# AdapLib: Uma Biblioteca para Execução de Dispositivos Adaptativos

F. L. Siqueira

***Resumo*— Neste artigo apresenta-se o embasamento teórico e a arquitetura (seguindo o modelo 4+1) da biblioteca para execução de dispositivos adaptativos, *AdapLib*. Essa biblioteca foi criada buscando tratar com fidelidade a teoria de dispositivos adaptativos e, ao mesmo tempo, permitir que ela seja estendida com facilidade para absorver os detalhes específicos de soluções que empreguem esse tipo de dispositivo. Como exemplo disso será apresentado um estudo de caso em que a biblioteca foi usada na criação de uma prova de conceito para o monitoramento e o diagnóstico de problemas em um portal de notícias *online*.**

***Palavras chaves*—Arquitetura (*architecture*), ferramenta de software (*software tool*), Sistemas adaptativos (*adaptive systems*).**

## I. Introdução

Os dispositivos adaptativos são uma proposta de formulação teórica para formalismos baseados em regras auto-modificáveis. Seguindo essa teoria, existem na literatura propostas de dispositivos como autômatos [1], árvores de decisão [2], tabelas de decisão [3], state-charts [4], entre outros. Por mais que as formulações definam e provem o funcionamento desses dispositivos, ainda é necessário implementá-los – de alguma forma – para permitir a sua execução (normalmente, usando uma linguagem de programação).

Entretanto os usuários dessas teorias encontram pouco ou nenhum suporte ferramental para aplicá-las na prática. Por mais que esses usuários decidam fazer uma implementação, muitas vezes ela é específica para um problema, indisponível para a comunidade, ou até mesmo apresenta problemas (não é eficiente, possui defeitos, é limitada conceitualmente, etc), o que inviabiliza uma reutilização e pode gerar novamente a necessidade de criar uma nova implementação para o formalismo em questão.

Considerando esse contexto, neste trabalho é apresentada a biblioteca AdapLib, discutindo principalmente sua arquitetura. Essa biblioteca implementa a teoria de dispositivos adaptativos usando os conceitos de orientação a objetos, buscando ao mesmo tempo fidelidade à teoria, eficiência e extensibilidade. A intenção foi criar uma biblioteca de código aberto que possa ser usada em aplicações (comerciais ou não comerciais) sem a preocupação de como os conceitos são implementados; esses detalhes são tratados pela AdapLib.

F. L. Siqueira é doutorando em Engenharia de Computação na Escola Politécnica da Universidade de São Paulo (POLI-USP), Av. Prof. Luciano Gualberto, Trav.3, n.158, 05508-900 (e-mail: levy.siqueira@usp.br).

Para apresentar essa biblioteca, este artigo está organizado da seguinte forma: na Seção II é apresentada a fundamentação teórica a respeito de dispositivos adaptativos. Em seguida, na Seção III é apresentada a biblioteca AdapLib, discutindo sua aderência à teoria, seus objetivos, os trabalhos relacionados e a sua arquitetura. Na Seção IV é apresentado um estudo de caso em que a biblioteca foi usada e os resultados obtidos. Por fim, na Seção V é apresentada a conclusão e as perspectivas de trabalhos futuros.

## II. Dispositivos Adaptativos

A teoria de dispositivo adaptativo é uma formulação geral de formalismos auto-modificáveis baseados em regras que busca uma maior proximidade com o formalismo não adaptativo original [3]. Dessa forma, há uma clara separação entre a parte não adaptativa, chamada de camada subjacente, e a parte que permite alterar o comportamento do dispositivo de forma dinâmica, chamada de camada adaptativa. Essa formulação pode ser vista como uma generalização da proposta de autômatos adaptativos feita em [1] em decorrência da aplicação de seus conceitos em outras representações [2].

A camada subjacente representa um dispositivo baseado em um conjunto finito de regras. A partir de uma configuração inicial, essas regras são aplicadas em resposta a eventos de entrada, fazendo com que o dispositivo mude a sua configuração. Ao final do processamento dos eventos, o dispositivo atinge uma determinada configuração que pode aceitar ou rejeitar a entrada em questão. Segundo Neto [3] esse dispositivo pode ser formalmente definido como D=(C, R, S, c0, A, O), onde:

- $C$ é o conjunto de possíveis configurações, com $c_0$ sendo a configuração inicial do dispositivo;
- $S$ é o conjunto de eventos válidos para o dispositivo, considerando $\varepsilon \in S$;
- $A \subseteq C$ é o conjunto de configurações que aceitam a entrada; e
- $O$ é o conjunto de símbolos que podem ser saída após a execução de uma determinada regra.
- $R \subseteq C \times S \times C \times O$ é o conjunto finito de regras definidas para o dispositivo. Uma regra $r \in R$ é representada por $r=(c_i, s, c_f, o)$, significando que a partir de uma configuração atual $c_i \in C$, um evento $s \in S$ estimula a passagem para uma configuração $c_f \in C$ e gera o símbolo $o \in O$ na saída.

É importante ressaltar que essa formulação permite que haja mais de uma regra aplicável ao considerar uma determinada configuração atual e um evento de entrada.

Caso o dispositivo possua apenas uma regra possível para cada par configuração atual e evento, ele é dito determinístico; caso contrário, ele é dito não-determinístico. Neste último caso, o dispositivo deve, teoricamente, aplicar cada regra possível em paralelo, obtendo como resposta diversas configurações possíveis. Se a subseqüente aplicação de regras em cada uma dessas possíveis configurações (que podem causar ainda mais não-determinismos) levar a uma configuração de aceitação, o dispositivo então aceita a cadeia, caso contrário a rejeita.

Sobre esse tipo de dispositivo pode ser colocada uma camada adaptativa que permite alterar o conjunto de regras e de configurações. Essa alteração é feita através de ações adaptativas que ficam associadas às regras da camada subjacente e são executadas antes da aplicação da regra (ações anteriores) ou após (ações posteriores). Essas ações adaptativas podem ser de três tipos: de inserção (que inserem regras), de remoção (que removem regras) e de busca (que apenas obtêm regras e dados3).

Ao aplicar ações adaptativas que alteram a camada subjacente se tem, conceitualmente, um novo dispositivo adaptativo. Com isso, a aplicação das ações faz com que se transite por um espaço de dispositivos válidos a partir de um dispositivo inicial. Dessa maneira, um dispositivo adaptativo pode ser definido como ADk=(Dk, AM), considerando k execuções de regras com ações adaptativas (ou passos). No caso, Dk é o dispositivo não adaptativo (camada subjacente) e AM é o mecanismo adaptativo que define as ações adaptativas a serem aplicadas a partir de cada regra da camada subjacente. De uma forma mais detalhada, segundo [3] o dispositivo adaptativo pode também ser formalmente definido como ADk=(Ck, ARk, S, ck, Ak, O, BA, AA) em que:

- $C_k$ é o conjunto de possíveis configurações após a execução de $k$ passos, com $c_k$ sendo a configuração inicial do dispositivo nesse passo;
- $AR_k \subseteq BA \times C_k \times S \times C_k \times O \times AA$ é o conjunto de regras adaptativas;
- $S$ é o conjunto de eventos válidos para o dispositivo, considerando $\varepsilon \in S$;
- $A_k \subseteq C_k$ é o conjunto de configurações que aceitam a entrada no passo $k$;
- $O$ é o conjunto de símbolos que podem ser saída após a execução de uma determinada regra; e
- $BA$ e $AA$ são conjuntos de ações adaptativas anteriores e posteriores, respectivamente (com isso, o mecanismo adaptativo $AM \subseteq BA \times R_k \times AA$, considerando as regras $R_k$ da camada subjacente).

Em geral, as ações adaptativas anteriores e posteriores são organizadas em funções adaptativas que além de agrupar as ações também recebem parâmetros, o que permite um maior reuso das ações adaptativas.

[3] As ações adaptativas de busca não alteram a estrutura da camada subjacente, mas obtêm regras ou dados que são usados pelas ações de inserção e de remoção.

## III. ADAPLIB

A teoria de dispositivos adaptativos define formalmente os elementos que constituem esse tipo de dispositivo e apresenta a forma de executá-los (através de um pseudo-algoritmo [3]). A biblioteca AdapLib, implementada em Java, busca seguir com fidelidade essa teoria ao prover uma implementação aderente ao formalismo original que possa ser usada em aplicações. Dessa maneira, a biblioteca permite representar dispositivos não adaptativos e acoplá-los como camada subjacente de um dispositivo adaptativo. Seguindo a definição, os dispositivos representados na biblioteca definem configurações, regras, eventos e símbolos, enquanto que o mecanismo adaptativo trata das regras, ações e funções adaptativas.

Além de buscar uma proximidade à teoria, um outro objetivo da AdapLib é ser extensível. Desenvolvedores podem implementar outros dispositivos não adaptativos e usar a infra-estrutura da biblioteca para usá-los como camada subjacente de um dispositivo adaptativo e executá-los. Mais que isso, usuários da biblioteca podem estender o formalismo original de um dispositivo para adequar a teoria às suas necessidades práticas. Pode-se, por exemplo, fazer com que ao aplicar uma regra seja executado um código específico (em linguagem de programação), executar um código ao passar para uma determinada configuração, ou até mesmo definir novos elementos para um dispositivo.

A AdapLib, portanto, tem como objetivo prover um ambiente de execução de dispositivos adaptativos que seja extensível e eficiente e que possa ser usado em aplicações.

### A. *Funcionalidades e limitações*

Considerando os objetivos da biblioteca, a seguir são apresentadas as suas principais funcionalidades em sua versão atual (2.0):

- Separação da camada subjacente da camada adaptativa.
- Uma camada adaptativa pode ser usada como camada subjacente para uma outra camada adaptativa (levando em conta a consideração discutida posteriormente nesta seção).
- Possibilidade de definir comportamentos específicos ao chegar a uma determinada configuração ou ao aplicar uma regra.
- Criação de ações adaptativas de remoção que podem considerar qualquer combinação de configuração inicial, configuração final e evento (menos a que remove todas as regras do dispositivo).
- Criação de ações adaptativas de inserção, definindo até chamadas de função adaptativas anteriores e posteriores na regra a ser inserida.
- Possibilidade de definir funções adaptativas:
  - Para cada função pode-se definir ações anteriores e posteriores à execução das ações definidas nas funções.

- Definição de geradores, ou seja, configurações que são criadas ao executar a função (sem limites para o número de geradores).
- Possibilidade de definir parâmetros para a função (sem limite no número de parâmetros) e usá-los nas ações adaptativas definidas no contexto da função.
- As ações adaptativas de inserção e de remoção não têm uma ordem de execução definida.

- Representação e execução de autômatos finitos e autômatos finitos adaptativos determinísticos.
- Permitir a definição ou alteração do comportamento das regras, configurações e eventos, ou até mesmo da execução do dispositivo subjacente ou do dispositivo adaptativo.

Existência de log para facilitar o entendimento da execução do dispositivo.

Ao definir essas funcionalidades pretendeu-se cobrir os principais conceitos da teoria de dispositivos adaptativos, permitindo que um usuário da biblioteca possa representar um dispositivo teórico com fidelidade ao usá-la. Apesar disso, ainda existem algumas considerações e limitações na versão atual da biblioteca. Talvez a principal delas seja a respeito do não determinismo: por enquanto a biblioteca não o permite. Por mais que a existência do não determinismo seja útil em diversas aplicações, existem algumas dificuldades de implementá-lo considerando a existência de uma camada adaptativa. Entre as dificuldades pode-se considerar a necessidade de criação de uma "onda de clones" [5] (ou seja, cópias da estrutura do dispositivo já que ao aplicar as regras adaptativas se tem diferentes dispositivos), a possível ineficiência em relação a tempo e/ou a memória de implementações de algoritmos de backtracking ou paralelismo.

Por mais que a biblioteca atualmente não permita o não determinismo, ainda é possível criar regras que não consomem símbolos de entrada. Entretanto, considerou-se que elas têm baixa prioridade, ou seja, regras que consomem símbolos devem ser aplicadas preferencialmente, caso haja as duas opções. Caso a camada subjacente em questão apresente mais de uma regra possível, a classe responsável pela execução (Executor) usa a primeira regra passada (de uma lista de regras possíveis).

Uma outra limitação é a respeito do uso de ações adaptativas de busca, que atualmente não são definidas. Por causa disso também não é possível definir variáveis (além dos geradores) dentro de uma função adaptativa. O problema, nesse caso, é como tratar a questão da ineficiência inerente de se fazer uma busca por regras e usar os resultados em outras ações adaptativas. Por não ser um problema tão complexo ou grave, essa funcionalidade será possivelmente implementada nas próximas versões.

Seguindo a formulação dos dispositivos adaptativos, uma outra consideração é em relação à possibilidade de existir adaptabilidade em diversos níveis (ou seja, uma camada adaptativa ser considerada como uma camada subjacente para uma outra camada adaptativa). Existem algumas possíveis interpretações de quais são as regras e as configurações da camada adaptativa subjacente A versão atual da biblioteca adota, por simplicidade, que as regras são as regras adaptativas e as configurações são as mesmas da camada subjacente não adaptativa. Mas uma outra possível resposta é que as ações adaptativas são as regras – e nesse caso é necessária uma análise mais cuidadosa para definir o que seriam as configurações. Dessa forma, ainda é necessária uma melhor análise dessa questão do ponto de vista teórico para considerar que a implementação existente atualmente na biblioteca é suficientemente adequada.

Por fim, atualmente está disponível apenas a implementação de autômatos de estados finitos como camada subjacente. Entretanto, considerando a arquitetura adotada – buscando seguir com fidelidade o formalismo original, imagina-se que a implementação de outros tipos de dispositivos como camada subjacente seja facilmente realizada (na Seção III.B.1. é descrito como um desenvolvedor pode fazê-lo).

### *B. Arquitetura*

Para apresentar de uma forma mais técnica a biblioteca AdapLib, será descrita a arquitetura de software idealizada, ou seja, "a organização fundamental de um software, envolvendo seus componentes, a relação entre eles e o ambiente e os princípios que guiam o seu projeto e sua evolução" [6]. No caso será usado o modelo 4+1 para a definição dos pontos de vista relevantes para a arquitetura [7]. Esse modelo propõe 5 pontos de vista:

- O ponto de vista lógico trata principalmente dos requisitos funcionais do software, descrevendo suas abstrações principais e seus relacionamentos. Em um projeto Orientado a Objetos, um dos principais diagramas dessa visão é aquele que representa suas classes.
- O ponto de vista de implementação trata da organização do código fonte, componentes, arquivos de dados e outros artefatos. Dessa forma, esse ponto de vista trata do ambiente de desenvolvimento e de aspectos da gerência de configuração do software.
- O ponto de vista de processo trata das questões de concorrência do software, cobrindo os requisitos não funcionais de desempenho, escalabilidade e taxa de transferência.
- O ponto de vista de implantação trata do mapeamento dos elementos de software nos elementos de hardware. Com isso, são tratadas as questões de instalação, implantação e também de desempenho do software.
- Por fim, o ponto de vista de caso de uso junta os elementos dos outros pontos de vista, representando os cenários de uso do software e abstraindo seus principais requisitos.

Seguindo essa divisão, a seguir serão apresentadas cada um dos pontos de vista considerando as funcionalidades descritas anteriormente4.

[4] Algumas classes da biblioteca são parametrizáveis. Entretanto por simplicidade essas classes serão apresentadas sem seus parâmetros.

*1) Ponto de vista lógico*

A atual implementação da biblioteca possui 37 classes e interfaces que serão apresentadas nesta seção de uma forma simplificada, focando em alguns aspectos principais: a descrição do dispositivo (Fig. 1), as funções adaptativas (Fig. 2) e seus parâmetros (Fig. 3), a execução (Fig. 4) e o autômato como camada subjacente (Fig. 5).

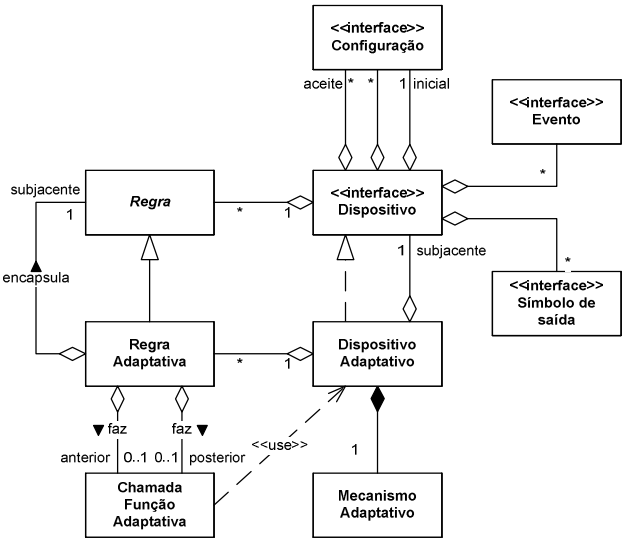


Fig. 1. As classes que tratam do dispositivo e do dispositivo adaptativo.

Na Fig. 1 são apresentados os principais elementos que descrevem um dispositivo e um dispositivo adaptativo. Seguindo a definição apresentada na Seção II, um *Dispositivo* possui *Eventos*, *Configurações*, *Símbolos de saída* e *Regras*. Um *Dispositivo Adaptativo* é também um *Dispositivo*, sendo que as *Regras* são especializadas considerando que elas podem fazer *Chamadas a funções adaptativas*. Além disso, o *Dispositivo Adaptativo* tem um *Mecanismo Adaptativo* que é responsável por tratar das mudanças da estrutura da camada subjacente (usadas pelas ações adaptativas).

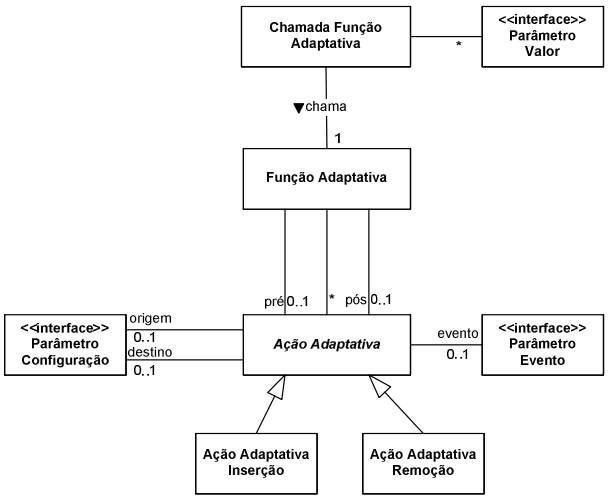


Fig. 2. As classes que tratam das funções adaptativas.

Especificamente sobre a função adaptativa, na Fig. 2 são apresentadas as classes que a representa. As *Chamadas de Função Adaptativas* (que são parte de uma regra adaptativa) recebem parâmetros passados por valor (*Parâmetro Valor*). Com esses valores é então chamada uma *Função Adaptativa* que executa uma *Ação Adaptativa* antes da execução das demais ações (pré) e uma ação que é executada após as demais ações (pós). Essas ações podem ser *Ação Adaptativa Inserção* e *Ação Adaptativa Remoção*. Uma *Ação Adaptativa* define parâmetros que estabelecem padrões para a configuração de origem e de destino e também para o evento (ações de inserção precisam definir todas essas informações, seja com valores ou com referência a valores que serão resolvidas durante a chamada da função, enquanto que as ações de remoção devem definir pelo menos uma dessas informações).

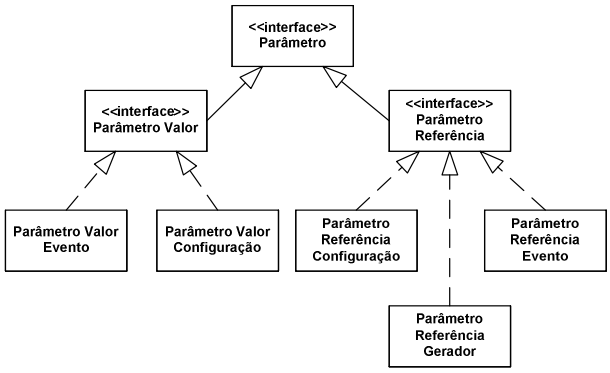


Fig. 3. A hierarquia de classes dos parâmetros.

Uma questão importante desse modelo é em relação aos parâmetros: existe uma hierarquia de tipos de parâmetros que busca evitar que o usuário da biblioteca defina de forma equivocada as funções adaptativas e suas chamadas. Na Fig. 3 é apresentado um diagrama de classes a esse respeito. Todos os parâmetros são do tipo Parâmetro e se têm duas outras hierarquias: se é por valor (*Parâmetro Valor*) ou por referência (*Parâmetro Referência*), ou se trata de configuração (*Parâmetro Configuração*)[5] ou de evento (*Parâmetro Evento*). A idéia dessa diferenciação é que a *Chamada de Função Adaptativa* deve apenas considerar valores, enquanto que as *Ações Adaptativas* podem considerar valores e referências (que são resolvidas durante a execução). Ao definir as ações, entretanto, é fundamental que se diferencie o que é configuração e o que é evento. Dessa forma, caso um desenvolvedor use por engano um parâmetro que representa um evento em uma função adaptativa que deveria receber uma configuração, em tempo de execução haverá um conflito de tipos, sendo gerada uma exceção (e não tentando executar a função adaptativa). Além disso, em tempo de compilação se detecta o uso de variáveis de tipos errados (como, por exemplo, ao tentar criar uma ação adaptativa de inserção em que a configuração inicial é representada equivocadamente como um evento).

[5] Existe um tipo especial de parâmetro de configuração e que também é parâmetro de referência que trata dos geradores (*Parâmetro Gerador*).

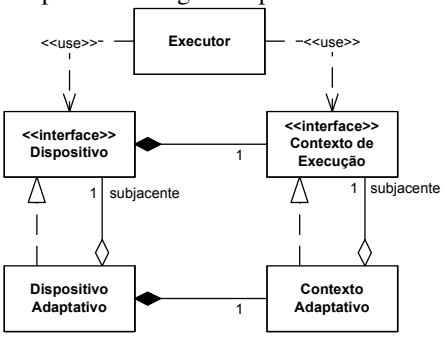

Fig. 4. As classes de execução.

Outra parte fundamental da biblioteca são as classes de execução, apresentadas na Fig. 4. A execução em si é feita pela classe *Executor* que usa um *Contexto de Execução*. Essa separação é brevemente discutida a seguir, na Seção III.B.3. Um exemplo simplificado da execução da biblioteca é apresentado na forma de um diagrama de seqüência na Fig. 8. Buscou-se seguir o pseudo-algoritmo descrito em [3], levando em conta as considerações e limitações da biblioteca. Com isso, a idéia é que o *Executor* chama o *Contexto* para obter as regras e aplicá-las. Por sua vez, o *Contexto* pede para que a *Regra Adaptativa* seja aplicada, fazendo a *Chamada de Função Adaptativa* e aplicando a *Regra* subjacente (não descrito no diagrama, mas usando o *Contexto* subjacente).

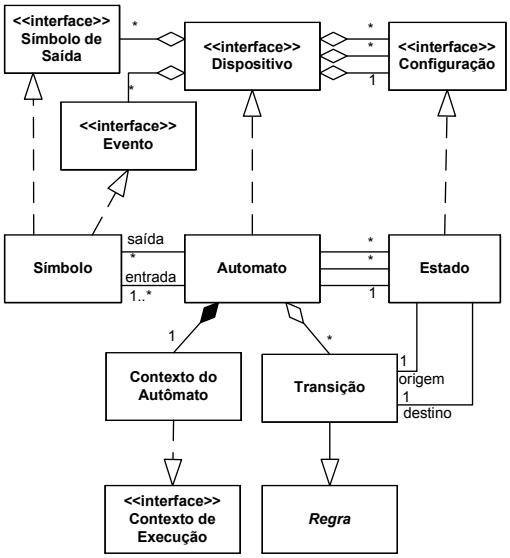

Fig. 5. As classes que tratam do autômato como dispositivo subjacente.

Por fim, na Fig. 5 são apresentadas as classes que tratam da implementação do autômato como dispositivo subjacente. Atualmente a biblioteca disponibiliza apenas esse dispositivo, mas caso seja necessário um outro dispositivo, o desenvolvedor deverá implementar as interfaces e classes abstratas que representam os conceitos de dispositivo e os conceitos associados a ele, ou seja, *Dispositivo*, *Contexto de Execução*, *Regra*, *Configuração*, *Evento* e *Símbolo de Saída*. Por exemplo, no caso do autômato, as classes criadas foram respectivamente *Autômato*, *Contexto do Autômato*, *Transição*, *Estado* e *Símbolo* (implementando ao mesmo tempo *Evento* e *Símbolo de Saída*). O desenvolvedor pode reusar essas classes para implementar outros tipos de autômato como dispositivos, apenas redefinindo e adicionando operações e atributos que sejam necessários.

Aos interessados em maiores detalhes sobre a execução e as classes, recomenda-se olhar o código fonte6.

*2) Ponto de vista de Implementação*

As classes definidas para a biblioteca foram organizadas em 4 pacotes, conforme apresentado no diagrama de pacotes da Fig. 6. O pacote raiz contém as interfaces e classes que definem o que é dispositivo e permitem a sua execução. O pacote Adaptativo contém os elementos referentes a adaptatividade e no seu sub-pacote Função estão as classes e interfaces que tratam das funções adaptativas e a hierarquia de tipos de parâmetros. As classes referentes ao mecanismo subjacente estão organizadas no pacote Subjacente. Por enquanto nesse pacote estão disponíveis apenas as classes que permitem um autômato como dispositivo subjacente.

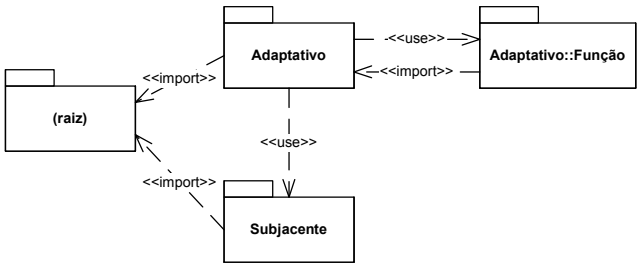

Fig. 6. Os pacotes definidos para organizar as classes da biblioteca.

As únicas dependências externas da biblioteca tratam do *log*, usando o Log4j [8]. Isso é apresentado na forma de um diagrama de componentes na Fig. 7. O artefato "log4j.xml" é necessário para definir qual o nível de *log* permitido e onde as informações de *log* serão apresentadas.

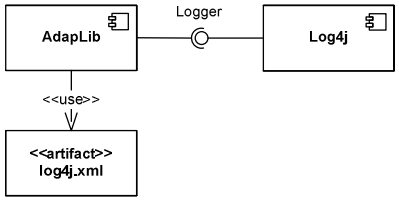

Fig. 7. As dependências externas da *AdapLib*.

*3) Ponto de vista de Processo*

Como atualmente o software utiliza apenas um *thread* de execução e não são especificados requisitos não funcionais de desempenho, este ponto de vista não será detalhado. Em futuras versões em que será tratada a questão do não determinismo, possivelmente haverá uma necessidade de trabalhar com mais de um *thread* de execução. Considerando essa necessidade, decidiu-se separar a execução do dispositivo do dispositivo em si, criando a classe *Contexto de Execução*.

[6] O código fonte da biblioteca *AdapLib* está disponível em: http://www.geocities.com/fabiolevy/projetos/adaplib/.

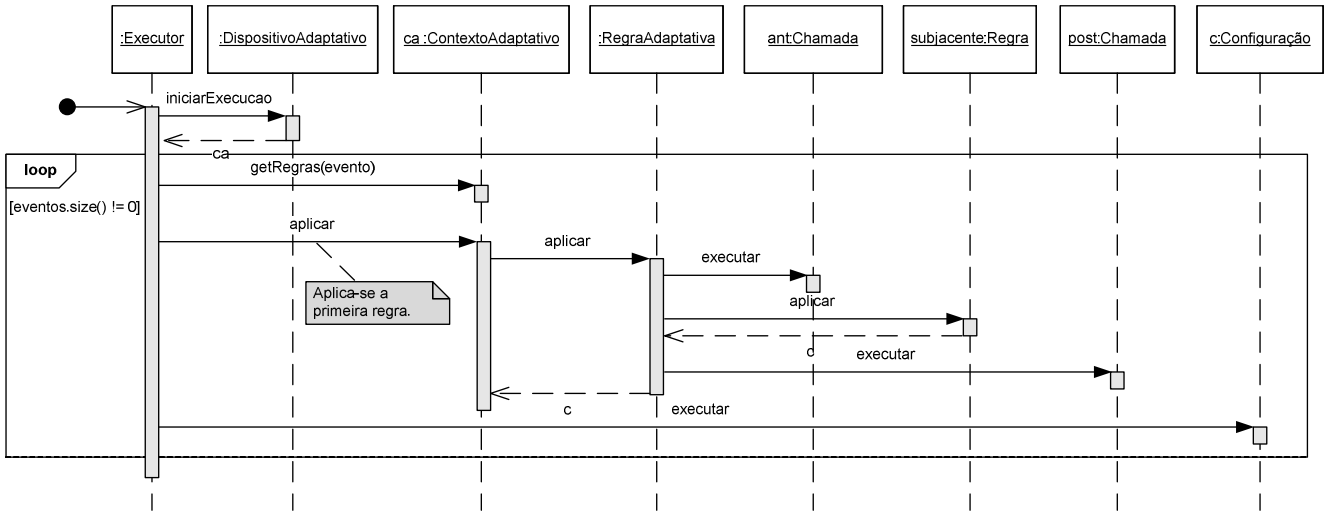


Fig. 8. O cenário de execução de um dispositivo adaptativo.

A intenção é permitir que existam diferentes contextos, um para cada execução possível. Entretanto, como ainda não foram analisados os detalhes de possíveis soluções, provavelmente serão necessárias outras mudanças no modelo em questão.

*4) Ponto de vista de Implantação*

Por ser uma biblioteca, existem poucas questões de implantação. Atualmente a biblioteca é distribuída em duas versões: o código fonte e um arquivo Jar (já que a biblioteca é feita em Java). Esse arquivo Jar permite agregar todos os arquivos compilados em um só, podendo ser usado diretamente por desenvolvedores que trabalham com a biblioteca.

*5) Ponto de vista de caso de uso*

O principal cenário de uso da *AdapLib* é a execução de um dispositivo adaptativo. Seguindo a filosofia da biblioteca, o cenário busca transpor para o software o algoritmo de funcionamento de um dispositivo adaptativo apresentado em [3] – considerando as limitações existentes. Esse cenário de execução é apresentado de forma simplificada na Fig. 8.

O responsável pela execução de um dispositivo é a classe *Executor*. Ela solicita ao *Dispositivo Adaptativo* um *Contexto de Execução Adaptativo* e assim começa a executar o dispositivo, usando as informações desse contexto. Enquanto houver eventos[7], pede-se ao contexto *Regras Adaptativas* que possam ser aplicadas. Caso não haja regras a execução termina, rejeitando a entrada. Se houver regras, aplica-se uma delas (a primeira de uma seqüência). Essa aplicação é feita ao chamar a função adaptativa anterior (se ela estiver definida), ao aplicar a regra subjacente e ao chamar a função adaptativa posterior (se ela estiver definida), nessa ordem. Se ao chamar a função adaptativa anterior à estrutura do dispositivo for alterada de tal forma que a regra que está sendo aplicada não exista mais, interrompe-se a aplicação da regra e o laço é reiniciado. Entretanto, caso a regra atual seja removida ao executar a função adaptativa posterior, nada é feito. Com a aplicação da regra, o dispositivo atinge uma nova configuração, que é executada e que com isso se volta ao laço para aplicar mais regras.

### C. Trabalhos relacionados

Na literatura existem alguns outros trabalhos que propõem criar ambientes de execução para dispositivos adaptativos. O principal deles é o *AdapTools*, que é uma ferramenta para aprendizado, implementação e depuração de dispositivos adaptativos [9]. Nessa ferramenta é possível representar autômatos de pilhas estruturados e autômatos de estados finitos usando uma linguagem própria (linguagem *AdapTools*). Além disso, a ferramenta também provê um compilador para a linguagem *AdapMap*, gerando código na linguagem da ferramenta. Dentre as principais funcionalidades dessa ferramenta está a possibilidade de ver em tempo de execução o funcionamento do autômato, as regras aplicadas, criadas e removidas (organizadas em uma tabela) e a possibilidade de criar rotinas semânticas, executadas após o término do processamento da cadeia. Além disso, é também possível trabalhar com não-determinismos e armazenar as regras em um banco de dados (buscando melhorar a eficiência da execução do autômato).

O *AdapTools* é, portanto, um ambiente mais completo e maduro (desenvolvido desde 2001) que a *AdapLib* – que é apenas uma biblioteca para a execução. Por mais que o *AdapTools* possa ser usado de forma "embutida" em outra aplicação [2], não há muitas informações de como fazê-lo – ou até mesmo de como estendê-la para tratar de detalhes específicos de algumas soluções. Além disso, a ferramenta trata de autômatos e não dispositivos adaptativos em geral. Dessa maneira, a *AdapLib* pretende ser uma biblioteca simples e focada apenas na execução de dispositivos, sendo possível projetar um dispositivo usando as abstrações definidas na própria linguagem de programação – usando para isso os conceitos da orientação a objetos.

Alguns outros trabalhos não apresentam ferramentas, mas provas de conceito que podem ser usadas para simulação de autômatos adaptativos. Werneck [10] analisa a viabilidade de uma implementação usando recursão para a aplicação das

[7] Na realidade a condição para continuar a execução é que ou haja eventos de entrada ou que a configuração atual não seja final e existam regras que não consomem eventos. Isso foi feito para permitir a existência de regras sem eventos e ainda assim não implementar o não determinismo.

regras, ao invés de um laço (que seria a forma tradicional). Para isso foi criado um simulador como prova de conceito que usa uma linguagem específica para definição de autômatos adaptativos (SDMBA) e que permite executar um autômato adaptativo a partir de uma entrada. Esse simulador, entretanto, apresenta algumas limitações frente à teoria de autômatos adaptativos: não é possível trabalhar com ações de remoção e de busca, não é possível passar parâmetros para as funções adaptativas e trata-se apenas de autômatos determinísticos.

De forma mais abstrata, Vega [5] discutiu o projeto de um ambiente de execução de autômatos adaptativos, apresentado como um mini-curso no WTA 2008. Por mais que não seja proposta uma ferramenta em si, uma prova de conceito foi discutida e apresentada – servindo como base para a discussão de problemas e de detalhes de implementação.

## IV. Estudo de Caso

Um dos principais objetivos da biblioteca *AdapLib* é que ela possa ser facilmente absorvida por aplicações que usem os conceitos de dispositivos adaptativos. Para discutir como a biblioteca pode ser usada – em um ponto de vista de implementação – será apresentado um projeto que usou a biblioteca para a criação de uma prova de conceito (POC).

O projeto em questão era a criação de um software para o monitoramento e diagnóstico de problemas em um portal de notícias *online*. O monitoramento precisaria considerar os ambientes de produção e de administração do portal, levando em conta as máquinas existentes (servidores *web*, servidores de aplicação e servidores de banco de dados) e alguns aplicativos específicos. Com as informações obtidas através de sondas existentes nas máquinas e nos aplicativos, era necessário diagnosticar o problema existente, buscando assim direcionar o trabalho da equipe de manutenção do portal. A solução adotada deveria considerar a grande quantidade de acessos ao portal e a possibilidade de inserção de novas máquinas e aplicativos, além da preocupação em se obter uma alta disponibilidade para o portal. Considerando que o objetivo do estudo de caso neste trabalho é apenas analisar o uso da *AdapLib*, não serão aqui discutidos maiores detalhes sobre o contexto e o racional para se chegar a solução aqui apresentada.

A solução adotada para a prova de conceito foi a criação de um autômato de estados finitos adaptativo que usava como alfabeto de entrada as possíveis informações obtidas por sondas nas máquinas e nos aplicativos. A partir dessas entradas, projetou-se um autômato que permitia diagnosticar problemas de desempenho e de exceções causadas nos aplicativos. Na Fig. 9 é apresentada uma parte do autômato que trata do alarme de exceção – a que trata da análise do problema ocorrida no *Servidor de Aplicação 1*. O estado 1 é o estado inicial e, a partir da informação que o problema foi de exceção (símbolo EXCE), o autômato passa ao estado 2. Nesse estado existem transições para as diversas máquinas consideradas, mas na figura há apenas uma transição para o estado 3 caso o problema seja no *Servidor de Aplicação 1* (consumindo o símbolo SA1). Nesse estado é analisado se o *Servidor de Banco de Dados 1* está ativo ou não (representado por "Obter BD1 ativo"). Se não estiver ativo (símbolo INAT), nada é feito (um outro alarme irá avisar a equipe de manutenção que o *Servidor de Banco de Dados 1* está inativo), seguindo para o estado 4 que apenas consome o restante da entrada e vai para o estado final 9 em que nada é feito. Entretanto, caso o *Servidor de Banco de Dados 1* estiver ativo, se verifica em qual página ocorreu a exceção (usando um coletor de nomes definido nas transições do estado 5 e 6 que usam uma codificação da página em binário). Caso ocorram 5 erros na mesma página é emitido um aviso de erro de página, levando ao estado 10 (isso é feito pela função adaptativa 1_conta5). Das próximas vezes que o erro acontecer, ele é desprezado, criando uma transição para o estado 7. Enquanto não ocorrerem os 5 erros, nada é feito, indo ao estado 4 e posteriormente ao estado 9.

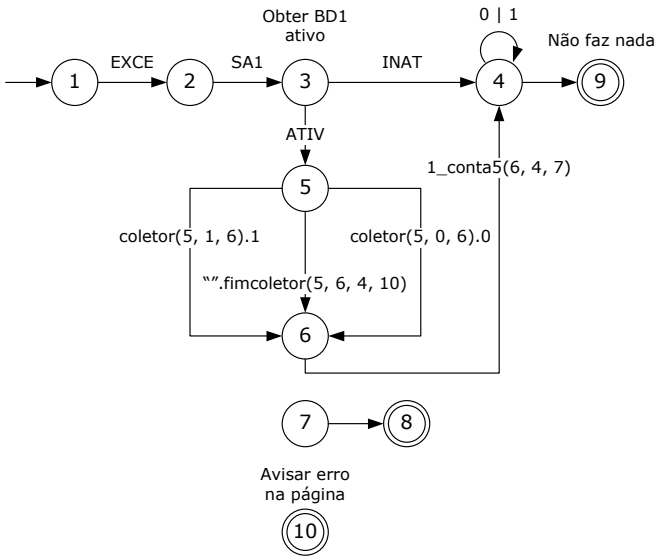


Fig. 9. Parte do autômato de estados finitos adaptativo implementado (os círculos representam estados, as setas transições e os círculos com bordas duplas estados finais).

Para executar o autômato de estados finito projetado foi utilizada a *AdapLib*. Como a biblioteca já possui uma implementação de autômato de estados finitos como dispositivo subjacente (apresentada anteriormente na Seção III.B.1.), só foi necessário adicionar algumas classes para definir o comportamento específico para o projeto. Considerando o contexto, foram necessárias apenas duas mudanças no formalismo de autômato de estados finitos adaptativos: (1) permitir que os estados obtivessem informações e (2) que os estados finais emitissem avisos sobre o problema diagnosticado. A mudança (1) fez com que fosse criado um tipo especial de estado que obtém informações, representado na classe *Estado Obtentor de Informação* e a solução (2) fez com que fossem criados dois outros tipos de estados: um estado final que executava alguma ação na interface homem-computador (*Estado Final Ação*) e um estado final que informava à interface homem-computador o problema diagnosticado (*Estado Final Problema*)[8]. Para representar esses três novos tipos de estado foi necessário apenas criar especializações da classe *Estado* e redefinir a

[8] Isso poderia ser feito ao escrever um símbolo específico na cadeia de saída e processá-lo para gerar a informação adequada na interface homem-computador. Entretanto essa solução exigiria a criação de um símbolo diferente para cada tipo de saída possível – o que, no caso, iria gerar um grande número de novos símbolos. Por causa disso preferiu-se a solução apresentada.

operação *executar*, que permite que um estado tenha um comportamento ao ser alcançado.

```
// Criando o autômato
Automato a = new Automato();

// Colocando uma camada adaptativa no autômato
DispositivoAdaptativo aa =
     new DispositivoAdaptativo(a);

// Estado 1
Estado e1 = new Estado("1");
a.adicionarConfiguracao(e1, true, false); // inicial

// Estado 2
Estado e2 = new Estado("2");
a.adicionarConfiguracao(e2, false, false);

// Estado 3
Estado e3 = new EstadoObtenedorDeInformacao("3",
     "O BD1 está ativo?");
a.adicionarConfiguracao(e3, false, false);

// os outros estados
...

// Criando as transições
a.adicionarRegra(e1, "EXCE", e2); // (1, "EXCE, 2)
a.adicionarRegra(e2, "SA1", e3);  // (2, "SA1", 3)
a.adicionarRegra(e3, "INAT", e4); // (3, "INAT", 4)
a.adicionarRegra(e3, "ATIV", e5); // (3, "ATIV", 5)

// (5, coletor(5, "0", 6)."0", 6)
ArrayList p = new ArrayList();
p.add(new ParametroValorConfiguracao("5"));
p.add(new ParametroValorEvento("0"));
p.add(new ParametroValorConfiguracao("6"));

aa.getMecanismoAdaptativo().adicionarRegraAdaptativa(
     new ChamadaFuncaoAdaptativa(coletor, p),
     e5, "0", e6,
     null);

// as demais transições
...
```

Fig. 10. Fragmento de código exemplificando o uso da *AdapLib* – sublinhado está a referência a um objeto com a função adaptativa do coletor.

Após criar essas novas classes para permitir uma comunicação diferenciada com a interface homem-computador, foi necessário passar para a linguagem de programação o autômato e as funções adaptativas projetadas. Como exemplo disso, uma parte desse código em Java está apresentada na Fig. 10, sublinhando uma variável que referencia a função adaptativa para o coletor de nomes (na transição do estado 5 para o 6)[9]. Cada estado é representado por um objeto e as transições entre eles são adicionadas à camada subjacente (caso não haja chamadas de funções adaptativas) ou à camada adaptativa (que se encarrega de registrar na camada subjacente a regra subjacente).

Uma parte da função adaptativa que trata do coletor de nomes é apresentada na Fig. 11. A função também é um objeto ao qual se deve atrelar as ações adaptativas que sejam necessárias.

[9] Por simplicidade e consistência à descrição arquitetural apresentada (que não representou os parâmetros das classes), o código é apresentado sem os recursos de *Generics* do Java.

```
FuncaoAdaptativa coletor =
     new FuncaoAdaptativa("coletor");
coletor.setGeradores(1);

// - (n0, n1, n2)
AcaoAdaptativaRemocao remocao =
     new AcaoAdaptativaRemocao(
          new ParametroReferenciaConfiguracao(0),
          new ParametroReferenciaEvento(1),
          new ParametroReferenciaConfiguracao(2));
coletor.adicionarAcao(remocao);

// +(n0, n1, g1)
AcaoAdaptativaInsercao insercao =
     new AcaoAdaptativaInsercao(
     new ParametroReferenciaConfiguracao(0),
     new ParametroReferenciaEvento(1),
     new ParametroReferenciaGerador(0));
coletor.adicionarAcao(insercao);

// +(g0, coletor(g0, 0, n2).0, n2)
// Primeiro, definindo os parâmetros da chamada da
// função adaptativa "coletor"
ArrayList parametros = new ArrayList();
parametros.add(new ParametroReferenciaGerador(0));
parametros.add(new ParametroValorEvento("0"));
parametros.add(new ParametroReferenciaConfiguracao(2));

// Criando a ação adaptativa de inserção em si
insercao = new AcaoAdaptativaInsercao(
     new ParametroReferenciaGerador(0),
     new ParametroValorEvento("0"),
     new ParametroReferenciaConfiguracao(2),
     "coletor",  // nome da função adaptativa anterior
                 // que a ser regra inserida chama
     parametros, // os parâmetros da função adaptativa
                 // anterior
     null,       // não há função adaptativa posterior
     null);      // sem parâmetros para a função
                 // adaptativa posterior
coletor.adicionarAcao(insercao);

// as demais ações adaptativas
...
```

Fig. 11. Parte do código que define a função adaptativa "coletor".

Com o autômato completamente representado na linguagem Java foram necessários diversos testes para garantir que o autômato idealizado estava corretamente descrito na linguagem de programação. Além dos erros que aconteceram na passagem para a linguagem de programação, também haviam erros no autômato idealizado – o que dificultou ainda mais a atividade de teste. Entretanto, considerando que o autômato possuía por volta de 50 estados e diversas transições, era esperado que problemas desse tipo acontecessem, principalmente por que o autômato foi apenas executado durante a atividade de testes.

## V. CONCLUSÃO

Neste trabalho foi apresentada a fundamentação teórica e a arquitetura da biblioteca para execução de dispositivos adaptativos *AdapLib*. Essa biblioteca busca seguir com fidelidade a teoria de dispositivos adaptativos, tendo como funcionalidades os principais conceitos dessa teoria. Um outro requisito importante no projeto da biblioteca foi permitir que os usuários a estendam, buscando fazer com que aplicações possam usar a biblioteca com facilidade – mesmo nos casos em que a teoria não possa ser seguida à risca ao considerar questões práticas. Além dessas questões, um outro ponto importante na construção da biblioteca foi a eficiência.

Entretanto esse requisito não funcional não foi discutido neste trabalho, cabendo a trabalhos futuros fazê-lo.

Durante a realização do estudo de caso percebeu-se que uma das dificuldades do uso da biblioteca é na depuração do dispositivo (no caso autômato) projetado. Por mais que a biblioteca permita habilitar durante o desenvolvimento informações detalhadas de *log*, não é possível ter uma representação geral do dispositivo que apresente, em tempo real, as mudanças ocorridas na estrutura do dispositivo. Para isso, ferramentas como o *AdapTools* [9] são mais adequadas, já que há uma representação gráfica da execução do autômato. Nesse sentido é importante ressaltar que o intuito não é que a biblioteca *AdapLib* seja uma ferramenta completa para o projeto e a implementação de dispositivos adaptativos. O objetivo é apenas permitir a execução desse tipo de dispositivo.

Na versão atual da biblioteca alguns conceitos definidos pela teoria de dispositivos adaptativos ainda não foram implementados: ação adaptativa de busca e o não determinismo. Além disso, o único dispositivo disponível pela biblioteca é o autômato (o que não impede que outros dispositivos sejam criados) e ainda é preciso uma análise mais criteriosa para permitir múltiplos níveis de adaptabilidade. Em trabalhos futuros pretende-se tratar dessas limitações e adicionar outras funcionalidades à biblioteca, sejam elas para tratar de aspectos teóricos ou de implementação.

## AGRADECIMENTOS

À Reginaldo Inojosa Filho pelo auxílio na versão inicial da biblioteca e pela realização em conjunto do estudo de caso.

À FAPESP pelo apoio na realização de meu doutorado, durante o qual foi feito este trabalho.

## REFERÊNCIAS

[1] J. J. Neto, "Adaptive automata for context-dependent languages", ACM SIGPLAN Notices, vol. 29, i. 9, pp. 115-124, Sep. 1994.

[2] H. Pistori, "Tecnologia adaptativa em engenharia de computação: Estado da arte e aplicações", Tese de doutorado, orientada por J. J. Neto, Universidade de São Paulo, São Paulo, Brasil, 2003.

[3] J. J. Neto, "Adaptive Rule-Driven Devices - General Formulation and Case Study", Lecture Notes in Computer Science: Implementation and Application of Automata, vol. 2494, pp. 466-470, 2002.

[4] J. R. Almeida, "STAD – Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos", Tese de doutorado, orientada por J. J. Neto, Universidade de São Paulo, São Paulo, Brasil, 1995.

[5] I. S. Vega, "Implementação OO para Autômatos Adaptativos: Impactos das Tomadas de decisão efetuadas durante o Design", Resumo do Mini-curso, Segundo Workshop de Tecnologia Adaptativa (WTA 2008), São Paulo, pp. viii-ix, 2008.

[6] Systems and software engineering - Recommended practice for architectural description of software-intensive systems, ISO/IEC 42010 (IEEE 1471-2000), Jul. 2007.

[7] P. Kruchten, "The 4+1 View Model of Architecture", IEEE Software, vol. 12, no. 6, pp. 42-50, Nov. 1995.

[8] Apache Software Foundation, "Apache Log4J", disponível em: http://logging.apache.org/log4j/, Acesso em 08 de outubro de 2008.

[9] L. Jesus, D. G. Santos, A. A. Castro JR. e H. Pistori, "AdapTools 2.0: Aspectos de Implementação e Utilização", IEEE Latin America Transactions, vol. 5, no. 7, pp. 527-532, Nov. 2007.

[10] N. L. Werneck, "Biblioteca para autômatos adaptativos com regras de transição armazenadas em objetos feita em C++", Segundo Workshop de Tecnologia Adaptativa (WTA 2008), São Paulo, pp.22-26, 2008.

**Fábio Levy Siqueira** é Engenheiro de Computação formando na Escola Politécnica da Universidade de São Paulo (2002) e Mestre em Engenharia Elétrica pela mesma instituição (2005).

Atualmente é doutorando em Engenharia Elétrica e pesquisador na Escola Politécnica da Universidade de São Paulo. Tem experiência na área de Engenharia da Computação, com ênfase em Engenharia de Software, atuando principalmente nas seguintes áreas: engenharia de requisitos, qualidade de software e desenvolvimento distribuído de software.