A Reference Model as Automated Process for Software Adaptation at Runtime

F. J. Affonso, M. C. V. S. Carneiro, E. L. L. Rodrigues and E. Y. Nakagawa

***Abstract*— The development of self-adaptive software (SaS) has specific characteristics compared to traditional one, since it allows that changes to be incorporated at runtime. Automated processes have been used as a feasible solution to conduct the software adaptation at runtime. In parallel, reference model has been used to aggregate knowledge and architectural artifacts, since capture the systems essence of specific domains. However, there is currently no reference model based on reflection for the development of SaS. Thus, the main contribution of this paper is to present a reference model based on reflection for development of SaS that have a need to adapt at runtime. To present the applicability of this model, a case study was conducted and good perspective to efficiently contribute to the area of SaS has been obtained.**

***Keywords*— Self-adaptive Software, Reference Model, Runtime, Automated Process.**

I. Introdução

OS SISTEMAS de software têm ocupado um papel importante na sociedade atual, atuando em diversos segmentos, tais como: instituições públicas e privadas, aeroportos, sistemas de comunicação, entre outros. Esses sistemas devem estar preparados para atuar em situações normais de funcionamento, permanecendo disponíveis 24/7, ou seja, 24 horas por dia e sete dias por semana. Além disso, devem estar preparados para operar em condições adversas, mantendo sua integridade de execução. Assim, características como robustez, confiabilidade, escalabilidade, customização, auto-organização e autoadaptação são cada vez mais presentes nesses sistemas. Essas três últimas características se enquadram em um contexto específico da engenharia de software, que corresponde aos sistemas de software autoadaptativos em tempo de execução. Esses sistemas são considerados específicos, pois permitem que novos recursos (estruturais ou comportamentais) sejam incorporados sem que suas atividades de execução sejam interrompidas [1] [2] [3] [4] [5] [6].

A adaptação de software, quando realizada manualmente, torna-se uma atividade onerosa (tempo e custo) e propensa a erros, devido à injeção involuntária de incertezas pelos desenvolvedores [7] [8] [9] [10]. Para contornar essas adversidades, processos automatizados são fortemente recomendados, pois representam uma alternativa factível para maximizar a velocidade da implementação do software e minimizar o envolvimento de seres humanos (desenvolvedores) na execução das atividades [11].

Em outra perspectiva, um modelo de referência pode ser definido como a decomposição de um problema específico em partes que cooperativamente contribuirão com a solução desse problema. Também pode ser considerado como uma estrutura abstrata ou ontologia de um domínio específico que consiste em um conjunto interligado de conceitos definidos por um especialista (ou grupo de especialistas) a fim de incentivar uma comunicação clara e objetiva entre as partes envolvidas [12]. Além disso, os modelos de referências servem como base para o projeto de arquiteturas de referência, cujo objetivo é facilitar e orientar a concepção de arquiteturas concretas para novos sistemas, novas versões ou extensões de produtos similares. Especificamente em engenharia de software alguns exemplos podem encontrados: (i) *modelo de referência* FORMS (*FOrmal Reference Model for Self-adaptation*), que tem sido utilizado em aplicações para veículos não tripulados e monitoramento (câmeras inteligentes) de congestiomento de trânsito; e (ii) *arquiteturas de referência* para sistemas orientados a serviços como a da fundação IBM [14], arquiteturas de referência para ambientes de engenharia de software [15]. Apesar dos exemplos apontados, não existem modelos de referência, baseados em reflexão, que suportem a automatização das atividades de sistemas de software que necessitam ser adaptados em tempo de execução.

Diante do contexto apresentado, o principal objetivo deste artigo é apresentar um modelo de referência baseado em reflexão para sistemas de software autoadaptativo, que necessitam se adaptar em tempo de execução. Basicamente, este modelo é composto por um conjunto de módulo que trabalham em "linha de montagem" para que a adaptação de software seja realizada automaticamente. A atividade de adaptação é feita utilizando reflexão, que pode ser considerado um recurso importante para inspeção e modificação de software. Como forma de observar a viabilidade deste modelo, um estudo de caso foi conduzido e os resultados podem ser considerados uma importante contribuição para a área de sistemas adaptativos.

Este artigo está organizado da seguinte maneira. A Seção II apresenta os conceitos, definições e trabalhos relacionados. A Seção III mostra o modelo de referência como processo automatizado para sistemas autoadaptativos. Na Seção IV um estudo de caso é apresentado. A Seção V apresenta as discussões e limitações deste artigo. Finalmente, na seção VI são

F. J. Affonso, Univ Estadual Paulista (UNESP), Departamento de Estatística, Matemática Aplicada e Computação, Campus de Rio Claro, Rio Claro - São Paulo, Brasil, frank@rc.unesp.br, affonso.frank@gmail.com.

M. C. V. S. Carneiro, Univ Estadual Paulista (UNESP), Departamento de Estatística, Matemática Aplicada e Computação, Campus de Rio Claro, Rio Claro - São Paulo, Brasil, mcsaenz@rc.unesp.br.

E. L. L. Rodrigues, Universidade de São Paulo (USP), Departamento de Engenharia Elétrica, São Carlos - São Paulo, Brasil, evandro@sc.usp.br.

E. Y. Nakagawa, Universidade de São Paulo (USP), Departamento de Sistemas de Computação, São Carlos - São Paulo, Brasil, elisa@icmc.usp.br.

apresentadas as conclusões e perspectivas de trabalho futuro.

II. Conceitos, definições e trabalhos relacionados

A reflexão computacional, ou simplesmente reflexão, pode ser definida como qualquer atividade realizada por um sistema sobre si mesmo, sendo muito similar à reflexão humana. Seu principal objetivo é a obtenção de informações sobre suas próprias atividades, com o objetivo de melhorar o seu desempenho, introduzindo novas habilidades, ou mesmo resolvendo seus problemas escolhendo o melhor procedimento. Além disso, pode-se dizer que o uso de reflexão torna um sistema de software mais flexível e mais susceptível a mudanças [1][7][8].

Borde et al. [15] utilizou a reflexão como uma técnica para a adaptação de componentes de software em estrutura e comportamento. Basicamente, as funcionalidades originais são preservadas e outras (novos requisitos) são adicionadas, constituindo um novo componente de software. Além disso, essa técnica também visa controlar o tamanho dos componentes quando uma adaptação é executada, pois eles podem aumentar de tamanho rapidamente e inviabilizar futuras adaptações.

Chen [17] apresentou uma extensão da RMI (*Remote Method Invocation*), chamada XRMI (RMI *eXtended*), que permite a um aplicativo a capacidade de monitoramento e manipulação de chamadas remotas entre os objetos remotos durante uma atividade de adaptação. Essa proposta utiliza como base o padrão *Proxy*, que possibilita que os objetos remotos se comportem como objetos locais.

Shi et al. [3] e Peng et al. [4] relatam que a reflexão foi utilizada com sucesso na reutilização de componentes de software e aplicada em larga escala no reúso da arquitetura de software e de seus componentes arquiteturais. Os autores apresentaram uma proposta que divide a arquitetura de software em dois níveis: (i) *meta*, que representa os componentes de arquitetura, informações que descrevem o nível-base, tais como topologia da arquitetura, componentes arquiteturais e conectores; e (ii) *base*, que pode ser considerado como arquitetura de software tradicional. Assim, observa-se que existem iniciativas importantes, em diferentes contextos, sobre a utilização de reflexão no desenvolvimento de sistemas de software.

Atualmente, na literatura não existe nenhum modelo de referência, baseado em reflexão, que apoie o desenvolvimento e que permita a adaptação do sistema de software em tempo de execução. No entanto, alguns trabalhos relacionados podem ser encontrados. Por exemplo, Weyns et al. [13] propuseram um modelo de referência para autoadaptação chamado FORMS. Esse modelo é composto por um conjunto de primitivas de modelagem formalmente especificadas que corresponde aos pontos-chave de variação dentro sistemas de software autoadaptativos. Os autores relatam que esse modelo tem sido utilizado com sucesso em sistemas para veículos não tripulados e monitoramento por câmeras (inteligentes) de congestiomento de trânsito.

Kramer & Magee [19] apresentaram uma abordagem de arquitetura para sistemas autogerenciáveis, que são capazes de autoconfiguração, autoadaptação, autocura, automonitorização, entre outros. Os autores também propuseram um modelo de arquitetura em dois níveis (meta e base) com características reflexivas. No entanto, as modificações do nível-base (aplicação) são gerenciadas por um plano de ação, que identifica a viabilidade de adaptação e determina quais são as modificações que podem ser implementadas.

Liu et al. [20] apresentaram uma proposta de arquitetura de referência para SOC (*Service-oriented Computing*), com base no mecanismo da auto-organização. A principal contribuição desse trabalho foi infundir o paradigma de projeto de computação orgânica em abordagens de SOC para estabelecer a auto-organização controlada. Dessa forma, mantendo a complexidade do sistema oculta aos participantes (seres humanos) do sistema.

Finalmente, Müller et al. [21] relataram sobre a necessidade de elementos de computação autônoma [22] para a concepção de sistemas de software autoadaptativos em tempo de execução. Segundo os autores, os sistemas de software podem ser chamados de autônomos se operarem principalmente sem intervenção externa (outro sistema ou humana), conforme regras ou políticas estabelecidas. Apesar dessas importantes iniciativas, não existem modelos de referência que apoiem o desenvolvimento e que permita a adaptação do sistema de software em tempo de execução.

III. Adaptação de Software em tempo de execução

Esta seção apresenta um modelo de referência como processo automatizado para adaptação de software em tempo de execução. Como forma de organizar os assuntos, a Seção A apresenta o modelo de referência e a Seção B apresenta o processo de adaptação.

A. Modelo de Referência

Esta seção apresenta o modelo de referência, baseado em reflexão, que tem como objetivo principal apoiar o desenvolvimento de sistemas de software autoadaptativo em tempo de execução. Esse tipo de sistema poderá ser referenciado deste ponto em diante como entidade de software, ou simplesmente entidade. O modelo proposto é resultado de vários anos de experiência no desenvolvimento sistemas de software adaptativos, assim como o uso de tecnologias para projetar tais sistemas. A solução adotada para esse modelo é direcionada para sistemas desenvolvidos em linguagens de programação que possui seguintes características: reflexão, compilação dinâmica e carregamento dinâmico de software. A reflexão permite que as informações sobre a estrutura, comportamento e estado de execução do software sejam recuperados e reutilizados quando o software é modificado. A compilação dinâmica e o carregamento dinâmico estão relacionados à maneira como o software é obtido, compilado e reinserido no ambiente de execução [1][5][7]. A Fig. 1 mostra a estrutura desse modelo de referência.

O modelo apresentado na Fig. 1 está organizado em um núcleo de adaptação (linha pontilhada), que é composto por três módulos (adaptação, carregamento dinâmico e

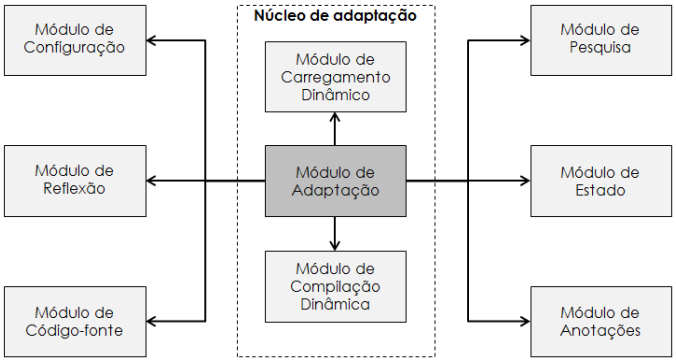

Figura 1. Modelo de Referência.

compilação dinâmica) e outros seis módulos adicionais (anotações, código-fonte, configuração, estado, pesquisa e reflexão). A seguir é apresentada uma breve descrição de cada módulo para que a adaptação de software seja executada automaticamente.

O **módulo de anotações** tem por objetivo auxiliar o engenheiro de software na definição do nível de adaptação de uma entidade de software que está sendo desenvolvida. Para isso, um metadado (anotação), indicando as permissões de adaptação suportadas deve ser inserido para que o módulo de reflexão possa identificar quais informações podem ser modificadas (removidas e/ou inseridas) em tempo de execução. Essa informação (nível de adaptação) permite identificar se determinada entidade de um sistema de software representa uma classe de lógica de sistema ou uma classe de persistência. Além disso, recomenda-se que esse módulo tenha uma funcionalidade para verificar se as anotações foram inseridas corretamente, pois, caso contrário, inviabilizaria o processo de adaptação.

O **módulo de estado** tem por objetivo preservar o estado de execução de uma entidade de software quando a atividade de adaptação está sendo executada. Inicialmente, as informações dessa entidade (estado de execução vigente) são recuperadas e armazenadas em um local externo. Em seguida, essa entidade é adaptada e as informações recuperadas são nela inseridas como forma de preservar seu estado de execução. Para isso, este módulo deve implementar duas funcionalidades (*parsers*): (i) para converter uma entidade de software em um arquivo XML (*eXtensible Markup Language*), que contém sua estrutura e conteúdo das informações; e (ii) para converter o arquivo XML em uma entidade de software. A escolha por esse tipo de arquivo está relacionada às suas facilidades de manipulação (leitura e escrita); de integração com diferentes linguagens de programação; e de implementação.

O **módulo de pesquisa** tem como objetivo localizar entidades de software no ambiente de execução. Quando um sistema de software (entidade) é desenvolvido e inserido no ambiente de execução, um mecanismo automático, chamado fábrica de regras, é responsável por "extrair" dessa entidade, via reflexão, um conjunto de informações que descrevem (regras) sua estrutura e comportamento desse sistema. Essas regras são armazenadas em uma base de regras (repositório) no ambiente de execução. Finalmente, vale ressaltar que o modelo de regras deve ser compatível com o metamodelo que será apresentado no módulo de reflexão.

O **módulo de configuração** tem por objetivo controlar o tamanho das entidades de software e suas metas quando a atividade de adaptação é realizada. Inicialmente, uma entidade de software é desenvolvida para atender propósitos específicos (metas) e atuar em um domínio restrito. Dessa forma, adaptações desordenadas podem fazer com que essa entidade aumente de tamanho rapidamente, inviabilizando futuras adaptações. Sendo assim, a presença de um gestor de configuração permite controlar o tamanho das entidades de software, assim como o número de adaptações, de modo que as metas originais e sua capacidade de adaptação sejam mantidas.

O **módulo de reflexão** tem por objetivo "desmontar" e "montar" uma entidade de software em tempo de execução. Inicialmente, esse módulo faz uma chamada ao módulo de anotações para obter o nível de adaptação suportado por essa entidade. De posse dessas informações, o processo de "desmontagem" é iniciado e as informações estruturais e comportamentais são obtidas (via reflexão) e inseridas em um metamodelo. Após instanciar esse metamodelo, novas informações (conforme os interesses de adaptação) podem ser adicionadas ou removidas, gerando um novo metamodelo. Finalmente, esse novo metamodelo é transferido ao módulo de código-fonte para que a nova entidade de software seja gerada. A Fig. 2 mostra este metamodelo.

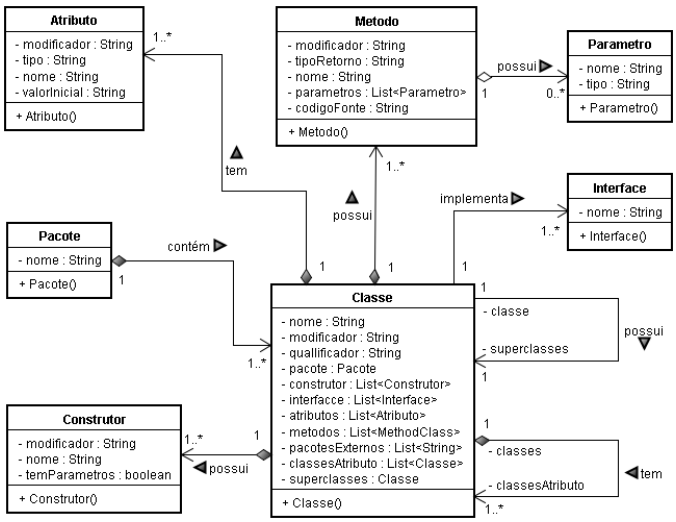

Figura 2. Metamodelo do módulo de reflexão.

Como pode-se observar na Fig. 2, esse metamodelo permite a adaptação de sistemas de software (entidades) orientados a objetos ou que fazem uso da estrutura de classes como unidades de sistemas de software, pois nota-se a presença de classes (`Classe`), atributos (`Atributo`), métodos (`Metodo`), entre outros. Essa estratégia pode ser considerada relevante para os desenvolvedores, pois permite que trabalhem com sistemas de software adaptativos em uma notação próxima do seu domínio de conhecimento.

O **módulo de código-fonte** tem por objetivo gerar as entidades de software com base no metamodelo instanciado no módulo de reflexão. Para executar essa operação, o

engenheiro de software deve fornecer um "*template*" baseado em um padrão arquitetural (camada lógica, camada de persistência, camada de apresentação, e outras). Basicamente, esse módulo deve implementar três funcionalidades que atendam os seguintes interesses de adaptação: (i) *estrutural*, quando apenas um atributo (ou uma lista de atributos) deve ser adicionado ou removido da entidade de software. Neste caso, especificamente, os métodos *getters* e *setters*, que manipulam esses atributos também são modificados; (ii) *comportamental*, quando apenas um método (ou uma lista de métodos) deve ser adicionado ou removido da entidade de software; e, (iii) *estrutural e comportamental*, quando atributos ou métodos (unidade ou lista) podem ser adicionados ou removidos da entidade de software.

Finalmente, o **núcleo de adaptação**, linha pontilhada, é formado por três módulos: (i) carregamento dinâmico, (ii) compilação dinâmica e (iii) adaptação. Em resumo, o módulo de carregamento dinâmico é responsável por "carregar", "descarregar" e "substituir" uma entidade de software no ambiente de execução. O módulo de compilação dinâmica tem por objetivo compilar e recompilar essas entidades em tempo de execução. Uma característica desejável para esse módulo é a capacidade de diagnóstico de erros em código-fonte, pois as mensagens de erro são informações úteis para a interpretação e correções no código-fonte. Por fim, o módulo de adaptação pode ser considerado como o "orquestrador" desse modelo de referência, pois realiza chamadas aos demais módulos de maneira coordenada, em um processo de "linha de montagem", fazendo com que a adaptação seja executada automaticamente. Assim, esse modelo surge como uma solução factível para a adaptação de sistemas de software (entidades) em tempo de execução de maneira automática e sem a participação de seres humanos (desenvolvedores).

B. Processo de adaptação

Essa seção apresenta o processo de adaptação das entidades software existente no modelo de referência apresentado na Seção III-A (especificamente no módulo de adaptação). Esse processo está organizado em uma sequência de oito passos, organizados em uma sequência lógica como se fosse uma "linha de montagem", como ilustra a Fig. 3. A seguir é apresentada uma breve descrição para que uma entidade de software seja adaptada. Nesta figura, os módulos da Fig. 1 estão representados por pacotes em notação UML.

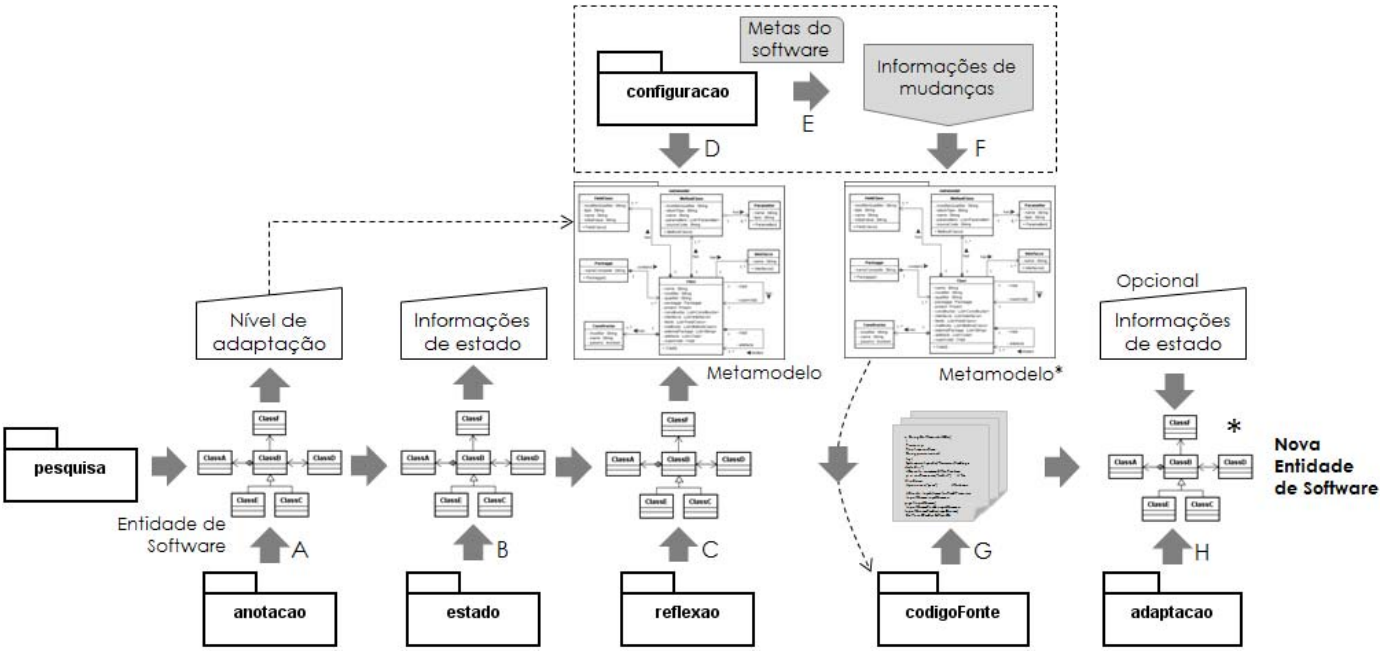

Figura 3. Processo de adaptação.

Inicialmente, uma entidade de software é selecionada, pelo módulo de pesquisa, para ser adaptada. De posse dessa entidade, o processo de adaptação é iniciado com a recuperação do nível de adaptação dessa entidade (passo A), que será utilizado na confecção do metamodelo (passo C). No passo B, o módulo de estado, que é o responsável por preservar o estado de execução vigente da entidade de software naquele momento, salva as informações de estado em um arquivo externo (XML). Vale ressaltar que essas informações podem ser reutilizadas no passo H, caso a entidade de software não tenha sofrido alterações de domínio (metas). No passo C, a entidade de software é desmontada pelo módulo de reflexão e um metamodelo (conforme apresentado na Fig. 2) é instanciado contendo as informações estruturais, comportamentais e nível de adaptação dessa entidade (todos obtidos no passo A). O passo D consiste em gerenciar a adaptação da entidade de software. No entanto, essa é uma atividade macro e complexa (linha pontilhada - Fig. 3), formada por três passos (D, E, F). O módulo de configuração visa estabelecer os critérios de adaptação para que as metas da entidade não se tornem inviáveis (impedindo futuras adaptações). Para isso, são utilizadas as bases de regras, que oferecem diretrizes quanto à

viabilidade de adaptação (passo E). Em seguida, deve ser estabelecido o procedimento (plano) de adaptação, com base nas metas e nas novas informações de mudança (requisitos de adaptação). Baseado nessas informações, um novo metamodelo (passo F) é gerado e transferido ao módulo de código-fonte para que as novas entidades sejam geradas (passo G). Finalmente, o código-fonte é compilado e as entidades são inseridas no ambiente de execução (passo H). Sobre esse último passo, vale ressaltar que, caso as modificações nas entidades não reflitam em impactos na mudança de domínio (metas), as informações preservadas (passo B) são reinseridas nessas novas entidades para que substituam as antigas no ambiente de execução. A "substituição" das entidades é realizada de maneira transparente aos seus interessados, pois eles não têm a percepção das mudanças ocorridas em função da preservação do estado de execução e da manutenção das instâncias das entidades.

IV. Estudo de Caso

Antes de iniciar a apresentação desse estudo de caso, algumas considerações devem ser enfatizadas. A **primeira** está relacionada à implementação do modelo de referência apresentado na Seção III, pois este foi instanciado na linguagem de programação Java [23] e, por motivos de insuficiência de espaços, os detalhes de implementação não são apresentados neste artigo. A **segunda** refere-se ao tamanho e organização da lógica do sistema escolhido, que pode ser caracterizado como um sistema de informação para gerenciamento de uma livraria. Este sistema foi selecionado por ser considerado suficiente para mostrar a utilização do modelo de referência apresentado neste artigo (Seção III) –adaptação estrutural. A **terceira** está relacionada ao uso dos módulos desse modelo de referência, pois são enfatizados apenas neste estudo de caso os passos relacionados à adaptação das entidades de software (módulo de adaptação) e geração de código-fonte das entidades de software (módulo de código-fonte). O uso dos demais módulos foi omitido por motivos de escopo/espaço. Finalmente, a **quarta** refere-se ao tipo de adaptação abordada, sendo apresentada neste artigo apenas a adaptação estrutural. No entanto, vale ressaltar que a adaptação comportamental também é suportada por este modelo de referência (Seção III). Outro fator a ser ressaltado é que este tipo de adaptação escolhido (estrutural) foi considerado pelos autores como suficiente para mostrar o funcionamento e a viabilidade do modelo de referência (Seção III) na adaptação de entidades de software.

Conforme mencionado anteriormente, o modelo de referência apresentado neste artigo (Seção III) atua na adaptação de sistemas de software como uma "linha de montagem", ou seja, com uma sequência de passos bem definida. Assim, a Fig. 4 mostra a adaptação da entidade de software `Cliente`, inicialmente desenvolvida para atuar em um sistema local, sendo adaptada para atuar em um sistema web com autenticação. Partindo da entidade original (**modelo UML da entidade cliente**), um metamodelo é instanciado. Esse metamodelo contém: (i) níveis de adaptação suportados pela entidade, que foram recuperados pelo módulo de anotações; e (ii) informações estruturais e comportamentais dessa entidade, que foram recuperadas pelo módulo de reflexão.

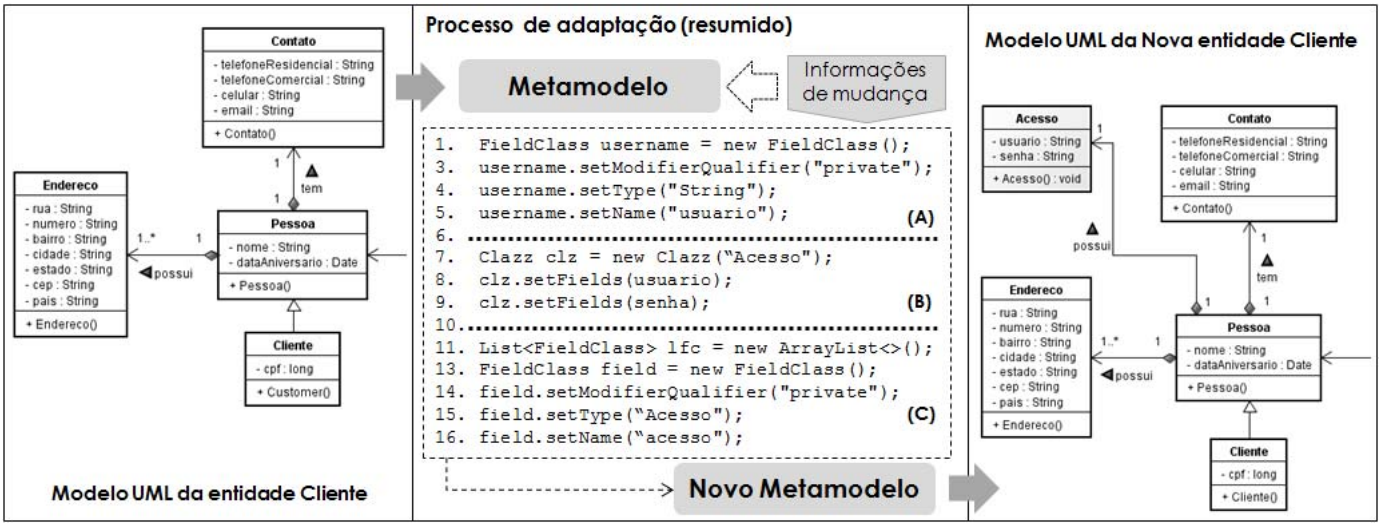

Figura 4. Adaptação da entidade Cliente.

De posse dessas informações (metamodelo) podem ser inseridas as informações de mudança (adaptação) - para que essa entidade possa atuar em um sistema web com autenticação. Para isso, uma entidade (`Acesso`) deve ser criada e associada (agregação por composição) à entidade `Pessoa`. Sobre a associação entre essas entidades (`Acesso` e `Pessoa`), vale ressaltar que as informações obtidas pelo módulo de anotações contêm metadados que determinam quais entidades, das entidades que estão sendo adaptadas, podem ser modificadas (receber atributos de relacionamentos).

A criação da entidade `Acesso` está representada entre as linhas 1 e 9 (**Processo de adaptação resumido**) - Seção (A) da Fig. 4. Inicialmente, é possível observar a criação do atributo `usuario` (linhas 1 a 5), em que são definidos os se-

guintes itens: modificador de acesso, tipo e o nome. O trecho referente à criação do atributo `senha` foi omitido, no entanto, sua criação foi realizada de maneira similar. Entre as linhas 7 e 9 (Seção B) é possível observar a criação da entidade `Acesso` com os atributos criados anteriormente. Finalmente, entre as linhas 11 e 16 - Seção (C), observa-se a criação do atributo `acesso` que será adicionado ao metamodelo da entidade `Cliente`, gerando uma nova entidade de software (novo metamodelo). Esse novo metamodelo (**modelo UML da nova entidade Cliente**) é transferido ao módulo de código-fonte para que a nova entidade de software seja gerada.

Para finalizar essa seção, vale ressaltar que o modelo de referência apresentado neste artigo tem-se mostrado como uma solução factível para a adaptação de software, pois a atuação dos seus módulos em linha de montagem permite que uma entidade de software seja desenvolvida e adaptada automaticamente sem a participação dos seres humanos (desenvolvedores).

V. Discussão dos Resultados e Limitações

Processos automatizados. Os processos automatizados em engenharia de software são essenciais para o desenvolvimento de sistemas de software, pois otimizam a produtividade (tempo e custo) e minimizam, ou praticamente eliminam, o envolvimento dos desenvolvedores. Conforme mencionado pelos autores [7] [8] [9], a adaptação de software, quando realizada manualmente, é uma atividade onerosa e propensa a erros. Considerando este cenário (automação x adaptação), o modelo de referência apresentado neste artigo emerge como uma solução factível e apresenta boa perspectiva para contribuir de forma eficiente com a área de sistemas autoadaptativos. As entidades de software são adaptadas por um conjunto de módulos através de uma sequência de passos bem definida (linha de montagem). Dessa forma, permite que os engenheiros de software (desenvolvedores) possam atuar em um nível de abstração mais elevado, evitando que conhecimentos específicos sejam necessários para que uma entidade de software seja adaptada.

Modelo de referência para sistemas adaptativos. Na literatura, é possível encontrar diferentes trabalhos que abordam a adaptação de software em tempo de execução em diversos níveis de abstração. Por exemplo, em um baixo nível de abstração, pode-se citar o framework ASM [24], que exige conhecimentos específicos sobre código de máquina para adaptação de software. No modelo de referência apresentado neste artigo, as entidades de software são adaptadas em uma sequência de passos como uma "linha de montagem". Basicamente, uma entidade é "desmontada" e suas informações (estruturais, comportamentais e adaptação) são inseridas em um metamodelo, que é modificado (recebendo novas informações de interesse de adaptação) para que a nova entidade seja gerada. Assim, pode-se dizer que os engenheiros de software atuam em um nível mais elevado de abstração, realizando tarefas mais próximas daqueles que estão acostumados a realizar.

Neste artigo, o modelo de referência foi instanciado na linguagem de programação Java [23], pois esta atende aos requisitos exigidos por este modelo (Seção III). No entanto, os resultados obtidos até o momento permite acreditar que este modelo pode ser instanciado em outras linguagens de programação.

Limitações. O modelo de referência apresentado neste artigo não cobre todos os casos e não resolve todos os problemas relacionados à adaptação de software em tempo de execução. No entanto, este modelo representa uma direção, a fim de estabelecer um modelo de referência para o domínio sistemas de software autoadaptativos, além de aliviar a pressão existente sobre o assunto. Além disso, conforme apresentado neste artigo, este modelo foi instanciado na linguagem de programação Java. Este modelo ainda não foi instanciado em outras linguagens de programação e, portanto, não se pode avaliar sua aplicabilidade e comportamento na adaptação de software em outras linguagens de programação. Finalmente, este modelo de referência não foi utilizado na indústria e, portanto, validado por completo. Assim, o planejamento futuro é desenvolver estudos de casos nesta direção.

VI. Conclusões

Este artigo apresentou um modelo de referência que pretende apoiar o desenvolvimento e adaptação de sistemas de software em tempo de execução. Através desse modelo, as entidades de software são monitoradas e adaptadas em tempo de execução de maneira transparente, sem a percepção de seus interessados. Para realizar as operações de adaptação, o modelo proposto utiliza um conjunto de módulos, que atuam em um processo como se fosse uma "linha de montagem", onde uma entidade de software é desmontada, adaptada e remontada automaticamente por esses módulos. Dessa forma, as principais contribuições deste artigo são:

- Para a área de sistemas autoadaptativo, como uma alternativa que facilita o desenvolvimento de sistemas de software adaptativos em tempo de execução;
- Para a área de arquitetura de referência e modelos de referência, propondo um modelo baseado em reflexão que considera adaptações de entidades de software em tempo de execução;
- Para a área de automação de software, pois neste modelo, as entidades de software são adaptadas como se fosse uma “linha de montagem”, ou seja, em uma sequência de passos bem definida. Além disso, é importante destacar que os módulos realizam as operações de adaptação das entidades de software sem intervenção de seus desenvolvedores; e
- Por fim, acredita-se que este modelo pode ser utilizado de forma adequada, juntamente com outros processos de desenvolvimento de software que estão sendo utilizados na indústria, uma vez que ambos parecem ser complementares.

Como trabalho futuro, três objetivos são apresentados: (i) **o primeiro** está relacionado com estudos de caso que serão realizados para avaliar o modelo de referência apresentado neste artigo, pois outros tipos de adaptação de entidades de software devem ser investigados; (ii) **o segundo** está relacionado com a instância deste modelo, pois pretende-se instanciá-lo em outras linguagens de programação para avaliar a sua aplicabilidade e comportamento na adaptação de entidades de software; e (iii) **o terceiro** está relacionado com a utilização deste modelo na indústria, dado que se pretende avaliar seu comportamento quando aplicado em grandes ambientes de desenvolvimento e execução. Por fim, acredita-se ter um cenário positivo de pesquisa, com a perspectiva de torná-lo uma contribuição efetiva para a comunidade de desenvolvimento de software.

AGRADECIMENTOS

Os autores receberam suporte financeiro da PROPe/UNESP (Pró-reitoria de Pesquisa da Universidade Estadual Paulista) para realização deste trabalho.

REFERÊNCIAS

[1] P. Maes, "Concepts and experiments in computational reflection". ACM SIGPLAN Notices, Orlando: ISSN:0362-1340, 1987;

[2] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@ Run.time to Support Dynamic Adaptation," Computer, vol.42, no.10, pp.44-51, Oct. 2009;

[3] Y. Shi, L. ZaoQing, W. JunLi, and W. FuDi, "A reflection mechanism for reusing software architecture", QSIC 2006, pp. 235-243, oct. 2006;

[4] Y. Peng, Y. Shi, J. Xiang-Yang, Y. Jun-Feng, L. Ju-Bo, and Y. Wen-Jie, "A reflective information model for reusing software architecture", ISECS/CCCM 2008, vol. 1, 2008, pp. 270-275;

[5] G. Coulson, G. Blair, and P. Grace, "On the performance of reflective systems software", CPCC 2004, 2004, pp. 763 – 769;

[6] X. Hongzhen and Z. Guosun, "Retracted: Specification and verification of dynamic evolution of software architectures", Journal of Systems Architecture, vol. 56, no. 10, pp. 523-533, 2010;

[7] S. Vinoski, "A time for reflection [software reflection]", Internet Computing, IEEE, vol. 9, no. 1, pp. 86-89, 2005;

[8] J. Andersson, R. de Lemos, S. Malek, and D. Weyns, "Reflecting on self-adaptive software systems", ISCE/SEAMS 2009, may 2009, pp. 38-47;

[9] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges", ACM Trans. Auton. Adapt. Syst., vol. 4, no. 2, pp. 1-42, May 2009;

[10] F. J. Affonso and E. L. L. Rodrigues, "A proposal of reference architecture for the reconfigurable software development", SEKE 2012, jul 2012, pp. 668–671;

[11] J. Whitehead, "Collaboration in software engineering: A roadmap", FOSE 2007, 2007, pp. 214-225;

[12] L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice", ISBN-10: 0321154959, 2a edição, 2003;

[13] D. Weyns, S. Malek, J. Andersson, "Forms: a formal reference model for self-adaptation". ICAC 2010, pages 205-214, 2010;

[14] S. G. R. High, S. Kinder, "Ibm's soa foundation - an architectural introduction and overview", IBM, Tech. Rep., 2005;

[15] E. Y. Nakagawa, F. C. Ferrari, M. M. Sasaki, and J. C. Maldonado, "An aspect-oriented reference architecture for software engineering environments," Journal of Systems and Software, vol. 84, no. 10, pp. 1670 – 1684, 2011;

[16] E. Borde, G. Haïk, and L. Pautet, "Mode-based reconfiguration of critical software component architectures", DATE 2009, 2009, pp. 1160-1165;

[17] X. Chen, "Extending rmi to support dynamic reconfiguration of distributed systems", CDCS 2002, 2002, pp. 401-408;

[18] D. Weyns, S. Malek, and J. Andersson, "Forms: a formal reference model for self-adaptation," ICAC 2010, 2010, pp. 205-214;

[19] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge", FOSE 2007, may 2007, pp. 259-268;

[20] L. Liu, S. Thanheiser, and H. Schmeck, "A reference architecture for self-organizing service-oriented computing", ARCS 2008, pp. 205-219, 2008;

[21] H. Müller, H. Kienle, and U. Stege, "Autonomic computing now you see it, now you don't: Design and evolution of autonomic software systems", Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 5413, pp. 32–54, 2009;

[22] IBM, "An architectural blueprint for autonomic computing", on-line, 2005. Disponível em: http://www.ginkgo-networks.com/IMG/pdf/AC_Blueprint_White_Paper_V7.pdf, Acessado em: 19 de jun de 2013;

[23] JAVA, "Java Platform, Standard Edition (Java SE)", on-line, 2013. Disponível em: http://www.oracle.com/us/technologies/java/standard-edition/overview/index.html, Acessado em: 24 de jun. de 2013;

[24] ASM, "ASM framework", on-line, 2013, Disponível em: http://asm.ow2.org/. Acesso em 26 de jun. de 2013;

Frank José Affonso possui graduação em Ciência da Computação pelo Centro Universitário Central Paulista (2000), mestrado em Ciência da Computação pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos (05/09/2003) e doutorado em Engenharia Elétrica pelo Programa de Pós-Graduação do Departamento de Engenharia Elétrica da Escola de Engenharia de São Carlos - Universidade de São Paulo (26/05/2009). Atualmente é professor assistente doutor em Regime (RDIDP) - Efetivo da PP-QDUNESP. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software e Programação JAVA, atuando principalmente nos seguintes temas: Engenharia de Software baseada em Modelos; Reengenharia de Interface; JAVA com desenvolvimento Web, Desktop e Distribuído; C/C++; UML; Framework de Persistência (Hibernate/JPA) e Reconfiguração de Software em tempo de Execução.

Maria Cecília Vecchiato Saenz Carneiro possui graduação em Bacharelado em Matemática pela Universidade Estadual Paulista Júlio de Mesquita Filho (1979), mestrado em Ciências da Computação e Matemática Computacional pela Universidade de São Paulo (1987) e doutorado em Ciências da Engenharia Ambiental pela Universidade de São Paulo (1996). Atualmente é professor assistente doutor em Regime (RDIDP) Efetivo da Universidade Estadual Paulista Júlio de Mesquita Filho. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: metodologias de desenvolvimento de sistemas, linguagem de modelagem UML, arquitetura e projeto arquitetural de software, sistemas de apoio a decisão voltados a políticas públicas.

Evandro Luís Linhari Rodrigues possui graduação em Engenharia Elétrica pela Escola de Engenharia de Lins (1983), mestrado em Engenharia Elétrica pela Universidade de São Paulo (1992) e doutorado em Física pela Universidade de São Paulo (1998). Atualmente é professor da Universidade de São Paulo. Tem experiência na área de Engenharia Elétrica, com ênfase em Automação Eletrônica de Processos Elétricos e Industriais, atuando principalmente nos seguintes temas: processamento de imagens, microprocessadores/microcontroladores, visão computacional, análise carpal e automação.

Elisa Yumi Nakagawa é graduada no curso de Bacharelado em Ciências de Computação pela Universidade de São Paulo, São Carlos, SP, em 1994, tem mestrado em Ciências de Computação e Matemática Computacional pela Universidade de São Paulo, São Carlos, SP, em 1998 e doutorado em Ciências de Computação e Matemática Computacional pela Universidade de São Paulo, São Carlos, SP, em 2006. Ela é docente do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo desde abril de 2001. As linhas de pesquisa abordadas são: arquitetura de software, ambiente de engenharia de software, teste de software, aplicações web e ontologia, todas no contexto da área de Engenharia de Software. Atua no Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional do ICMC/USP (Nível 6 da Capes) como orientadora de alunos de mestrado e doutorado e como responsável ministrando disciplinas de pós-graduação. Também é co-orientadora de alunos

de mestrado e doutorado do PhD-Program of Computer Science e do Regular Master in Computer Science da Universidade de Kaiserslautern, Alemanha, e do Groningen Graduate School of Science da Universidade de Groningen, Holanda. Em termos de formação acadêmica, conduziu seu projeto de pós-doutorado em 2011-2012 na Fraunhofer Institute for Experimental Software Engineering (Fraunhofer IESE), Alemanha.