocorrer antes, nem simultaneamente com a ação a; e √ → a; a ação a é habilitada a ocorrer.

O operador e ($\wedge$) e o operador ou ($\vee$) podem ser usados para modelar condições de causalidades complexas. Por exemplo, b $\vee$ ¬ c $\rightarrow$ a representa que a

ação a ocorre depois da ação b ter ocorrido e a ação c ainda não ter ocorrido. Além disso, um atributo de probabilidade para cada condição pode ser usado para representar a probabilidade da inter(ação) acontecer quando a condição for satisfeita.

O conceito de relação de causalidade permite a modelagem de muitas diferentes relações entre ações. Isto é conveniente para representar estes relacionamentos diretamente, em vez da composição de relações de causalidade de ações individuais. A Figura 45 ilustra graficamente algumas relações comuns entre duas ou três ações. Os operadores $\wedge$ e $\vee$ são graficamente representados, respectivamente, pelos símbolos ■ e □. A condição inicial $\surd$ é representada por uma seta com nenhuma ação anterior ligada a ela.

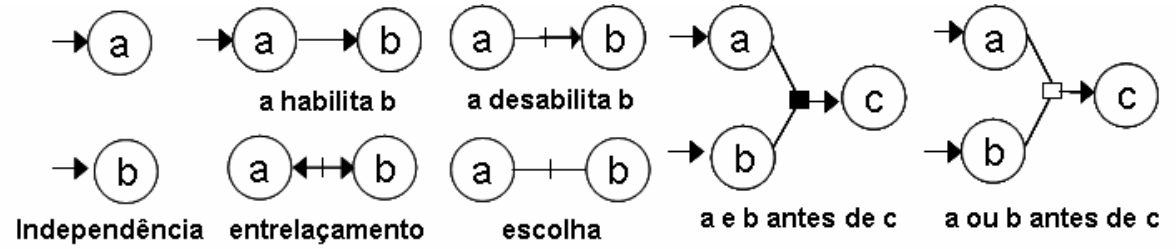


Figura 45. Algumas ações e relações de causalidade ISDL

O dispositivo ISDL suporta duas técnicas ortogonais para estruturação de comportamentos por meio da composição de subcomportamentos menores e mais simples: estrutura orientada à causalidade e estrutura orientada à restrição. A estrutura de comportamentos orientada à causalidade é fundamentada na decomposição de uma relação de causalidade em uma construção sintática, que permite definir uma ação e suas relações de causalidades em diferentes subcomportamentos.

A estrutura de comportamentos orientada à restrição é fundamentada na decomposição de uma ação em uma ou mais interações, o que permite que uma ação seja definida para um comportamento como uma composição de interações de subcomportamentos. Esta técnica pode ser usada para decompor condições e construções complexas na execução de uma ação em simples subcondições e subconstruções que são assumidas para contribuições de interações definidas em subcomportamentos distintos. Além disso, a estrutura orientada à restrição é necessária para estruturar um comportamento em subcomportamentos de forma que o mesmo possa ser atribuído a diferentes entidades, desde que as entidades possam se comunicar por meio de interações.

A Figura 46 mostra a modelagem do acesso, da requisição de serviços e do gerenciamento de requisições em que ambas às técnicas de estruturação de comportamento são usadas. Por meio da estrutura orientada à restrição é modelado o acesso do usuário ao sistema, o envio e recebimento de requisições e a execução dos serviços solicitados. Na referida modelagem, o usuário realiza a solicitação de acesso ao sistema por meio da interação *Login*. Após receber e validar a requisição de acesso, o sistema habilita o usuário a fazer requisições ao subcomportamento *Gerenciamento Serviços* que os disponibiliza aos usuários (ex: Fim Serviço). Utilizando-se da estruturação de comportamento orientada à causalidade, foram modeladas as interações existentes entre os Usuários e os subcomportamentos *Serviço Autenticação* e *Serviço Gerenciamento*. enquanto a estruturação orientada à causalidade foi utilizada para representar a habilitação da interação Requisição do subcomportamento *Serviço Gerenciamento* por meio de pontos de entrada e saída.

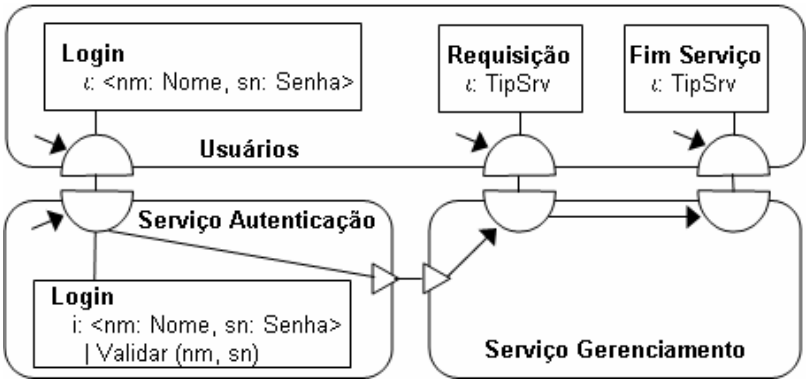


Figura 46. Exemplo de estruturação de comportamento ISDL

Em (Testbed, 2005) uma ferramenta gráfica para AMBER, um dialeto de ISDL constituído para reengenharia de processos, foi desenvolvido e o Projeto Friends (FRIENDS, 2005) tem adaptado e estendido esta ferramenta para suportar a modelagem de componentes de software e suas composições para sistemas telemáticos.

5.3.2. Formalização $ISDL_{Adp}$

A arquitetura básica do dispositivo *Adaptive Insteraction System Design Language* (*$ISDL_{Adp}$*) (CAMOLESI; NETO, 2004a), representada na Figura 47, constitui-se de um núcleo não-adaptativo ISDL (QUARTEL, 1997), envolvido por uma camada de ações adaptativas que têm por finalidade a realização de mudanças

das características do sistema representado. A camada adaptativa é formada por um conjunto de ações adaptativas anteriores e posteriores, semelhantes às ações definidas para os Autômatos Adaptativos (NETO, 1993). As ações adaptativas têm por objetivo realizar mudanças no comportamento da especificação ISDL.

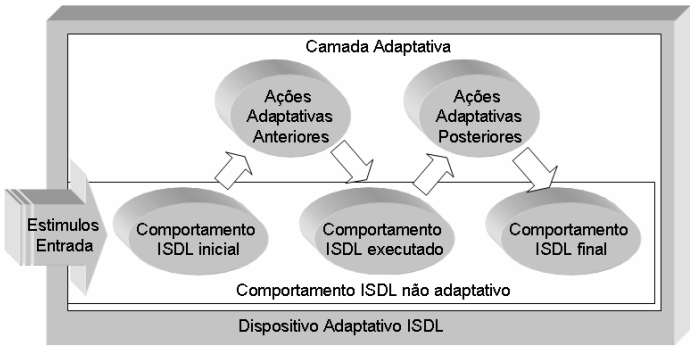


Figura 47. Estrutura dispositivo $ISDL_{Adp}$

Tomando-se como base o mecanismo adaptativo geral apresentado por Neto (2001), pode-se definir o dispositivo $ISDL_{Adp}$ iniciando sua operação na configuração inicial c0 na forma $ISDL_{Adp0}$= (C_0, AR_0, Σ, c_0, A, NA, BA, AA). No passo k$\geq$0, um estímulo de entrada movimenta $ISDL_{Adp}$ para uma próxima configuração e então prossegue sua operação no passo k+1 se, e somente se, uma ação adaptativa não-vazia for executada. Assim, estando $ISDL_{Adp}$ em seu passo k, na forma $ISDL_{Adpk}$ = (C_k, ARC_k, Σ, ck, A, NA, BA, AA), a execução de uma ação adaptativa não-vazia o leva à forma $ISDL_{Adpk+1}$ = (C_{k+1}, AR_{k+1}, Σ, c_{k+1}, A, NA, BA, AA). Nesta formulação:

- $ISDL_{Adp}$ = ($ISDL_0$, AM) é um mecanismo adaptativo formado por um mecanismo inicial subjacente $ISDL_0$ e um mecanismo adaptativo AM;
- ISDL é o dispositivo subjacente não-adaptativo, cujo funcionamento descrito na seção 2, no passo k. $ISDL_0$, é o mecanismo subjacente, definido no conjunto inicial RC_0 (conjunto de ações ou relações de causalidades representando um comportamento não-adaptativo). Por definição, qualquer ação ou relação de causalidade não-adaptativa em RC_k tem sua regra correspondente adaptativa em ARC_k;
- C_k é o conjunto de todos os possíveis comportamentos de ISDL no passo k, e $c_k \in C_k$ é o seu comportamento inicial no passo k. Para k=0, tem-se, respectivamente, C_0, o conjunto inicial de ações e relações de causalidade válidas e $c_0 \cdot C_0$, a configuração inicial de $ISDL_0$ e de ISDL.
- ε (“cadeia vazia”) denota a ausência de qualquer outro elemento válido do conjunto correspondente;

- Σ é o conjunto (finito, fixo) de todos os possíveis eventos (inclusive o evento vazio ε) de que se compõem a cadeia de entrada AD;
- A $\subseteq$ C é o subconjunto de ações e relações de causalidade de aceitação do comportamento ISDL;
- F = C - A é o conjunto das configurações de rejeição, ou seja, é o conjunto resultante da subtração das configurações de aceitação do conjunto de todas as possíveis configurações de comportamento;
- BA e AA são conjuntos de ações adaptativas. Ambas incluem a ação vazia ($\varepsilon \in BA \cap AA$)
- $w = w_1 w_2 ... w_n$, é a cadeia de entrada, na qual $w_k \in \Sigma - \{\varepsilon\}$, k=1, ...,n com $n \geq 0$;
- NA, com $\varepsilon \in NA$, é um conjunto (finito, fixo) de todos os símbolos que podem ser gerados como saídas por $ISDL_{Adp}$, em resposta à aplicação de regras (ações e relações) adaptativas. Analogamente ao que ocorre com os mecanismos não-adaptativos, a cadeia de saída assim obtida pode (se for conveniente) ser interpretada como uma sucessão de chamadas dos procedimentos correspondentes;
- ARC_k é o conjunto das regras adaptativas que definem o comportamento adaptativo de $ISDL_{Adp}$ no passo K e é dado por uma relação $ARC_k \subseteq BA \times C \times \Sigma \times C \times ISDL \times AA$.

Em particular, ARC_0 define o comportamento inicial de $ISDL_{Adp}$. Ações adaptativas modificam o comportamento adaptativo corrente ARC_k de $ISDL_{Adp}$ para um novo comportamento ARC_{k+1} adicionando e/ou eliminando inter(ações) e relações de causalidade em ARC_k. Regras $arc \in ARC_k$ são da forma:

<<ba>, ⟨⟨*a*, I, T, Λ, ς⟩, Γ, υ, ⟨*I-Refs*, *T-Refs*, *L-Refs*, *ITLRefs* ⟩, ⟨*I-Caus*, *T-Caus*, *L-Caus*, *ITL-Caus*⟩⟩, <aa>>

significando que, em resposta a algum símbolo da cadeia de entrada $\sigma \in \Sigma$, inicialmente arc executa a ação adaptativa $ba \in BA$. Se a execução de ba eliminar uma regra arc de ARC_k, a execução de arc será abortada; caso contrário, aplica-se a regra subjacente não-adaptativa:

- arc = ⟨⟨a, I, T, Λ, ς⟩, Γ, υ, ⟨I-Refs, T-Refs, L-Refs, ITLRefs ⟩, ⟨I-Caus, T-Caus, L-Caus, ITL-Caus⟩⟩ $\in$ RC_k, conforme descrito anteriormente; e finalmente, executa-se a ação adaptativa $aa \in AA$.

- Define-se ARC como o conjunto de todas as possíveis regras adaptativas para $ISDL_{Adp}$;
- Define-se RC como o conjunto de todas as possíveis ações e relações de causalidade não-adaptativas para $ISDL_{Adp}$;
- AM $\subseteq$ BA×RC×AA, definido para um particular mecanismo adaptativo $ISDL_{Adp}$, é um mecanismo adaptativo aplicado a todas as regras no passo k em $RC_k \subseteq$ RC. AM deve ser interpretado da mesma forma como se fosse aplicada a qualquer subdomínio $RC_k \subseteq$ RC. Isso determinará um único par de ações adaptativas associadas a cada regra não-adaptativa.
- O conjunto $ARC_k \subseteq$ARC pode ser obtido colecionando-se todas as regras adaptativas construídas pela associação de cada par de ações adaptativas às correspondentes regras não-adaptativas em RC_k.

O comportamento não-adaptativo $ISDL_i$, em cada passo de execução do mecanismo adaptativo, constitui o dispositivo $ISDL_{Adp}$ por meio de suas ações, suas condições de causalidade básica, das relações entre ações. Uma ação $ISDL_{Adp}$ pode ser representada por:

$$\langle\langle a, I, T, \Lambda, \varsigma\rangle : \mathcal{A} \rightarrow \langle\langle a`, \Gamma`, T`, \Lambda`, \varsigma`\rangle, \Gamma`, \upsilon`, \langle \text{I-Refs`, T-Refs`, L-Refs`, ITLRefs`}\rangle, \langle \text{I-Caus`, T-Caus`, L-Caus`, ITL-Caus`}\rangle\rangle : \mathcal{B}$$

A composição $\langle\langle a`, \Gamma`, T`, \Lambda`, \varsigma`\rangle, \Gamma`, \upsilon`, \langle$I-Refs`, T-Refs`, L-Refs`, ITLRefs`$\rangle$, $\langle$I-Caus`, T-Caus`, L-Caus`, ITL-Caus`$\rangle\rangle$ representa a situação do comportamento C_i antes da execução de uma ação adaptativa e é determinado de acordo com a definição anterior para comportamentos ISDL.

A Figura 48 ilustra a representação de uma ação $ISDL_{Adp}$ genérica e suas funções adaptativas anteriores (A) e posteriores (B). Graficamente, estas funções são representadas em caixas textos acopladas às partes superior e inferior das ações ISDL tradicionais.

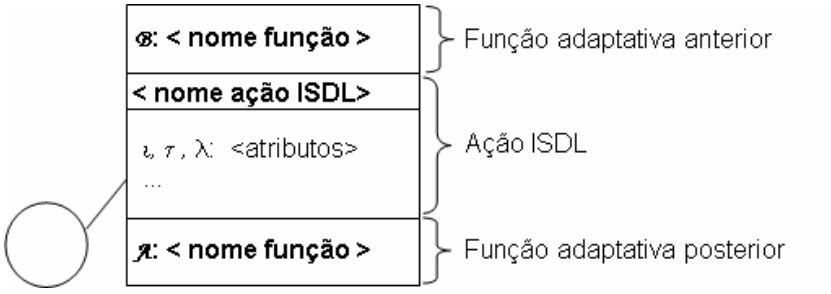


Figura 48. Representação de ações adaptativas $ISDL_{Adp}$

Visando diminuir a complexidade e facilitar a leitura das especificações $ISDL_{Adp}$ geradas, as funções adaptativas são especificadas separadamente das ações $ISDL_{Adp}$. Textualmente, foram adicionadas às especificações ISDL tradicionais duas seções: Before e After, que servem, respectivamente, para modelar as funções adaptativas anteriores e posteriores. No interior de cada seção são declaradas as funções adaptativas associadas a cada inter(ação). A Figura 49 ilustra a estrutura básica definida para as seções adaptativas anteriores e posteriores e suas respectivas funções e ações adaptativas.

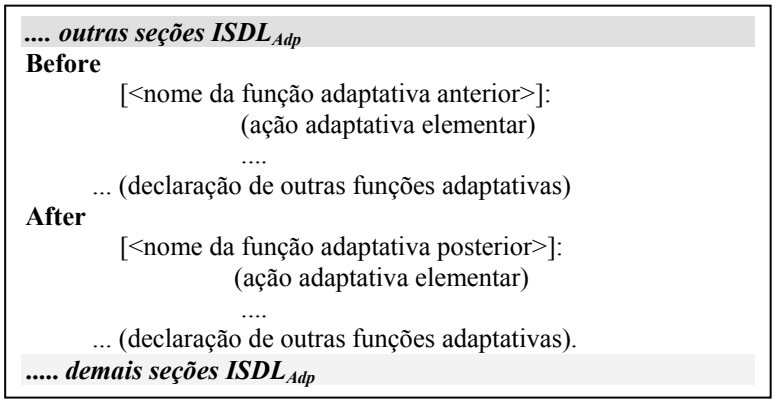


Figura 49. Representação de funções e ações elementares adaptativas $ISDL_{Adp}$

As ações adaptativas elementares $ISDL_{Adp}$ são definidas de forma semelhante às ações adaptativas propostas para Autômatos Adaptativos (NETO, 1993). As ações adaptativas $ISDL_{Adp}$ elementares assumem o seguinte formato: prefixo [padrão da produção], no qual prefixo representa um dos três tipos de ações adaptativas elementares a serem executadas: "?" (ação de inspeção), "-" (ação de eliminação) e "+" (ação de inserção), enquanto o padrão da produção corresponde a um nome da função que representa a ação ISDL tradicional. Desta forma, temos uma ação adaptativa elementar representada da seguinte forma:

[πρεφιξο] ⟨⟨a, I,T, Λ, ς⟩, Γ, υ, ⟨I-Refs`, T-Refs`, L-Refs`, ITLRefs` ⟩, ⟨I-Caus`, T-Caus`, L-Caus`, ITL-Caus`⟩⟩

em que: ⟨⟨a, I,T, Λ, ς⟩, Γ, υ, ⟨I-Refs`, T-Refs`, L-Refs`, ITLRefs` ⟩, ⟨I-Caus`, T-Caus`, L-Caus`, ITL-Caus`⟩⟩ faz parte do comportamento ISDL definido na anteriormente e o prefixo da ação adaptativa é definido conforme o tipo de uma ação adaptativa (prefixo), que desempenha uma função passível às seguintes características:

- Ação adaptativa de inspeção: tem função de inspecionar/verificar se determinadas estruturas ISDL fazem parte da especificação do comportamento modelado.

- Ação adaptativa de eliminação: tal estrutura é utilizada para eliminar estruturas ISDL necessárias às modificações que devem ocorrer no sistema modelado. Ao executar uma ação adaptativa de eliminação estruturas são eliminadas do comportamento ISDL
- Ação adaptativa de inserção: serve para adição de novas estruturas ao comportamento ISDL. É por meio da execução de ações adaptativas de adição que novas estruturas são adicionadas ao comportamento ISDL

A operação do dispositivo $ISDL_{Adp}$ ocorre por meio da modificação das ações pertencentes a um comportamento pela evolução gradual do seu conjunto de regras, com a adição ou remoção de ações efetuadas por meio da execução das ações adaptativas.

Estando o $ISDL_{Adp}$ em sua situação inicial, este recebe uma seqüência de eventos externos. As transições são executadas conforme essa seqüência, consumindo os eventos externos, repetindo-se o processo até o final da seqüência. Para cada evento recebido, verifica-se qual ou quais são as produções a serem executadas. No que se refere à execução do $ISDL_{Adp}$, se a ação adaptativa A estiver presente, será executada em primeiro lugar, ao passo que se B estiver declarada, será executada por último.

5.3.3. Modelagem da aplicação *CpCl* usando $ISDL_{Adp}$

Nesta seção será desenvolvido um exemplo ilustrativo utilizando o dispositivo $ISDL_{Adp}$ para a modelagem de aplicações complexas. A aplicação escolhida para ser modelada é comum entre os usuários de computadores pessoais e de tecnologia Microsoft Windows (MICROSOFT, 2005a) e tem por objetivo ilustrar o funcionamento das opções "*Copiar*", "*Colar*" e "*Colar Especial*", que será denominda aplicação *CpCl*, oferecidas na maioria dos produtos disponíveis para esta tecnologia. Na Figura 50 é ilustrada parte da interface da ferramenta Microsoft Office Word 2003 (MICROSOFT, 2005b), na qual aparecem desabilitadas as opções "*Copiar*", "*Colar*" e "*Colar Especial*" devido ao conteúdo da área de transferência estar vazio e nenhum conteúdo ter sido selecionado.

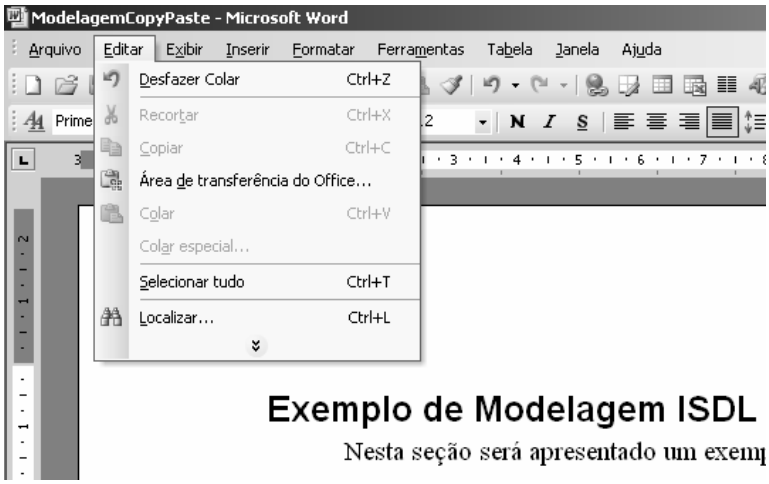


Figura 50. Situação inicial MS-Word Office XP 2003

Para a opção "*Colar*" ser ativada, faz-se necessário que o usuário selecione um texto, imagem ou algum outro objeto que torne a opção ativa.

Ao selecionar a opção "*Copiar*" ou ativá-la por teclas de atalho, é copiado o conteúdo selecionado para a Área de Transferência e habilitada a ocorrência das opções "*Colar*" e "*Colar Especial*" do menu de opções, conforme ilustrado na Figura 51.

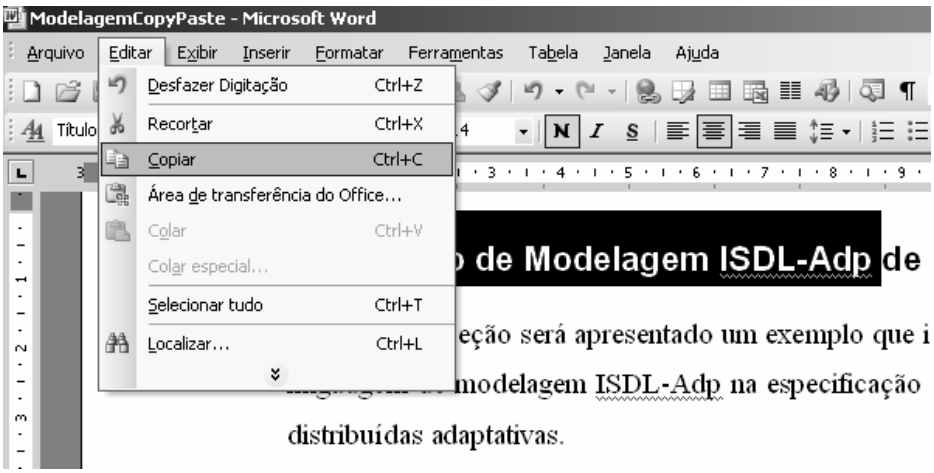


Figura 51. Habilitação da opção *"Copiar"*

Pode-se observar que ao ser selecionado e copiado um texto, seu conteúdo para a área de transferência é atribuído à operação de colar "*Texto Formatado*" para a opção "*Colar*" e as operações de colar "*Texto unicode sem formatação*", "*Formato Html*", "*Texto Formatado*", "*Texto sem Formatação*", "*Texto Formatado RTF*" etc., para a opção "*Colar Especial*". Tais opções são habilitadas conforme o conjunto de programas instalados anteriormente, os quais permitiram que determinadas operações fossem disponibilizadas. A Figura 52 ilustra a interface que é apresentada após a escolha da opção "*Colar Especial*". A interface habilitada apresenta ao

usuário uma lista de possíveis operações para serem realizadas conforme sua necessidade e escolha.

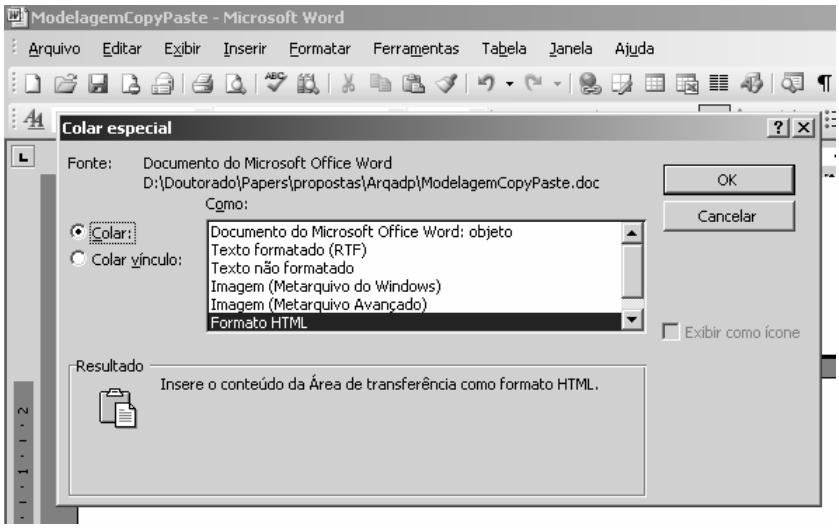


Figura 52. Operações disponíveis para *"Colar Especial"* para objetos do tipo *texto*

Em relação à cópia do conteúdo de uma imagem para a área de transferência, ocorre situação semelhante a que ocorre ao ser copiado um texto, ou seja, ao selecionar uma imagem, é executada a ação de cópia de conteúdo para a área de transferência disponível na opção "*Copiar*", que atribuirá as operações "*Colar Bitmap*" para a opção "*Colar*", e as operações: "*Bitmap*", "Imagem *(Meta Arquivo Avançado)*", "*Imagem GIF*" etc., para a opção "*Colar Especial*". Tal situação se dá de forma análoga a opção de "*Colar Especial*" para um texto. As opções de tipo de colagem a ser executado são disponibilizadas conforme o conjunto de programas instalados anteriormente no equipamento, bem como o tipo de imagem selecionado e a origem do conteúdo a ser copiado para a área de transferência (programa em que foi gerada a cópia). Desta forma, pode-se observar, na Figura 53, que o conjunto de operações disponibilizadas pelo comando "*Colar Especial*" pode ser alterado em tempo de execução e disponibilizar diferentes operações conforme características da origem e do conteúdo copiado. Tal situação ilustra a aplicação de tecnologia adaptativa e demonstra o seu uso na prática. Vale ressaltar que este trabalho não tem foco principal na modelagem de características das opções "*Copiar*", "*Colar*" e "*Colar Especial*", e sim, na aplicação de tecnologia adaptativa à modelagem de aplicações complexas. Outras opções de cópias de conteúdos como imagem vetorial, áudio, vídeo etc., são sugeridas para efeitos de estudos futuros.

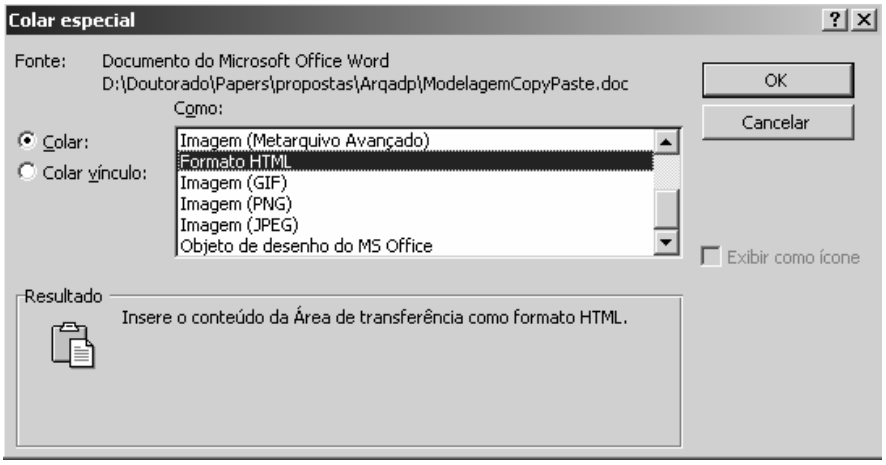


Figura 53. Operações disponíveis para *"Colar Especial"* para objetos tipo *imagem*

Fundamentado no problema descrito anteriormente, é apresentada a modelagem $ISDL_{Adp}$ da aplicação *CpCI*. A Figura 54 representa a modelagem de entidades da aplicação *CpCI* que é representada na visão externa (Figura 54a) por um Usuário que interage com um Sistema de Computação. Na visão interna do sistema de computação (Figura 54b) foram abstraídos alguns detalhes visando a simplificar sua apresentação. Desta forma, temos o Sistema de Computação composto por três subentidades: Interface da Aplicação, Sistema Operacional e Área de Transferência. A Interface da Aplicação tem por finalidade receber interações vindas do usuário e enviar ao sistema operacional, objetivando sua execução. Esta entidade também recebe mensagens vindas do Sistema Operacional e da Área de Transferência e as apresenta aos usuários. A subentidade Sistema Operacional tem por objetivo realizar o gerenciamento do Sistema de Computação; e a subentidade Área de Transferência é um espaço de memória que permite ao usuário realizar cópias de dados ao utilizar suas aplicações ou intercambiar dados entre as aplicações utilizadas.

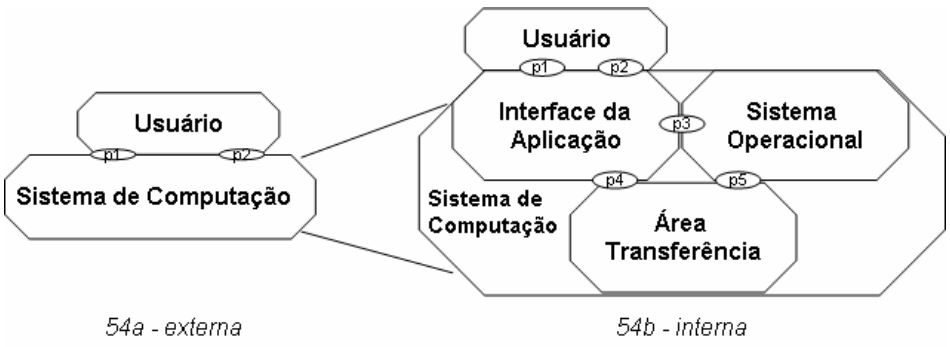


Figura 54. Modelo entidades aplicação *CpCI*

A subentidade Interface da Aplicação possui diversas funcionalidades conforme a aplicação que é executada. Na Figura 55 é ilustrada parte do comportamento *C_Copiar_Colar*. Tal comportamento representa as funcionalidades de copiar e colar conteúdo para Área de Transferência e é comum para a maioria das aplicações existentes. O comportamento ilustrado representa a configuração inicial da aplicação e constitui-se da interação *i_Cop* que tem por objetivo modelar o recebimento das interações vindas do usuários referentes à requisição de cópia de um determinado conteúdo selecionado para a área de transferência. Ao ser ativada a interação *i_Cop*, ocorre a habilitação das ações *aCopTxt* e *aCopImg*, que possuem a função adaptativa anterior *FReset(str)*, responsável por restaurar o comportamento em execução para a situação inicial caso este tenha sido adaptado anteriormente, e a função adaptativa posterior *FCop(str,str)*, responsável por realizar mudanças no comportamento atual conforme o conteúdo selecionado pelo usuário.

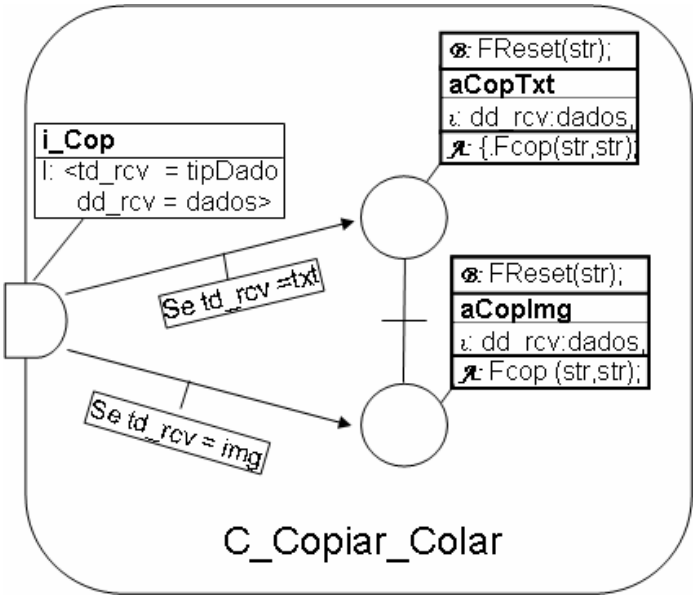


Figura 55. Comportamento inicial aplicação *CpCI*

Na Figura 56 são apresentados os subcomportamentos *C_Cop_Txt* e *C_Cop_Img*. O subcomportamento *C_Cop_Txt* é responsável por adicionar ou remover do comportamento inicial funcionalidades relacionadas à cópia do conteúdo do tipo texto. Este subcomportamento, ao receber uma mensagem oriunda da interação *icl* no ponto de entrada *e1*, referente a colar o conteúdo da área de transferência, habilita a ocorrência da ação *aCl_TxtFmt* responsável por colar o conteúdo da área de transferência como se fosse do tipo texto formatado. Ao receber uma mensagem no ponto de entrada *e2*, oriunda da interação *icle* (colar especial), ocorre a habilitação da escolha de ocorrência de uma das ações

aCl_TxtFmt, *aCl_TxtSemFmt* e *aCl_TxtHtml*, responsáveis, respectivamente, por colar textos formatados, sem formatação ou no formato HTML. Após a execução de uma das possíveis ações, é enviada mensagem por meio do ponto de saída *s1* contendo informações para serem apresentadas ao usuário por meio da interação *i_apu*.

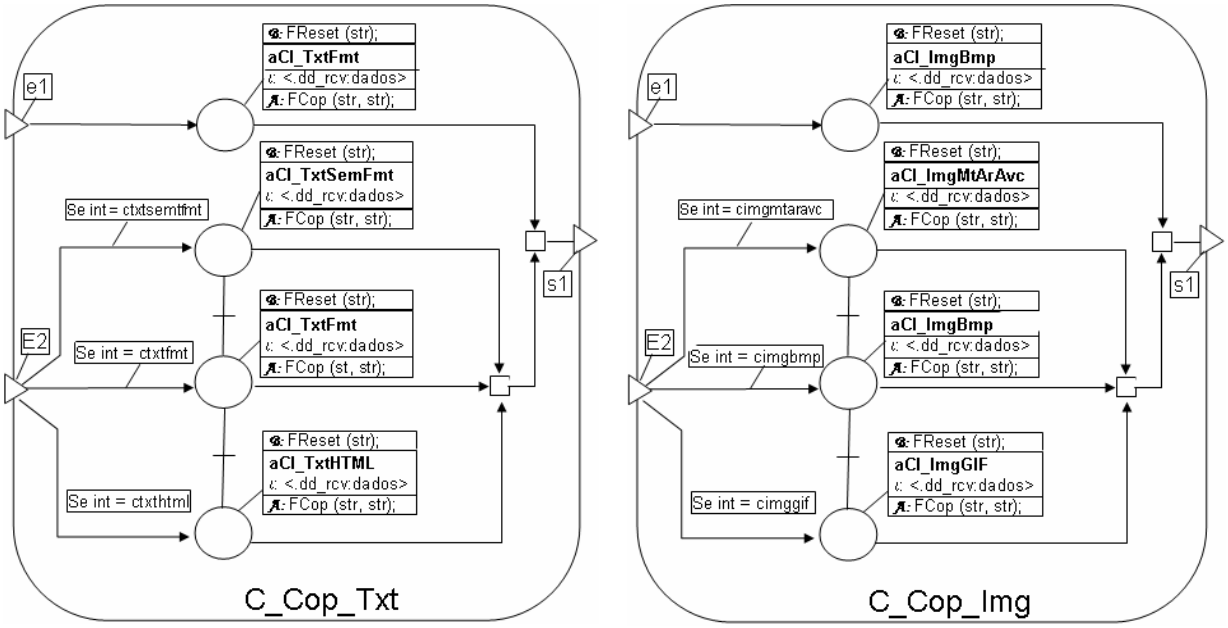


Figura 56. Subcomportamentos *C_Cop_Txt* e *C_Cop_Img*

O subcomportamento *C_Cop_Img* é constituído de forma semelhante ao comportamento *C_Cop_Txt*, diferenciado-se, no conjunto, de ações a serem desempenhadas e que estão relacionadas à operação de colar imagens. A ação *ACl_ImgBmp* é responsável por colar imagens do tipo *Bitmap* e é padrão para a interação *icl*. As ações *aClImgBmp*, *aCl_ImgMtArAvc* e *aCl_ImgGif* são responsáveis, respectivamente, por colar imagem do tipo *Bitmap*, *Meta Arquivo Avançado* e *Gif* estão relacionadas à interação *icle* e ocorrem em escolha.

Na Figura 57 são apresentadas as funções adaptativa *Fcop(str, str)* e *FReset(str).* Tais funções são responsáveis por realizar mudanças no comportamento corrente em um determinado instante de execução. A função *Fcop(str,str)* recebe dois parâmetros referente ao nome do comportamento inicial e comportamento que será inserido (adaptado). Ao ser chamada esta função, ocorre a execução de um conjunto de ações adaptativas elementares que têm por objetivo inserir um novo subcomportamento que deverá desempenhar a função de colar conteúdo da área de transferência. Após a inserção do novo subcomportamento, ocorre a ligação de seus pontos de entrada e de saída com as respectivas

interações do comportamento inicial, de forma que a aplicação possa disponibilizar as novas funcionalidades inseridas em seu conjunto de regras.

```
FCop(inicial: char, adaptador: char);
{
   + [adaptador];
   + [icl.inicial   ← e1.adaptador];
   + [icle.inicial ← e2.adaptador];
   + [s1.inicial   ← i_apu.adaptador];
   + [aCopTxt.inicial ∨ aCopImg.inicial ← icl.inicial];
   + [aCopTxt.inicial ∨ aCopImg.inicial ← icle.inicial];
}
```

Figura 57. Funções adaptativas *FCop(...)*

A função *FReset(str),* descrita na Figura 58, executa, inicialmente, uma ação elementar de consulta e verifica no conjunto de regras a existência de uma regra que atenda às condições exigidas. Na existência da regra, a variável adaptador recebe o nome do respectivo subcomportamento que atende a referida consulta. Após ser definido qual subcomportamento pertence à configuração atual, este e suas ligações de entrada e de saída devem ser removidas de forma que a aplicação retorne a sua situação de configuração inicial.

```
FReset (inicial: char)
var
  adaptador: char;{
     ? [icl.inicial ← e1.adaptador];
     - [icl.inicial  ← e1.adaptador];
     - [icle.inicial ← e2.adaptador];
     - [s1.adaptador ← i_apu.adaptador];
     - [aCopTxt.inicial ∨ aCopImg.inicial ← icl.inicial];
     - [aCopTxt.inicial ∨ aCopImg.inicial ← icle.inicial];
     - [adaptador];
}
```

Figura 58. Funções adaptativas FReset(...)

Na Figura 59 é apresentado o comportamento *C_Copiar_Colar* que foi modificado ao ser habilitada a execução da ação *aCopImg* e suas respectivas ações adaptativas anterior e posterior. Anteriormente à execução de *aCopImg*, foi executada a função adaptativa *FReset(str)*, que restaurou o comportamento para a configuração inicial. Na seqüência, ocorreu a execução da ação *aCopImg*, que copiou o conteúdo selecionado para a área de transferência e, após a execução de *aCopImg*, foi executada a ação adaptativa *FCop(str, str)*, que realizou mudanças no comportamento inicial adicionando novas funcionalidades e permitindo a realização de colar conteúdo da área de transferência do tipo imagem.

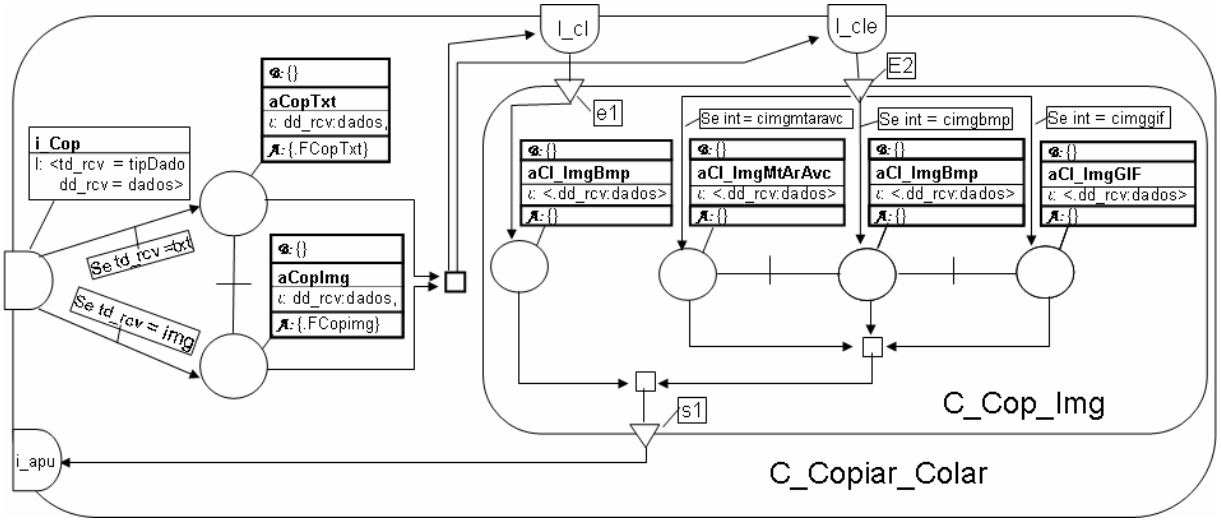


Figura 59. Comportamento *Cp_Cl* após execução de funções adaptativas

5.3.1. Mapeamento do dispositivo $ISDL_{Adp}$ e da aplicação *CpCl* para modelo lógico.

Para realizar o mapeamento da especificação da aplicação *CpCl*, faz-se necessário o mapeamento do dispositivo $ISDL_{Adp}$ para a estrutura de dados genérica para representação de dispositivos adaptativos dirigidos por regras. Inicialmente, é definido um objeto na classe *Dispositivos,* ilustrada na Tabela 32, para representar o dispositivo $ISDL_{Apd}$.

Tabela 32. Mapeamento de objetos da classe Dispositivo para o dispositivo $ISDL_{Adp}$

Atributo	Valor
código_dsp	2
descrição	*Adaptive Interaction Sytem Design Language* -$ISDL_{Adp}$
data_criação	15/01/2005
data_última_atualização	15/07/2005
especialista	Almir Rogério Camolesi

Depois de serem definidos os elementos básicos do dispositivo, devem ser definidos os tipos de componentes que o mesmo oferece para o projetista durante a especificação de suas aplicações. Na Tabela 33 são descritos os possíveis tipos de componentes permitidos para o dispositivo $ISDL_{Adp}$.

Tabela 33. Mapeamento de objetos da classe Tipo de Componente para o dispositivo $ISDL_{Adp}$

código_tcpt	código_dsp	descrição	conexão
5	2	Comportamento $ISDL_{Adp}$	T
6	2	Sub-Comportamento $ISDL_{Adp}$	F
7	2	Interação $ISDL_{Adp}$	F
8	2	Ação $ISDL_{Adp}$	F
9	2	Conector E $ISDL_{Adp}$	F
10	2	Conector OU $ISDL_{Adp}$	F
11	2	Ponto de Entrada $ISDL_{Adp}$	F
12	2	Ponto de Saída $ISDL_{Adp}$	F

De forma semelhante aos tipos de componentes, devem ser mapeados os tipos de conexão que o dispositivo suporta. Na Tabela 34 são ilustrados os possíveis tipos de conexão disponíveis para o dispositivo $ISDL_{Adp}$.

Tabela 34. Mapeamento de objetos da classe Tipo de Conexão para o dispositivo $ISDL_{Adp}$

código_tcnx	código_dsp	descrição
3	2	Habilitação $ISDL_{Adp}$
4	2	Desabilitação $ISDL_{Adp}$
5	2	Escolha $ISDL_{Adp}$
6	2	Sincronização $ISDL_{Adp}$
7	2	Entrelaçamento $ISDL_{Adp}$

Também ao se definir um dispositivo, devem ser mapeados os conceitos referentes aos tipos de atributos que o mesmo suporta. Na Tabela 35 é apresentada a materialização dos tipos de atributos disponíveis para o dispositivos $ISDL_{Adp}$.

Tabela 35. Mapeamento de objetos da classe Tipo de Atributo para o dispositivo $ISDL_{Adp}$

código_ta	código_dsp	descrição
6	2	Informação $ISDL_{Adp}$ (ι)
7	2	Tempo $ISDL_{Adp}$ (τ)
8	2	Localização $ISDL_{Adp}$ (λ)

Com base nos elementos mapeados para a definição do dispositivo $ISDL_{Adp}$ é realizada a especificação da aplicação *CpCI*. Inicialmente, conforme ilustrado na Tabela 36 é definido um novo projeto que representa as características da aplicação que será desenvolvida.

Tabela 36. Mapeamento de objetos da classe Projeto – Aplicação *CpCI*

Atributo	Valor
código_prj	3
código_dsp	2
descrição	Aplicação Copiar/Colar
data_criação	20/01/2005
data_última_atualização	15/07/2005
autor	Almir Rogério Camolesi

Após ser definido o projeto, deve ser realizado o mapeamento da especificação do comportamento não-adaptativo da aplicação *CpCI*. Sendo assim, devem ser definidos os componentes relacionados à aplicação. Os componentes do

comportamento inicial da aplicação *CprCl* e do subcomportamento *C_Cop_Txt* são apresentados na Tabela 37.

Tabela 37. Mapeamento de objetos da classe Componente – Aplicação *CpCl*

código_cpt	código_prj	código_tipo_cpt	descrição	código_cpt_rel
7	3	5	C_Copy_Paste	0
8	3	7	I_Cop	7
9	3	8	A_CopTxt	7
10	3	8	a_CopImg	7
11	3	7	i_Cl	7
12	3	7	i_Cle	7
55	3	7	i_apu	7
32	3	5	C_Cop_Txt	0
33	3	7	e1	32
34	3	7	e2	32
35	3	7	s1	32
36	3	8	aCl_TxtFmt	32
37	3	8	aCl_TxtFmt	32
38	3	8	aCl_TxtSemFmt	32
39	3	8	aCl_TxtHTML	32
40	3	10	ou_DvsTxtFmt	32
41	3	10	ou_TpTxtFmt	32

Ao se definir um componente, também devem ser definidos os atributos que estão relacionados ao mesmo. Na Tabela 38 são apresentados os atributos relacionados aos componentes da aplicação *CpCl* e do subcomportamento *C_Cop_Txt*. Devido todos os atributos serem do mesmo tipo e manipularem os mesmos dados, a tabela será apresentada de forma resumida.

Tabela 38. Mapeamento de objetos da classe Atributo de Componente – Aplicação *CpCl*

código_atr	código_cpt	tipo_atr	descrição	valor
4	8	6	id_rcv	
...	...	...	...	...
17	48	6	dd_rcv	
18	48	6	dd_rcv	

Para que o comportamento não-adaptativo possa ter sentido, faz-se necessário a especificação dos relacionamentos existentes entre seus componentes. Tal relacionamento possibilita a constituição do comportamento da aplicação. Na Tabela 39 são descritos os relacionamentos (conexões) existentes para a aplicação *CpCl*.

Tabela 39. Mapeamento de objetos da classe Conexão – Aplicação *CpCI*

código_cnx	código_prj	tipo_cnx	código_org	código_dst	descrição
7	3	3	8	9	Se td_rcv = txt → aCopTxt
8	3	3	8	10	Se td_rcv = img → Hab aCopImg
9	3	5	9	10	aCopTxt V → aCopImg
10	3	3	9	13	a_CopTXT → ou_TXTIMG
12	3	3	13	11	out_TXTIMG → i_Cl
13	3	3	13	12	out_TXTIMG → i_Cle
14	3	3	33	36	e1 →aCl_TxtFmt
15	3	3	34	37	e2 →aCl_TxtFmt

Os atributos de conexões também devem ser mapeados. A Tabela 40 representa, de forma resumida, os atributos de conexões do subcomportamento *a_Cop_Txt*, da aplicação *CpCI*.

Tabela 40. Mapeamento de objetos da classe Atributo de Conexão – Aplicação *CpCI*

código_atr	código_cnx	tipo_atr	descrição	valor
14	7	6	dd_rcv	
15	8	6	dd_rcv	
16	9	6	dd_rcv	
....	....	....	....	....
28	23	6	dd_rcv	
29	24	6	dd_rcv	
30	25	6	dd_rcv	

Depois de realizada a especificação do comportamento não-adaptativo, deve ser realizada a especificação da camada adaptativa. Na Tabela 41 é apresentado o mapeamento das funções adaptativas *FCop()* e *FReset()* que fazem parte da especificação da aplicação *CpCI*.

Tabela 41. Mapeamento de objetos da classe Função Adaptativa – Aplicação *CpCI*

código_fa	projeto	descrição
3	3	FCop (inicial, adaptador)
4	3	FReset(inicial)

Para que uma função adaptativa desempenhe o seu papel, faz-se necessário a especificação de suas ações adaptativas. Na Tabela 42 é descrito o mapeamento das ações adaptativas da função *FCop()*.

Tabela 42. Mapeamento de objetos da classe Ação Adaptativa – Aplicação *CpCI*

código_aa	código-fun_adp	tipo_fun	tipo_cnx	tipo_org	código_org	tipo_dst	código_dst	descrição
0	3	3	3	5	&adaptador	0		+adaptador
1	3	3	3	7	i_cl.inicial	11	e1.adaptador	+icl.inicial → e1.adaptador
2	3	3	3	7	i_cle.inicial	11	e2.adaptador	+icle.inical → e2.adaptador
3	3	3	3	7	s1.inicial	7	i_apu.adaptador	+ s1.inicial → i_apu.adaptador
4	3	3	3	8	aCopTxt.inical	10	ouCopTxtIMG	+aCopTxt.inical → ouCopTxtIMG
5	3	3	3	8	aCopImg.&inicial	10	ouCopTxtImg	+aCopImg.&inicial → ouCopTxtImg
6	3	3	3	10	ouCopTxtImg	7	icl.&inicial	+ouCopTxtImg → icl.&inicial
7	3	3	3	8	aCopTxt.&inical	10	ouComtxtImg_e	+aCopTxt.&inical →ouComtxtImg_e
8	3	3	3	8	aCopImg.&inicial	10	ouCopTxtImg_e	+aCopImg.&inicial →ouCopTxtImg_e
9	3	3	3	10	ouCopTxt.Img_e	7	icle.&inicial	+ouCopTxt.Img_e → icle.&inicial

Conforme apresentado anteriormente, somente são definidos, para a aplicação *CpCI*, os atributos de informação referente aos dados recebidos. Desta forma, os atributos de componentes de ações adaptativas de origem e de destino, são semelhantes aos atributos de componentes. E ainda, os atributos de conexão de ações adaptativa também são idênticos aos atributos de conexão e são definidos da mesma forma que os atributos apresentados na Tabela 39.

Depois de definidos as funções adaptativas e suas ações, deve ser realizada a associação das funções adaptativa aos componentes e/ou conexões da especificação subjacente da aplicação. Tal mapeamento é apresentado na Tabela 43.

Tabela 43. Mapeamento de objetos da classe Acoplamento – Aplicação *CpCI*

código_acoplamento	tipo_acoplamento	código_cptcnx	fun_adp_ant	fun_adp_pos
5	C	9	3	4
6	C	10	3	4

Para que uma função adaptativa possa ser executada é preciso que sejam especificados seus geradores e parâmetros. No exemplo, aplicação *CpCI*, não foram utilizados geradores, porém foram definidos alguns parâmetros para as funções *FCop()* e *FReset()*. Na Tabela 44 são descritos os parâmetros que foram definidos para as funções pertencentes a tal aplicação.

Tabela 44. Mapeamento de objetos da classe Parâmetros – Aplicação *CpCI*

código_par	código_fun	descrição	valor
11	6	Inicial	
12	6	Adaptador	
13	7	Adaptador	

6. Considerações finais

Este trabalho buscou contribuir para o projeto e desenvolvimento de aplicações empregando Tecnologia Adaptativa. Buscou-se trazer contribuições tanto para a área teórica quanto no uso de técnicas e ferramentas que possibilitem a utilização da Tecnologia Adaptativa no projeto de aplicações complexas. Neste capítulo são apresentadas uma avaliação geral do trabalho realizado, suas principais contribuições e sugestões para trabalhos futuros.

6.1. Avaliação do trabalho

Inicialmente, foram desenvolvidos estudos com o objetivo de agregar aos dispositivos não-adaptativos subjacentes os recursos providos pela camada adaptativa (NETO, 1993). Valendo-se da representação geral para dispositivos adaptativos elaborada por Neto (2001), foram definidos dois novos dispositivos: Rede de Petri Adaptativa (RP_{Adp}) e *Adaptive Interaction System Design Language* ($ISDL_{Adp}$).

Outros dispositivos adaptativos (Autômatos Adaptativos e Autômato de Estados Finitos, Statechart e Gramática), previamente definidos individualmente em outros trabalhos, foram denotados usando a representação geral para dispostivos adaptativos. Com base nos dispositivos definidos e nas denotações realizadas, foi possível avaliar a forma geral para dispositivos adaptativos e verificar que a configuração corrente (status) de uma aplicação em um determinado momento não é representada de forma adequada em tal estrutura. Desta forma, sugere-se que estudos sejam realizados para o aprimoramento da referida representação, os quais deverão buscar uma melhor formalização. Também se observou que é possível adicionar novas camadas adaptativas a dispositivos adaptativos já existentes, passando então estes a terem a possibilidade de realizarem modificações no conjunto de funções e ações adaptativas.

Com base nos estudos realizados, foi possível propor uma estrutura de dados geral (modelo lógico) que permite representar os elementos conceituais dos dispositivos - subjacentes e/ou adaptativos - definidos. O modelo lógico desenvolvido foi organizado em camadas para facilitar o seu entendimento e o mapeamento de formalismos e especificações para este tipo de estrutura. Uma outra

vantagem de o modelo lógico ser organizado em camadas é a possibilidade de sua utilização para a representação de dispositivos subjacentes e as especificações de aplicações produzidas com base em tais dispositivos apenas — sem a necessidade de usar a camada adaptativa. Uma vez assim definido, o modelo lógico permite a descrição de dispositivos - adaptativos ou não adaptativos - dirigidos por regras e provê recursos para que outros dispositivos sejam estudados e mapeados.

Ainda em relação ao desenvolvimento de aplicações, por meio de dispositivos adaptativos, foi detectada a necessidade de um método para o projeto de aplicações usando tais dispositivos. Nesse sentido, foi definida uma metodologia visando ao desenvolvimento de aplicações, utilizando-se de tecnologia adaptativa, pela qual pôde ser observada, também, a necessidade de um ambiente para auxiliar o projetista no desenvolvimento de suas aplicações. Tal ambiente deve dar suporte ao projetista em todo o ciclo de vida (especificação, análise e representação física) do desenvolvimento de aplicações e permitir que inconsistências sejam verificadas e corrigidas durante a realização do trabalho. O ambiente proposto fundamenta-se no modelo lógico definido para descrever os dispositivos definidos e as especificações produzidas. A utilização do modelo lógico mostrou-se de fundamental importância, pois permitiu uma maior agilidade e facilidade na definição de novos dispositivos para suportarem tecnologia adaptativa. Ainda em relação ao ambiente, também foi proposta uma arquitetura concebida em módulos. Tal arquitetura permite que partes da ferramenta (módulos) sejam alteradas, inseridas ou removidas sem que ocorram mudanças na estrutura do ambiente proposto.

Com a realização dos trabalhos de definição de dispositivos, de estruturação do modelo lógico e da proposição do ambiente para a modelagem de aplicações, verificou-se que seria importante a concepção de um método para definição de dispositivos adaptativos. Tal método tem por objetivo auxiliar os especialistas na definição de novos dispositivos e consiste em três etapas: teórica, lógica e física. Na etapa de representação formal (teórica), o especialista, com base na representação geral faz a definição do dispositivo subjacente e acrescenta a ele a camada adaptativa. Na seqüência, realiza o mapeamento do novo dispositivo para o modelo lógico e, posteriormente, utilizando-se de um ambiente, pode realizar o projeto de suas aplicações. Para tornar mais rápido e eficiente o trabalho do especialista que utiliza o referido método foi proposta a concepção de um gerador de ambientes para modelagem com uso da tecnologia adaptativa. O gerador de ambientes tem por

objetivo auxiliar os especialistas na realização das etapas do método para definição de dispositivos adaptativos e obter, quase automaticamente, os novos dispositivos adaptativos e os seus respectivos ambientes para a modelagem de aplicações.

O gerador de ambientes permite a um especialista que, depois de realizar a formalização de um dispositivo, utilize um conjunto de ferramentas para auxiliar na descrição do modelo lógico e dos principais elementos que constituirão o ambiente de modelagem para o referido dispositivo. Inicialmente, o especialista define o dispositivo e a descrição dos elementos necessários para a geração do ambiente. Depois de realizadas essas tarefas, o gerador de ambientes é executado e obtém-se como saída um ambiente para a modelagem de aplicações adaptativas para ser utilizado pelos projetistas na realização de seu trabalho.

Com o objetivo de validar o gerador proposto, foi definido um ambiente simplificado para a modelagem de aplicações adaptativas. Tal ambiente consiste de duas ferramentas: uma para descrição de dispositivos e especificação de aplicações usando o modelo lógico e outra para simular as aplicações especificadas, ambas utilizando-se do modelo lógico e de um *framework* para auxiliar os especialistas na implementação de simuladores para os dispositivos definidos. A realização destes trabalhos serviu para demonstrar que a proposta do gerador é consistente e ajudou a validar o modelo lógico definido. As ferramentas obtidas também podem auxiliar um projetista na especificação de suas aplicações, apesar de não possuírem uma interface tão intuitiva e amigável ao usuário.

Com o objetivo de avaliar os dispositivos definidos e as ferramentas desenvolvidas foram realizados estudos de casos que empregam o uso da tecnologia adaptativa no seu projeto. Neste contexto, foram desenvolvidas três estudos de caso, que buscaram mostrar o uso dos dispostivos AEF_{Adp}, RP_{Adp} e $ISDL_{Adp}$ e as ferramentas desenvolvidas para tais dispositivos. O estudo de caso realizado em relação ao AEF_{Adp} permitiu demonstrar a utilização da tecnologia adaptativa na modelagem de elementos abstratos. O exemplo apresentado, no qual foi utilizado o RP_{Adp}, permitiu a demonstração do emprego de tal dispositivo na modelagem de aplicações telemáticas e pôde também possibilitar a constatação de que este pode ser empregado na modelagem de outros tipos de aplicações. Em relação ao dispositivo $ISDL_{Adp}$, este permitiu avaliar o emprego da tecnologia adaptativa na especificação de interfaces de aplicações (CAMOLESI; NETO, 2002),

bem como a sua utilização no desenvolvimento de aplicações complexas (CAMOLESI; NETO, 2004a).

6.2. Contribuições

Resumidamente, são apresentadas, a seguir, as principais contribuições alcançadas por este trabalho:

- Definição do dispositivo Rede de Petri Adaptativa;
- Definição do dispositivo ISDL Adaptativo;
- Representação de outros dispositivos na forma geral para dispositivos adaptativos proposta em (NETO, 2001);
- Definição de um modelo lógico para mapeamento dos conceitos da formulação geral para dispositivos adaptativos;
- Proposição de um método para auxiliar os especialistas na definição de novos dispositivos adaptativos;
- Proposição de um método para auxiliar os projetistas no projeto de aplicações usando Tecnologia Adaptativa;
- Proposição e definição de um ambiente completo para o projeto de aplicações com Tecnologia Adaptativa;
- Proposição de um gerador de ambientes para a modelagem de aplicações usando Tecnologia Adaptativa;
- Implementação de ferramentas para a concepção do gerador de ambientes e, correspondentemente, do ambiente simplificado proposto;
- Realização de estudos de casos valendo-se dos dispositivos definidos e das ferramentas desenvolvidas;
- Material e ferramentas para auxílio didático/pedagógico relacionado ao ensino e utilização de Tecnologia Adaptativa.
-

6.3. Sugestões para trabalhos futuros

Os trabalhos desenvolvidos serviram para avaliar alguns conceitos desenvolvidos para a área de Tecnologia Adaptativa e trouxe uma diversidade de

questionamentos relacionados ao uso de tais conceitos no desenvolvimento de aplicações complexas. Desta forma, é possível sugerir alguns tópicos para serem pesquisados e avaliados em relação ao uso da tecnologia adaptativa neste contexto:

- Estudos das implicações na extensão de um dispositivo adaptativo para suportar uma nova camada adaptativa, ou seja, utilizar os dispositivos adaptativos desenvolvidos como dispositivos subjacentes e permitir que o conjunto de funções e ações adaptativas sejam modificados;
- Estudos referentes ao mapeamento de dispositivos com características denotacionais (gramáticas) para a representação geral proposta por Neto (2001) e, conseqüentemente, para o modelo lógico proposto neste trabalho;
- Estudos que visem a estender a utilização de outros dispositivos (StateChart, Cadeias de Markov, Máquinas de Turing, etc.) dirigidos por regras para a avaliação da estrutura geral proposta neste trabalho;
- Estudos relacionados à definição de uma camada superior à definida para metamodelos e metambientes realizados neste trabalho, tornando possível a descrição de outros tipos de formalismos (contínuos, etc.) e a sua forma de operação;
- Construção do ambiente geral completo para o projeto de aplicações adaptativas proposto no Capítulo 5 (ferramentas para edição e análise, e transformação para representação física) e do gerador de ambientes específicos que usam o ambiente geral como base;
- Construção de um ambiente para auxiliar os especilistas na definição dos elementos necessários ao gerador de ambientes específicos;
- Definições de novos formalismos e o desenvolvimento de novas aplicações utilizando o ambiente simplificado para a modelagem de aplicações adaptativas com o objetivo de avaliar e buscar aprimoramentos nas ferramentas desenvolvidas;
- Estudos de aplicação dos dispositivos adaptativos no projeto de aplicações comerciais;
- Utilizar os conceitos e as ferramentas desenvolvidos em ambientes educacionais para o ensino dos conceitos de Tecnologia Adaptativa.

Referências bibliográficas

AL-KHALIFA H.; DAVIS H. **The Evolution of Metadata from Standards to Semantics in e-learning applications**. 7° Conference on Hypertext and Hypermedia, Odense, Denmark, 2006.

ALMEIDA, J.R. **STAD - Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos.** Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, São Paulo, 1995.

ARGOUML. **Ferramenta para modelagem de aplicações usando linguagem UML**. Tigris.org - Open Source Software Engineering Tools. Disponível em: http://argouml.tigris.org/, Janeiro, 2007.

ATOM3**, AToM 3 Project**, in: http://atom3.cs.mcgill.ca/, Janeiro, 2007.

BARRETO, C.M. **Modelo de metadados para a descrição de documentos eletrônicos na WEB**. Dissertação de Mestrado, DC - Instituto Militar de Engenharia, Rio de Janeiro, Fevereiro, 1999.

BIRRER, A.; EGGENSCHWILER, T. **Frameworks in the financial engineering domain**. Springer-Verlag, proceedings of the European conference on object-oriented programming, Kaiserslautern, Germany, 1993.

CAMOLESI, A.R. **Uma metodologia para o Design de Serviços de TV-Interativa**. Dissertação de Mestrado, PPG-CC, UFSCar, 2000.

CAMOLESI, A.R.; NETO, J.J. **Modelagem AMBER$_{Adp}$ de um Ambiente para Gerenciamento de Ensino a Distância.** Anais do XIII Simpósio Brasileiro de Informática na Educação - SBIE 2002, pp. 401-409, São Leopoldo, RS, Novembro 12-14, 2002.

CAMOLESI, A.R.; NETO, J.J. **An adaptive model for specification of distributed systems.** IX Congreso Argentino de Ciencias de la Computación, La Plata, Argentina, 6-10 de Outubro, 2003.

CAMOLESI, A.R.; NETO, J.J. **Modelagem Adaptativa de Aplicções Com-plexas**. XXX Conferencia Latinoamericana de Informática - CLEI'04. Arequipa - Peru, Setiembre 27 - Octubre 1, 2004a.

CAMOLESI, A.R.; NETO, J.J. **Representação Intermediária para Dispositivos Adaptativos Dirigidos por Regras**. 3rd International Information and Telecommunication Technologies Symposium, UFSCar, São Carlos, Brasil, 2004b.

CAMOLESI, A.R. **Modeling a tool for the generation of programming environments for adaptive formalism.** International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2005), Springer Verlag editor, pp. 341-344, Coimbra University, Coimbra, Portugal, 21 - 23 march 2005.

CARDOSO, J.; VALLETE, R. **Redes de Petri**. Editora UFSC, Florianópolis, 1997.

DUVAL, E. **Metadata standards: What, who & why. Journal of Universal Computer Science, 7(7):591-601,** Special Issue: I-Know 01 - International Conference on Knowledge Management. in: http://www.jucs.org/jucs_7_7/metadata_standards_what_who, **July 2001**.

DUVAL, E. et al. **Metadata Principles and Practicalities**. D-Lib Magazine, Vol 8, Number IV, April 2002

ECLIPSE. **Ambiente para o desenvolvimento de aplicações usando a linguagem Java.** Eclipse Foudation Inc. Disponível em http: //www.eclipse.org, Janeiro, 2007.

FREITAS, A.V. **Aspectos do projeto e implementação de ambientes multilinguagens de programação.** Dissertação de Mestrado, USP, São Paulo, 2000.

FREITAS, A.V.; NETO, J.J. **Aspectos da Implementação de um Ambiente Multilinguagem de Programação.** Anais do CACIC2000 - VI Congreso Argentino de Ciencias de la Computación. Ushuaia, Argentina, 2000a.

FREITAS, A.V.; NETO, J.J. **Ambiente Multilinguagem de Programação - Aspectos do Projeto e Implementação.** Boletim Técnico PBT/PCS/0109, ISSN 1413, 215X, Escola Politécnica, São Paulo, 2000b.

FREITAS, A.V. ; NETO, J.J. **Uma Ferramenta para Construção de Aplicações Multilinguagens de Programação**. CACIC'2001 - Congreso Argentino de Ciencias de la Computación, 15-20 Outubro, El Calafate, Argentina, 2001.

FRIENDS Project, in http://www.telin.nl/Middleware/FRIENDS/ENindex.htm, Agosto, 2005.

GEF. **Graph Edit Framework**. Tigris.org - Open Source Software Engineering Tools. disponível em: http://gef.tigris.org/, Janeiro, 2007.

HAREL D. et al. **On the formal semantics of statecharts.** In: Symposyum on logic in Computer Science, 2°, Ithaca, Proceedings, IEEE Press, pp. 54-64, New York, 1987.

IWAI; M.K. **Um formalismo gramatical adaptativo para linguagens dependentes de contexto.** Tese de Doutorado, USP, São Paulo, 2000.

JAVA. **Kit de desenvolvimento jdk e manuais da linguagem Java**. Sun Microsystems Inc. Disponível em http: //www.sun.com, Janeiro, 2007.

LARA, J.; VANGHELUWE, H.; ALFONSECA, M. **Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3**. Software and Systems Modeling, Springer Verlag, Volume 3, Number 3, August, 2004

LEGO G. em http://www.lego.com/ , visitado em Fevereiro, 2007.

LEWIS, H.; PAPADIMITRIOUS, C.H. **Elements of the theory or computation**. Prentice Hall, editor, 1998.

LUZ, J.C. **Tecnologia adaptativa aplicada à otimização de código em compiladores**. Dissertação de Mestrado, USP, São Paulo, 2004.

LUZ, J.C.; NETO, J.J. **Tecnologia adaptativa aplicada à otimização de código em compiladores.** IX Congreso Argentino de Ciencias de la Computación, La Plata, Argentina, 6-10 de Outubro, 2003.

Microsoft Office Word 2003, Disponível: http://www.microsoft.com/brasil/office/editions/standard.asp, Agosto 2005b.

Microsoft Windows XP, em: *http://www.microsoft.com/brasil/windows/default.asp*, Agosto 2005a.

NETBEANS. **Ambiente para o desenvolvimento de aplicações usando a linguagem Java.** NetBeans Foudation Inc. Disponível em http: //www.netbeans.org, Janeiro, 2007.

NETO, J.J. e MAGALHÃES, M.E.S. **Um Gerador Automático de Reconhecedores Sintáticos para o SPD.** VIII SEMISH - Seminário de Software e Hardware, pp. 213-228, Florianópolis, 1981.

NETO, J.J. **Cross-Assemblers para Microprocessadores** - Geração Automática Através do SPD. I CONAI - Congresso Nacional de Automação Industrial, pp. 501-509, São Paulo, 1983.

NETO J.J. **Introdução a Compilação**. EDITORA: LTC, Rio de Janeiro, 1987

NETO, J.J. **Uma Solução Adaptativa para Reconhecedores Sintáticos**. Anais EPUSP - Engenharia de Eletricidade - série B, vol. 1, pp. 645-657, São Paulo, 1988.

NETO, J.J.; KOMATSU, W. **Compilador de Gramáticas Descritas na Notação de Wirth Modificada**. Anais EPUSP - Engenharia de Eletricidade - série B, vol. 1, pp. 477-517, São Paulo, 1988.

NETO, J.J. **Contribuições à metodologia de construção de compiladores**. Tese de Livre Docência, USP, São Paulo, 1993.

NETO, J.J. **Adaptive Automata for Context-Sensitive Languages**. SIGPLAN NOTICES, Vol. 29, n. 9, pp. 115-124, September, 1994.

NETO, J.J.; ALMEIDA Jr.J.R.; NOVAES, J.M. **Synchronized Statecharts for Reactive Systems**. Proceedings of the IASTED International Conference on Applied Modelling and Simulation, pp.246-251, Honolulu, Hawai, 1998.

NETO, J.J.; IWAI, M.K. **Adaptive Automata for Syntax Learning**. CLEI 98 - XXIV Conferencia Latinoamericana de Informatica, MEMORIAS. pp. 135-149, Quito, Equador, 1998.

NETO, J.J. **Adaptive Rule-Driven Devices - General Formulation and Case Study**. Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA

2001, Springer-Verlag, Vol.2494, pp. 234-250, Pretoria, South Africa, July 23-25, 2001.

NETO, J.J.; SILVA, P.S.M. **An adaptive framework for the design of software specification language**. Proceedings of the International Conferenceon Adaptive and Natural ComputINing Algorithms (ICANNGA), Springer Verlag editor, pp. 349-352, Coimbra University, Coimbra, Portugal, 21 - 23 march 2005.

NOVAES, J.M. **Um formalismo adaptativo com mecanismo de sincronização para aplicações concorrentes.** Dissertação de Mestrado, USP, São Paulo, 1997.

PEDRO, R. **Construção de uma Ferramenta Textual para Configuração de Ambientes e realização de Modelagem de Aplicações utilizando Tecnologia Adaptativa.** Trabalho de Conclusão de Curso - Bacharelado em Ciência da Computação, Instituto Municipal de Ensino Superior, FEMA, Assis-SP, 2005.

PEREIRA, J.C.D.; NETO, J.J. **Um Ambiente de Desenvolvimento de Reconhecedores Sintáticos Baseado em Autômatos Adaptativos**. II Simpósio Brasileiro de Linguagens de Programação - SBLP97, pp. 139-150, Campinas, 1997.

PEREIRA, J.C.D. **Ambiente integrado de desenvolvimento de reconhecedores sintáticos, baseado em autômatos adaptativos**. Dissertação de Mestrado, USP, São Paulo, 1999.

PETRI, C.A. **Kommunikation mit Automaten.** Bonn: Institut für Instrumentelle Mathematik,Schriften des IIM Nr. 2, German, 1962.

PISTORI, H.; NETO, J.J. AdapTree - **Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas**. Anais Conferência Latino Americana de Informática - CLEI 2002, Montevideo, Uruguai, Novembro, 2002.

PISTORI, H. **Tecnologia Adaptativa em Engenharia de Computação: Estado da Arte e Aplicações**. Tese de Doutorado, USP, São Paulo, 2003.

PISTORI, H.; NETO, J.J.; COSTA, E.R. **Utilização de Tecnologia Adaptativa na Detecção da Direção do Olhar**. SPC Magazine, v.2., n.2, pp.22-29, Lima, Perú, Maio, 2003.

PISTORI H. et al. **Adaptive finite states automata and genetic algorithms: merging individual adaptation and population evolution**. International Conference on Adaptive and Natural Computing Algorithms (ICANNGA), Springer Verlag editor, pp. 333-336, Coimbra University, Coimbra, Portugal, 21 - 23 march 2005.

PYTHON. **Linguagem de programação e manuais**. Python Software Foundation. Disponível em: http://www.python.org/, Janeiro, 2007.

QUARTEL, D**. Actions Relations – Basic design concepts for behaviour modelling and refinement.** Ph.D Thesis Twente University, Netherlands, 1997.

RDF, **Resource Description Framework**, in http://www.w3.org/RDF/, in Janeiro 2007.

ROCHA, R.L.A. **Um método de escolha automática de soluções usando tecnologia adaptativa.** Tese de Doutorado, USP, São Paulo, 2000.

ROCHA, R.L.A.; NETO, J.J**. Uma proposta de método adaptativo para a seleção automática de soluções**. Proceedings of ICIE Y2K - International Congress on Informatics Engineering, Buenos Aires, 2000.

ROCHA, R.L.A. **Uma proposta de uso de tecnologia adaptativa para simulação de redes neurais em um dispositivo computacional**. In: IX Encuentro Chileno de Computación 2001, Punta Arenas. Proceedings of the Encuentro Chileno de Computación. Punta Arenas: Universidad de Magallanes, v. CD-ROM, pp. 1-9, 2001.

ROCHA, R.L.A.; NETO, J.J. **Construção e Simulação de Modelos Baseados em Autômatos Adaptativos em Linguagem Funcional**. Proceedings of ICIE 2001 - International Congress on Informatics Engineering, Buenos Aires: Computer Science Department - University of Buenos Aires, v. CD-ROM, pp. 509-521, 2001.

RUBINSTEIN, R.; SHUTT, J.N. **Self-Modifying Finite Automata.** Worcester, Massachusetts, December 1993.

RUBINSTEIN, R.; SHUTT, J.N. **Self-modifying finite automata.** In: *Proceeding of the 13th IFIP World Computer Congress*. Amsterdam: North-Holland: [s.n.], 1994.

RUBINSTEIN, R.; SHUTT, J.N. **Self-modifying finite automata: An introduction.** *Information Processing Letters*, v. 56, n. 4, p. 185.190, 1995.

SANTOS, A.C. **Introdução a Metadados**. Revista Cientefico, Ano II, Volume II. Salvador-BA, 2003

SHETH, A.; RAMAKRISHNAN, C.; THOMAS, C., **Semantics for the Semantic Web: The Implicit, the Formal and the Powerful**. International Journal on Semantic Web & Information Systems, 2005.

SHUTT, J.N. **Recursive Adaptable Grammars.** 1993.

SHUTT, J.N. **Self-Modifying Finite Automata - Power and Limitations.** Worcester, Massachusetts, 1995.

SINDEREN M. et al. **A design model for open distributed processing systems**. Computer Networks and ISDN Systems, pp. 1263-1285, 1995.

SOUSA, M.A.A.; HIRAKAWA, A.R. **Robotic mapping and navigation in unknown environment using adaptive automata**. International Conference on Adaptive and Natural Computing Algorithms (ICANNGA), Springer Verlag editor, pp 345-348, Coimbra University, Coimbra, Portugal, 21 - 23 march 2005.

SOUSA, M.A.A.; HIRAKAWA, A.R.; NETO, J.J. **Adaptive automata for mobile robotic mapping**. Proceedings of VIII Brazilian Symposium on Neural Networks - SBRN'04. São Luís/MA - Brazil. September 29 - October 1, 2004a.

SOUSA, M.A.A.; HIRAKAWA, A.R.; NETO, J.J. **Adaptive automata for mapping unknown environments by mobile robots**. Lecture Notes in Artificial Intelligence.

C. Lemaître, C. A. Reyes, J. A. González (Eds.): Advances in Artificial Intelligence - IBERAMIA 2004, Springer-Verlag, pp 562-571, 2004b.

SOUZA, W.L. et al. **Design de Aplicações Multimídia Distribuídas (DAMD**, Anais do II Seminário Franco-Brasileiro em Sistemas Distribuídos, Fortaleza – CE, Novembro 1997.

STAD. **State-Charts Adaptativos**. em *www.pcs.usp.br/lt*a , visitado em Janeiro de 2007.

TESTBED Studio Tool, in http://www.bizzdesign.com. Agosto 2005.

VON ATZINGEN, Maria Cristina. **A história dos brinquedos: para as crianças conhecerem e os adultos se lembrarem**. São Paulo: Allegro, 2001.

APÊNDICE

A. Descrição do Modelo Lógico

Este apêndice tem por objetivo a descrição das classes que compõe o Modelo Lógico definido neste trabalho. Tal documento encontra-se organizado em 4 seções: Especificação de Classes do Pacote Camada de Dispositivo, Especificação de Classes do Pacote Camada Subjacente, Especificação de Classes do Pacote Camada Adaptativa e Especificação de Classes do Pacote Camada de Memória.

A.1. Descrição de classes do Pacote da Camada de Dispositivo

As classes do Pacote Camada de Dispositivo são responsáveis pela definição dos elementos básicos de um determinado dispositivo adaptativo dirigido por regras. Este pacote possui classes que representam os elementos que determinam os tipos de componentes, os tipos de conexões, os tipos de atributos e a descrição de um determinado dispositivo adaptativo dirigido por regras e é estruturado por 4 classes (*Dispositivo*, *Tipo de Componente*, *Tipo de conexão* e *Tipo de Atributo*), descritas a seguir e ilustradas na Figura 60.

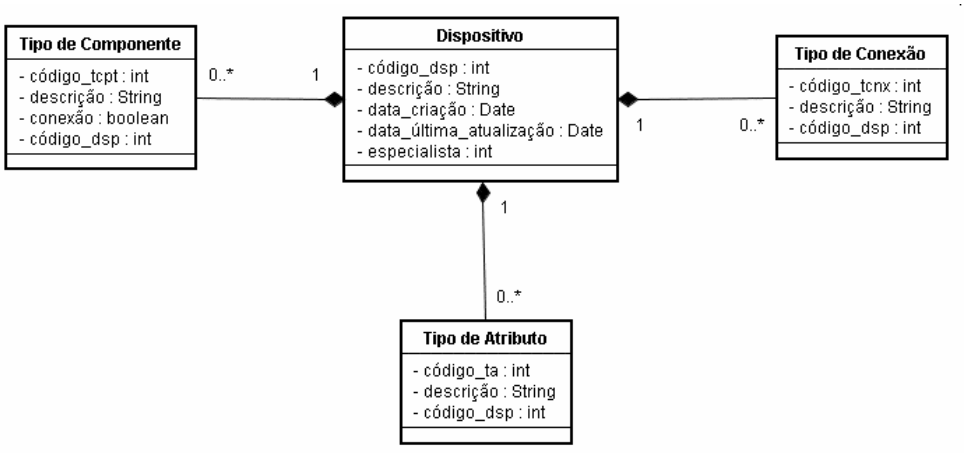


Figura 60. Diagrama de classes da Camada de Dispositivo

A.1.1. Classe Dispositivo

A classe *Dispositivo* é formada por elementos que representam as propriedades de um dispositivo: descrição do dispositivo, nome do especialista que

definiu o dispositivo, data de definição e data da última atualização realizada em sua estrutura.

A classe *Dispositivo* é constituída pelos seguintes atributos:

- *código_dsp*: atributo do tipo contador de inteiros - gera um número seqüencial para identificar um novo objeto criado - e identifica um novo dispositivo configurado;
- *descrição*: atributo do tipo texto que identifica a descrição do dispositivo definido;
- *data_criação*: atributo do tipo data que identifica a data de criação de um dispositivo. Ao ser inserido um novo dispositivo deve ser gravada a data atual do sistema neste atributo;
- *data_última_atualização*: atributo do tipo data que armazena a data da última atualização realizada no referido dispositivo. Ao ser modificado um atributo da classe dispositivo, ou das classes tipo componente, conexão ou atributo, deve ser atualizado também este atributo com informações referentes à data de atualização;
- *especialista*: atributo do tipo texto que identifica o nome do especialista responsável pela configuração do referido dispositivo.

A.1.2. Classe Tipo de Componente

A classe *Tipo de Componente* é organizada por elementos que representam as informações referentes aos tipos de componentes - elementos físicos de um dispositivo, por exemplo, estados finais e não finais de um autômato de estados finitos. Esta classe é constituída pelos seguintes atributos:

- *código_tcpt*: atributo do tipo contador de inteiros que identifica cada um dos tipos de componentes pertencentes à classe;
- *código_dsp*: atributo do tipo inteiro que representa o dispositivo com o qual o componente está associado;
- *descrição*: atributo do tipo texto que identifica a descrição do tipo de componente;
- *conexão*: atributo do tipo lógico que identifica se o componente está associado a outros componentes. Este atributo serve para indicar que

um componente pode pertencer a uma hierarquia de componentes, ou seja, um determinado componente pode ter um agrupamento de outros componentes que estão hierarquicamente relacionados e ele.

A.1.3. Classe Tipo de Conexão

A classe *Tipo de Conexão* é estruturada por elementos que representam o tipo de ligação existente entre os componentes que fazem parte da especificação de uma aplicação, por exemplo, as transições de um autômato de estados finitos. A classe *Tipo de Conexão* possui os seguintes atributos:

- *código_tcnx*: atributo do tipo contador de inteiros que identifica cada um dos tipos de conexão possíveis para um dispositivo;
- *descrição*: atributo do tipo cadeia de caracteres que representa a descrição do tipo de conexão;
- *código_dsp*: atributo do tipo inteiro que associa o tipo de conexão ao seu respectivo dispositivo.

A.1.4. Classe Tipo de Atributo

A classe *Tipo de Atributo*[2] representa os tipos de atributos, que podem ser associados aos componentes ou conexões de um determinado dispositivo, por exemplo, a quantidade de fichas associadas a um local ou transição de uma Rede de Petri. Esta classe é responsável por armazenar informações referentes ao tipo de informação associada a um determinado componente ou conexão de um dispositivo. Esta classe é constituída pelos seguintes atributos:

- *código_ta*: atributo do tipo contador de inteiros que identifica cada um dos tipos de atributos pertencentes a classe;
- *descrição*: atributo do tipo cadeia de caracteres que representa a descrição do tipo de atributo;
- *código_dsp*: atributo do tipo inteiro que relaciona o tipo de atributo ao seu respectivo dispositivo.

[2] Diferenciar definição de Tipos de Atributo da palavra atributo que é utilizada no texto relacionada ao conceito de definição de Classe em Orientação a Objetos.

A.2. Descrição de classes do Pacote da Camada de Especificação Subjacente

A Camada de Especificação Subjacente é responsável por representar as especificações das aplicações modeladas por um projetista e é representada pelo Pacote da Camada de Especificação Subjacente organizado estruturalmente pelas seguintes classes: *Projeto, Componente, Atributo de Componente, Conexão e Atributo de Conexão*. Tais classes podem ser visualizadas graficamente na Figura 61 e, na seqüência, são detalhadas a sua estruturação.

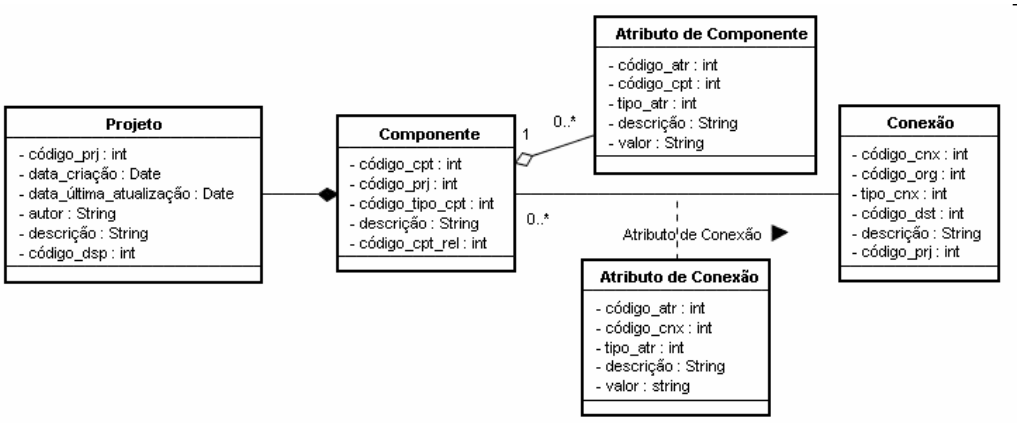


Figura 61. Diagrama de classes da Camada de Especificação Subjacente

A.2.1. Classe Projeto

A classe *Projeto* é responsável por armazenar as propriedades elementares de um projeto e é formada da seguinte maneira:

- *código_prj*: atributo do tipo contador de inteiros que identifica cada novo projeto definido na estrutura. Ao se definir um novo projeto, este recebe uma identificação única que é utilizada para associar o projeto aos demais elementos (componentes, conexões, atributos, etc..) que o compõem.
- *descri*ção: atributo do tipo cadeia de caracteres responsáveis por armazenar a descrição que identifica o projeto;
- *data_criação*: atributo do tipo data que armazena a data de criação do projeto;
- *data_última_atualização*: atributo do tipo data que armazena a data da última atualização realizada no projeto. Ao realizar uma atualização ou

mudança na especificação da aplicação (componentes, conexões, atributos, etc..) deve ser atualizado este atributo.

- *autor*: atributo do tipo cadeia de caracteres que guarda o nome do projetista responsável pela especificação da aplicação;
- *código_dsp*: atributo do tipo inteiro utilizado para armazenar o código do dispositivo utilizado para projetar a aplicação. Este atributo serve para associar as demais classes desta camada às classes da camada de dispositivos.

A.2.2. Classe Componente

A classe *Componente* é utilizada para representar as informações dos componentes que constituem uma determinada especificação. Ao definir o comportamento de uma aplicação, são utilizados diversos elementos que representam suas ações ou estados num certo momento. Um componente pode representar um estado no comportamento de um autômato de estados finitos ou um *(local/transição)* de uma Rede de Petri; ou ainda, uma ação, uma interação ou conectores (*and* ou *or*) no dispositivo ISDL. Sua estrutura é organizada como segue:

- *código_cpt*: atributo do tipo contador de inteiros que identifica cada componente adicionado à estrutura. Ao se definir um novo componente para uma aplicação, este recebe um código que permitirá o seu relacionamento com as demais classes deste ou de outros pacotes;
- *código_prj*: atributo do tipo inteiro que relaciona o componente definido ao projeto em que pertence;
- *código_tipo_cpt*: atributo do tipo inteiro que identifica o tipo de um componente. Ao se definir um componente, é necessário informar qual o tipo representado com base nos objetos definidos na Classe Tipos de Componente;
- *descrição*: atributo do tipo cadeia de caracteres que armazena a descrição de um componente;
- *código_cpt_rel*: atributo tipo inteiro que associa o respectivo componente a um outro componente pertencente a uma hierarquia de

componentes[3]. Este atributo permite criar um agrupamento de componentes e faz parte da definição de um outro componente;

A.2.3. Classe Atributo de Componente

A classe *Atributo de Componente* armazena informações dos atributos que pertencem a um componente. Ao definir um componente, o projetista pode associar valores que serão manipulados durante a execução do sistema, ou domínios de valores que podem ser manipulados durante a fase de análise ou de tradução para representação física. A seguir, são descritos os atributos da referida classe:

- *código_atr*: atributo do tipo contador de inteiros que identifica cada um dos atributos definidos para um componente;
- *código_cpt*: atributo do tipo inteiro que armazena o código do componente ao qual o atributo pertence;
- *tipo_atr*: atributo do tipo inteiro que relaciona o tipo de atributo definido para um determinado componente;
- *descrição***:** atributo do tipo cadeia de caracteres que armazena a descrição do Atributo de Componente;
- *valor:* atributo do tipo cadeia de caracteres que armazena informações de valores associados aos Atributos de Componentes ou outras informações definidas pelo projetista da aplicação. Tais informações são definidas pelo especialista no momento da definição de dispositivos adaptativos de acordo com as características conceituais do dispositivo que está sendo definido.

A.2.4. Classe Conexão

A classe *Conexão* representa as ligações existentes entre os componentes de uma aplicação. Por exemplo, ao se definir um comportamento utilizando o dispositivo de Autômato de Estados Finitos, é necessário representar as transições (conexões) existentes entre os estados (componentes) da aplicação. Esta classe é representada a seguir:

[3] Como exemplo, pode-se citar a definição de subcomportamentos para comportamentos ISDL.

- *código_cnx*: atributo do tipo contador de inteiros que identifica cada conexão que é adicionada ao projeto. Ao ser inserida a ligação entre dois componentes de uma determinada especificação é inserido também um registro contendo suas informações. O atributo *código_cnx* é utilizado, posteriormente, para permitir o relacionamento de uma conexão com as demais classes do pacote especificação e também é responsável por permitir o acoplamento das funções e ações adaptativas;
- *código_prj*: atributo tipo inteiro que relaciona a conexão ao seu respectivo projeto;
- *tipo_cnx*: é um atributo do tipo inteiro responsável por armazenar o código do Tipo de Conexão que está sendo especificado;
- *código_org*: toda conexão possui dois componentes relacionados: origem e destino que definem os componentes que serão relacionados por meio de uma conexão. O atributo *Cod_Org* é do tipo inteiro e serve para armazenar o código do componente de origem de uma conexão;
- *código_dst*: atributo do tipo inteiro que indica o componente de destino de uma conexão;
- *descrição:* atributo do tipo cadeia de caracteres que é utilizado para representar informações adicionais (para) a uma determinada conexão.

A.2.5. Classe Atributo de Conexão

A classe *Atributo de Conexão* armazena as informações de valores relacionados a uma conexão. Ao definir um atributo de conexão, o projetista associa valores que serão manipulados por uma conexão durante a fase de execução, de simulação, etc. Esta classe armazena os atributos associados a uma conexão como, por exemplo, os símbolos relacionados às transições no dispositivo de autômato de estados finitos. Os atributos desta classe têm a seguinte funcionalidade:

- *código_atr*: atributo do tipo contador de inteiros que identifica cada um dos atributos definidos por um projetista;
- *código_cnx*: atributo do tipo inteiro que representa a conexão que está relacionada ao referido atributo;

- *tipo_atr*: atributo do tipo inteiro que representa qual o tipo de atributo definido para uma determinada conexão;
- *descrição***:** atributo do tipo cadeia de caracteres que armazena a descrição ou o atributo definido para uma conexão;
- *valor:* atributo do tipo cadeia de caracteres que armazena informações de valores associados aos atributos ou outras informações definidas pelo projetista da aplicação. Tais informações são oriundas das características dos elementos conceituais do dispositivo, definidas pelo especialista durante a definição dos dispositivos adaptativos.

A.3. Descrição de Classes do pacote Camada de Especificação Adaptativa

Ao desenvolver uma especificação utilizando-se dos recursos da tecnologia adaptativa, o projetista pode acoplar funções e ações adaptativas a sua especificação. Tais estruturas são representadas no modelo proposto por meio da camada adaptativa. A camada adaptativa é representada pelas classes que constituem o pacote Camada Adaptativa. Tais classes são exemplificadas na Figura 62 e definidas a seguir.

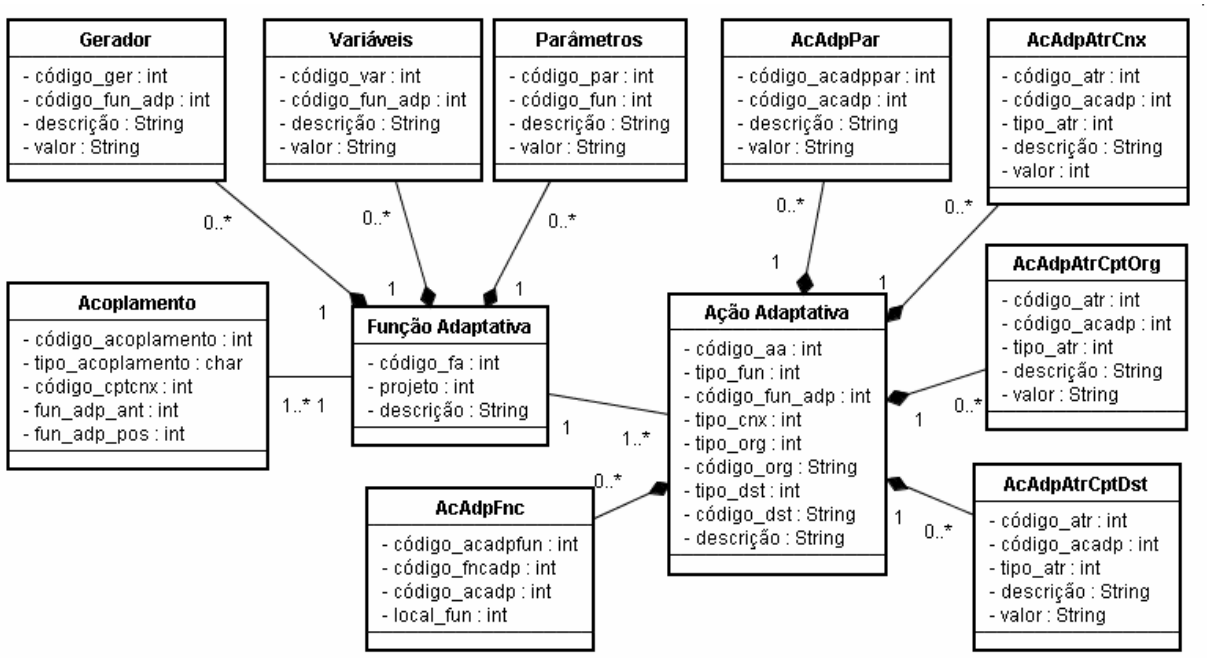


Figura 62 . Diagrama de classes da Camada de Especificação Adaptativa

A.3.1. Classe Função Adaptativa

A classe *Função Adaptativa* representa as informações referentes à definição de uma Função Adaptativa. Ao ser associada uma função adaptativa a um componente ou a uma conexão, devem ser definidas as suas propriedades. Esta classe é representada pelos seguintes atributos:

- *código_fa*: atributo contador do tipo inteiro que identifica a Função Adaptativa modelada;
- *projeto*: atributo tipo inteiro que relaciona a referida Função Adaptativa ao projeto a que esta pertence;
- *descrição*: atributo do tipo cadeia de caracteres que armazena a descrição da função adaptativa.

A.3.2. Classe Ação Adaptativa

Ao adicionar uma função adaptativa, o projetista deve definir um conjunto de ações que a ela estarão associadas e que permitirão a modificação do comportamento de uma aplicação. As ações adaptativas são representadas na estrutura de dados proposta pela classe Ação Adaptativa e os seus respectivos atributos:

- *código_aa*: atributo contador do tipo inteiro que identifica a ação adaptativa;
- *código_fun_adp*: atributo tipo inteiro que relaciona a ação adaptativa a sua respectiva função;
- *tipo_fun*: atributo do tipo inteiro que indica o tipo de ação adaptativa (1-Consulta, 2-Exclusão e 3-Inclusão);
- *tipo_cnx*: atributo do tipo inteiro que relaciona o tipo de conexão de uma ação adaptativa;
- *tipo_org:* atributo do tipo inteiro que relaciona qual tipo de componente é o da classe *Ação Adaptativa* com a classe *Tipo Componente* e serve para indicar o tipo de componente que é consultado, inserido ou removido;

- *código_org:* atributo do tipo inteiro que indica o código do componente de origem da ação adaptativa.
- *tipo_dst:* atributo do tipo inteiro que se relaciona com a classe *Tipo Componente* e serve para indicar o tipo de componente consultado, inserido ou removido;
- *código_dst:* atributo do tipo inteiro que indica o código do componente de destino da ação adaptativa.
- *descrição*: atributo do tipo cadeia de caracteres, usado para descrever a ação adaptativa.

A.3.3. Classe Acoplamento

A classe *Acoplamento* permite realizar a associação da camada adaptativa à camada subjacente. Ao definir objetos, nesta classe é realizado o acoplamento das ações adaptativas aos componentes ou conexões aos quais pertencem. A não existência de objetos nesta classe para um determinado projeto, indica que a especificação produzida não agrega os recursos providos pela camada adaptativa e a mesma passa a ser interpretada somente pelo dispositivo subjacente. A classe Acoplamento é composta pelos seguintes atributos:

- *código_acoplamento*: atributo do tipo contador de inteiros que identifica uma determinada associação de Função Adaptativa ao respectivo componente ou conexão;
- *tipo_acoplamento*: atributo do tipo inteiro que informa se a função adaptativa está relacionada com um componente ou uma conexão. Se for definido o valor ‘C’, a função estará relacionada a um componente, caso contrário será o valor ´X´, e a função estará relacionada a uma conexão;
- *código_cptcnx*: atributo do tipo inteiro que armazena o código do componente ou conexão relacionado à função adaptativa;
- *fun_adp_ant*: atributo do tipo inteiro que armazena o código da função adaptativa anterior;
- *fun_adp_pos*: atributo do tipo inteiro que armazena o código da função adaptativa posterior.

A.3.4. Classe Gerador

A classe *Gerador* representa contadores, utilizados durante a execução das funções adaptativas no núcleo adaptativo, que definem o código de novos componentes adicionados ao comportamento de uma aplicação. Tais contadores possibilitam que seus valores sejam acrescidos e, desta forma, permitem que novos componentes sejam adicionados ao comportamento da aplicação a qual pertencem. Os elementos constituintes desta classe são:

- *código_ger*: atributo do tipo contador de inteiros que identifica cada um dos geradores na estrutura proposta;
- *código_fun_adp*: atributo do tipo inteiro que relaciona o gerador a sua respectiva função adaptativa;
- *descrição*: atributo do tipo cadeia de caracteres que descreve o gerador;
- *valor*: atributo do tipo cadeia de caracteres que armazena o valor do gerador durante a execução da aplicação no núcleo adaptativo.

A.3.5. Classe Parâmetros

A classe *Parâmetros* armazena objetos que representam valores a serem passados a uma função adaptativa no momento da execução. Tal classe é estruturada da seguinte forma:

- *código_par*: atributo do tipo inteiro que identifica cada um dos parâmetros pertencentes a uma determinada função;
- *código_fun_adp*: atributo do tipo inteiro que relaciona o parâmetro a sua respectiva função adaptativa;
- *descrição*: atributo do tipo cadeia de caracteres que descreve ou nomeia o referido parâmetro;
- *valor*: atributo do tipo cadeia de caracteres que armazena o valor passado para um parâmetro durante a execução da aplicação.

A.3.6. Classe Variáveis

A classe *Variáveis* armazena objetos que representam valores que são consultados pelas ações adaptativas elementares de consulta durante a execução. Tal classe é estruturada da seguinte forma:

- *código_var*: atributo do tipo inteiro que identifica cada uma das variáveis pertencentes a uma determinada função;
- *código_fun_adp*: atributo do tipo inteiro que relaciona a variável a sua respectiva função adaptativa;
- *descrição*: atributo do tipo cadeia de caracteres que descreve ou nomeia a variável;
- *valor*: atributo do tipo cadeia de caracteres que armazena o valor que foi obtido durante uma ação adaptativa elementar de consulta.

A.3.7. Classe Função Adaptativa de Ação Adaptativa

Ao ser especificada uma ação adaptativa, podem ser associadas a mesma uma determinada função adaptiva. Para tal foi definida a classe *Função Adaptativa de Ação Adaptativa (AcAdpFun)* que serve para representar tais informações. Essa classe é constituída pelos seguintes atributos:

- *código_acadpfun*: atributo do contador de inteiros que identifica cada um dos funções adaptativas de ações adaptativas associadas a uma determinada ação adaptativa;
- *código_acadp*: atributo do tipo inteiro que relaciona a função adaptativa a respectiva ação adaptativa;
- *código_funadp*: atributo do tipo cadeia de caracteres que descreve o função adaptativa relacionada a uma determinada ação adaptativa;
- *local_fun*: atributo do tipo inteiro que indica se a função adaptativa será do tipo 0 (anterior) ou 1 (posterior).

A.3.8. Classe Parâmetros de Ação Adaptativa

Ao ser especificada uma ação adaptativa, podem ser adicionadas novas funções adaptativas. Sendo assim, ao se definir uma ação adaptativa e esta for do tipo função adaptativa, deverão ser definidos os parâmetros a ela relacionados. A classe *Parâmetros de Ação Adaptativa (AcAdpPar)*, descrita abaixo, representa estes valores:

- *código_acadppar*: atributo do contador de inteiros que identifica cada um dos parâmetros adicionados a uma determinada ação adaptativa;
- *código_acadp*: atributo do tipo inteiro que relaciona o parâmetro de função adaptativa correspondente a uma ação adaptativa;
- *descrição*: atributo do tipo cadeia de caracteres que descreve o parâmetro de ação adaptativa;
- *valor*: atributo do tipo cadeia de caracteres que armazena o valor do parâmetro definido para uma função, descrito por uma ação adaptativa.

A.3.9. Classe Atributo de Conexão de Ação Adaptativa

Ao especificar uma ação adaptativa, faz-se necessário também especificar a conexão e os atributos da conexão relacionados a esta estrutura. A classe *Atributo de Conexão de Ação Adaptativa (AcAdpAtrCnx)* tem por objetivo representar os atributos de conexão, e possui os seguintes atributos:

- *código_atr*: atributo do tipo contador de inteiros que identifica os atributos de conexão definidos para ações adaptativas;
- *código_acadp*: atributo do tipo inteiro que relaciona o referido atributo a sua respectiva ação adaptativa;
- *tipo_atr*: atributo do tipo inteiro que define qual tipo de atributo de conexão está sendo definido;
- *descrição***:** atributo do tipo cadeia de caracteres que armazena informação referente à descrição do referido atributo;
- *valor:* atributo do tipo cadeia de caracteres, responsável por armazenar o valor associado a um determinado atributo.

A.3.10. Classe Atributo de Componente de Origem de Ação Adaptativa

A classe *Atributo de Componente de Origem de Ação Adaptativa (AcAdpAtrCPtOrg)* representa os atributos dos componentes de origem de uma ação adaptativa. Esta classe tem estrutura semelhante à classe apresentada anteriormente, a *AcAdpAtrCnx.*

A.3.11. Classe Atributo de Componente de Destino de Ação Adaptativa

A classe *Atributo de Componente de Destino de Ação Adaptativa (AcAdpAtrCPtDst)* representa os atributos dos componentes de destino de uma ação adaptativa e tem estrutura semelhante às classes *AcAdpAtrCnx e AcAdpAtrCptDst.*

A.4. Descrição de classes do Pacote da Camada de Memória

A camada de memória representa as informações necessárias à execução do ambiente adaptativo. A seguir, são descritas as classes que compõem o pacote Memória: *Cadeia de entrada, Atributos de cadeia de entrada* e *Componentes ativos* e na Figura 63 é apresentada de modo gráfico as referidas classes.

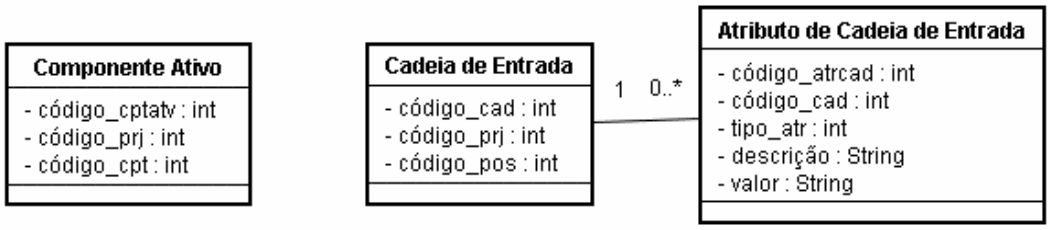


Figura 63. Diagrama de classes da Camada de Memória

A.4.1. Classe Cadeia de Entrada

A classe *Cadeia de Entrada* é responsável por representar os elementos da cadeia (estímulos) de entrada de uma determinada aplicação que será executada no ambiente de execução adaptativa, proposto neste trabalho. Tal classe é constituída dos seguintes atributos:

- *código_cad*: atributo do tipo contador de inteiros que identifica cada um dos objetos pertencentes a esta classe;

- *código_prj*: atributo tipo inteiro que relaciona a referida posição da cadeia de entrada ao projeto a que pertence;
- *código_pos*: atributo do tipo inteiro que identifica a posição do elemento da cadeia de entrada. Ao ser iniciado, o ambiente de execução adaptativa percorre os objetos desta classe e verifica o valor de cada um, de forma semelhante à realizada na varredura da cadeia de entrada do dispositivo correspondente.

A.4.2. Classe Atributo de Cadeia de Entrada

A classe Atributos de *Cadeia de Entrada* é responsável por representar os atributos de cada posição da cadeia de entrada. Se o dispositivo definido for um Autômato de Estados Finitos, por exemplo, é nesta classe que serão definidos os objetos contendo cada um dos símbolos para cada posição da cadeia de entrada. No caso de ser um outro dispositivo, os atributos devem ser definidos conforme as características de cada dispositivo. A seguir, é descrita a estrutura desta classe:

- *código_atr*: atributo do tipo contador de inteiros que identifica cada um dos atributos de cadeia de entrada;
- *código_pos*: atributo tipo inteiro que relaciona o atributo da cadeia de entrada com a sua respectiva posição na classe *Cadeia*;
- *tipo_atr*: atributo do tipo inteiro que relaciona o tipo de atributo que está sendo especificado na cadeia de entrada;
- *descrição*: atributo do tipo cadeia de caracteres que identifica a descrição do Atributo de Cadeia de Entrada;
- *valor*: atributo do tipo cadeia de entrada que armazena o valor do atributo definido pelo usuário no momento da execução, no ambiente de execução adaptativa.

A.4.3. Classe Componente Ativo

A classe *Componente Ativo* é responsável por representar os componentes que estão ativos em um determinado instante de tempo durante a execução da

aplicação, no ambiente de execução adaptativa. Tal classe é constituída dos seguintes atributos:

- *código_cptatv*: atributo do tipo contador de inteiros que identifica cada um dos objetos pertencentes a esta classe;
- *código_prj*: atributo tipo inteiro que relaciona um componente ativo ao seu respectivo projeto;
- *código_prj*: atributo tipo inteiro que indica qual o componente que está ativo. Tal atributo também serve para relacionar o componente a sua respectiva classe *Componente*.