SASeS: A Framework for the Development of Service-based Self-adaptive Applications

E. C. Júnior P. H. M. Maia and F. J. Affonso

***Abstract*— Self-adaptive Systems (SaS) enable the structural or behavioral adaptation at runtime in response to context changes or user's new needs without interruption in their execution. In service-based SaS, the adaptation activity can be considered as a complex activity, since such systems need to modify or replace a service (e.g., due to an unforeseen situation or failure) at runtime without perception of their stakeholders (i.e., client of these services). Based on this scenario, a framework called SASeS (Self-Adaptive Service-based System) was design. In short, this framework was developed based on web services standards and design patterns that support common tasks in the development of such systems. Moreover, SASeS provides a mechanism for the management of the control loop (or feedback loop), which enables constantly self-evaluating (i.e., monitor, analyze, plan, and execute) the adaptation activity. In order to show the applicability of our framework, two case studies are reported in this paper.**

***Keywords*— framework, service-based systems, self-adaptive systems.**

I. Introdução

A SOCIEDADE atual está cada vez mais dependente de sistemas de software para a execução de tarefas diárias. Instituições públicas, aeroportos, sistemas de comunicação, comércio de produtos, sistemas financeiros são alguns dos exemplos que possuem informatização em seus processos. Em geral, esses sistemas manipulam informações e geram serviços que devem estar disponíveis 24/7 (ou seja, 24 horas por dia e sete dias por semana). A interrupção, mesmo que momentânea, em tais sistemas pode gerar transtornos leves a severos [5][34][35]. Dessa forma, características como robustez, confiabilidade, escalabilidade, customização, auto-organização, autoadaptação são cada vez mais requeridas por esses sistemas. Especificamente, essas três últimas características podem indicar a necessidade de modificação dos sistemas de software em tempo de execução, seja pelas novas necessidades de seus usuários ou pela adequação ao contexto ao qual está inserido (ou seja, ambiente de execução). Esses sistemas são considerados específicos, pois permitem que novos recursos, estruturais ou comportamentais, sejam incorporados em tempo de execução sem que suas atividades sejam interrompidas [5][22][25][33][34].

Ainda sobre o cenário atual de sistemas de software, observa-se um alto grau de complexidade nesses sistemas, pois estão sendo projetados, simultaneamente ou não, para atuar em ambientes distribuídos, móveis e/ou embarcados. Assim, pode-se dizer que a manutenção e/ou evolução de tais sistemas está se tornando complexa em proporção igualitária ou até superior. Portanto, projetar sistemas capazes de se adaptar ao longo do seu ciclo de vida tem sido uma alternativa factível. Em geral, sistemas autoadaptativos são capazes de modificar sua estrutura e/ou comportamento em resposta a sua percepção de mudanças (ou seja, contexto ao qual está inserido ou por novas necessidades de seus usuários) sem a intervenção humana [5][17][19][22].

De acordo com [9][17][22][32], a adaptação de software, quando realizada manualmente torna-se uma atividade onerosa (tempo e custo) e propensa a erros (injeção involuntária de incertezas pelos desenvolvedores). Para contornar essas adversidades, processos automatizados (ou seja, ferramentas, *frameworks*, geradores de código, entre outros) são fortemente recomendados, pois representam uma alternativa factível para maximizar a velocidade de implementação do software e minimizar o envolvimento de seres humanos (desenvolvedores) na execução das atividades de adaptação.

Em [4] pode-se encontrar um conjunto de aplicações com características de autoadaptação. Dentre elas, as que mais destacam são aquelas voltadas para: (i) computação autônoma, (ii) sistemas embarcados, (iii) robótica, (iv) sistemas multi-agentes, (v) aplicações orientadas a serviços, e (vi) computação ubíqua. As aplicações orientadas a serviço têm recebido atenção especial tanto na academia quanto na indústria, pois são utilizadas em diferentes tipos de sistemas. Essas aplicações são construídas a partir de serviços de baixo acoplamento e podem envolver vários provedores. Especificamente, no contexto de sistemas autoadaptativos baseados em serviços, uma aplicação pode, em tempo de execução, modificar um serviço para execução de uma tarefa (ou seja, um serviço pode ser descoberto e invocado dinamicamente). Baseado nessa característica, os projetistas podem desenvolver aplicações mais rapidamente, utilizando serviços de terceiros para compor a aplicação desejada [29][30][31].

Na literatura é possível encontrar trabalhos que abordam o desenvolvimento de sistemas autoadaptativos baseados em serviços, que reportam modelos arquiteturais [6], frameworks [28] e ferramentas [23]. Para [29][30][31], uma maneira de desenvolver tais sistemas é prover a capacidade de seleção de serviços similares em tempo de execução, a partir de conjuntos de serviços funcionalmente equivalentes associados a diferentes níveis de desempenho, confiabilidade e custo. Nesse sentido, [22] relata que existem diversas lacunas de pesquisa que englobam o desenvolvimento de novos *frameworks*, ferramentas e linguagens adequadas à criação de sistemas autoadaptativos baseados em serviços. Isso se deve a

E. C. Júnior, Universidade Estadual do Ceará (UECE), Centro de Ciências e Tecnologia, Campus do Itaperi, Fortaleza, Ceará, Brasil, junior.facanha@gmail.com

P. H. M. Maia, Universidade Estadual do Ceará (UECE), Centro de Ciências e Tecnologia, Campus do Itaperi, Fortaleza, Ceará, Brasil, pauloh.maia@uece.br

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

melhorias constantes nas técnicas que abordam o gerenciamento de recursos autônomos e monitoramento dos atributos de qualidade para estes sistemas.

Baseado no cenário apresentado, este artigo apresenta o SASeS (do inglês, *Self-Adaptive Service-based System*), um *framework* cujo objetivo é apoiar o desenvolvimento de sistemas autoadaptativos baseados em serviços. Esse *framework* implementa o *loop* de controle MAPE (do inglês, *Monitor, Analyze, Plan and Execute*) [15] como um meio de gerenciar a evolução dos serviços. Sobre a implementação, o *framework* foi desenvolvido em Java[20] e utiliza a padrão de projeto *Observer* [8]. Dessa forma, os componentes do *loop* podem se comunicar por meio de um sistema de notificação (notificador - notificado). Além disso, o SASeS provê uma interface visual para que o desenvolvedor possa descrever os possíveis serviços que serão utilizados em futuras adaptações. Outra contribuição de SASeS é que é possível utilizar tanto *sockets* quanto arquivos XML para invocação dos serviços alvos da adaptação. Por fim, um dos diferenciais desse trabalho em relação aos trabalhos encontrados na literatura é a disponibilidade de seu código fonte na Internet (endereço curto: http://goo.gl/r3igB5) para que os desenvolvedores possam utilizá-lo em suas aplicações ou até mesmo replicar os estudos de caso apresentados neste artigo.

Este artigo está organizado da seguinte maneira: a Seção II apresenta os conceitos, definições e os trabalhos relacionados; a Seção III mostra o *framework* SASeS; a Seção IV apresenta o estudo de caso desenvolvido como forma de validar o *framework* proposto; a Seção V reporta as discussões e limitações deste artigo; e, finalmente, na Seção VI são apresentadas as conclusões e perspectivas de trabalho futuro.

II. Conceitos, Definições e Trabalhos Relacionados

Esta Seção apresenta conceitos, definições e trabalhos relacionados que contribuíram para o desenvolvimento do *framework* SASeS. Inicialmente, Seção A, conceitos e definições sobre assuntos relacionados ao tema de pesquisa são apresentados. Em seguida, Seção B, trabalhos relacionados ao *framework* proposto neste artigo são reportados, assim como suas contribuições, restrições e limitações.

A. Conceitos e Definições

Segundo [26][27], sistemas de software autoadaptativos modificam seu próprio comportamento em resposta às mudanças de contexto ou por novas necessidades de seus interessados. Pelo contexto entende-se o ambiente no qual o sistema está inserido, ou seja, quaisquer itens observáveis do sistema, tais como: entradas de usuário, dispositivos externos de hardware, sensores, entre outros. Já os interessados são aqueles que fazem uso do sistema e que necessitam de modificação para que suas novas necessidades sejam atendidas. Um ponto considerado como chave em tais sistemas é o ciclo de vida interminável, pois necessitam constantemente de autoavaliação para que possam implantar as solicitações de mudanças requeridas, tanto pelo ambiente quanto pelos usuários. Ainda nesse contexto, para que um software possa evoluir ou se autoadaptar, três razões são consideradas fundamentais [29][30]: (i) a implementação do software não está satisfazendo as especificações; (ii) o conhecimento do ambiente diverge das suposições de domínio; ou (iii) os requisitos não estão de acordo com as necessidades do usuário.

De acordo com [6][13][17][22], o *loop* de controle MAPE-K (do inglês, *Monitor, Analyze, Plan, Execute over Knowledge base*) [13][15] tem sido utilizado como uma alternativa factível para viabilizar a autoadaptação em sistemas de software. Nesse *loop*, o componente Monitor é responsável por realizar o monitoramento do ambiente ou da aplicação adaptada através de, por exemplo, sensores ou *logs* de execução. Essas informações são repassadas para o componente de Análise, que verifica se propriedades do sistema estão sendo respeitadas através de técnicas como *model checking* [24]. Em caso negativo, ele sinaliza a necessidade de uma adaptação ao componente Planejador. Este, por sua vez, contém possíveis estratégias de adaptação e decide qual a mais apropriada a ser aplicada e a informa ao componente Executor, que realiza de fato a adaptação na aplicação, por exemplo substituindo um serviço por outro. Esse conhecimento é armazenado em uma base representada pelo componente *Knowledge*.

Por exemplo, um modelo de referência [12] e uma arquitetura de referência [10] para sistemas autoadaptativos foram elaborados com base no *loop* de controle MAPE-K [15]. Como resultado específico desses trabalhos, um *framework* de apoio à tomada de decisão baseado em técnicas de aprendizado foi desenvolvido [11]. Em resumo, tal solução permite a identificação de um problema, a recomendação de uma solução e o teste das soluções propostas antes de serem efetivadas no ambiente de execução. Sensores e efetores também são componentes desse *framework*, pois representam um meio de interação entre o sistema supervisor e as entidades supervisionadas (ou seja, a abordagem proposta em [10][12]).

Por fim, em [22] é apresentada uma taxonomia sobre abordagens de adaptação. Basicamente, essas abordagens se dividem em (i) interna, quando a lógica de adaptação está embutida no sistema de software, e (ii) externa, quando a lógica de adaptação está isolada do sistema de software. Essas abordagens são utilizadas em diferentes tipos de sistemas de software autoadaptativos, inclusive os baseados em serviços. A seguir, Seção B, são apresentados os trabalhos relacionados a este.

B. Trabalhos Relacionados

Sistemas baseados em serviço são aplicações de software construídas a partir de serviços de baixo acoplamento, que independem de outros serviços ou aplicações de vários fornecedores. Tais sistemas podem ser encontrados em diferentes domínios de aplicação, tais como: comércio eletrônico, *internet banking*, *health care*, entre outros. Esses sistemas são caracterizados por frequentes mudanças em suas operações. Além disso, esses sistemas atuam em ambientes abertos, onde eventos externos podem gerar situações imprevistas e dinâmicas, uma vez que novos serviços podem ser descobertos e selecionados em tempo de execução [3][30]. Além disso, [7] afirma que tais alterações podem incluir a implantação de novas instâncias de um determinado tipo de serviço, a remoção de serviços já existentes, ou até mesmo mudanças globais na aplicação. A seguir são apresentados os principais trabalhos encontrados na literatura que apresentam iniciativas para o

contexto deste artigo (ou seja, sistema de software autoadaptativos baseados em serviços).

Um *framework* para apoiar o desenvolvimento de sistemas pervasivos que promova a adaptação da aplicação, para que esta possa atuar em diferentes dispositivos, foi proposto por [23]. A aplicação desenvolvida com base nesse *framework* deve ser capaz de verificar a compatibilidade entre os recursos requeridos e os fornecidos pelos dispositivos para que a portabilidade possa ser realizada. Para isso, uma abordagem declarativa desenvolvida em Java é utilizada. Essa abordagem é baseada em formulação matemática (lógica), que permite descrever a (in)compatibilidade e viabilizar a adaptação.

Em [14] os autores propuseram o *Adaptive Server Framework* (ASF), que pode ser utilizado para a criação de servidores de aplicação com comportamento adaptativo. O ASF provê uma separação clara entre implementação de comportamentos adaptativos e o servidor de aplicação da lógica de negócio. Além disso, o ASF possui uma arquitetura leve em que incorre baixa sobrecarga de CPU e uso de memória.

Cronologicamente, em 2008[16], 2010[1] e 2015[11], podem ser encontrados na literatura trabalhos que envolvem o projeto de *frameworks* como um meio de apoio ao mecanismo de tomada de decisão para sistemas autoadaptativos. O primeiro, chamado *Autonomic Management Toolkit* (AMT) [16], possui um mecanismo de regras para o raciocínio e tomada de decisão (*loops* de adaptação). O segundo, chamado FUSION [1], visa resolver o problema da previsão das mudanças do ambiente. Para isso, técnicas de aprendizagem de máquina foram utilizadas para que o sistema possa coletar informações e apresentar soluções. O terceiro, chamado DmF (*Decision-making Framework)* [11], é baseado no *loop* de controle MAPE-K e em técnicas de aprendizado de máquina. A diferença em relação ao segundo é o item de autoteste, que permite que uma solução proposta seja avaliada antes de ser efetivada no ambiente de execução.

Por fim, [29][30][31] propuseram a utilização de *proxies* inteligentes para o desenvolvimento de sistemas autoadaptativos baseados em serviços. Tais *proxies* são utilizados na composição de serviços com a propriedade de autoverificação.

Assim, é possível notar que existem importantes iniciativas voltadas ao desenvolvimento de sistemas de software autoadaptativos baseados em serviços. No entanto, nenhum dos trabalhos contempla uma abordagem que permite a criação da aplicação adaptável de maneira rápida e objetiva. Para isso, o SASeS possui um gerenciador externo (Módulo Adaptador), que é responsável pelo gerenciamento do ciclo de adaptação. Esse módulo permite que serviços sejam selecionados em função das necessidades da aplicação. Os serviços selecionados pelo módulo não precisam ser pré-definidos, assim é possível remover e adicionar novos serviços da aplicação em tempo de execução. Outro ponto a ser destacado é a possibilidade de utilizar arquivos XML ou *sockets* para configuração e troca de mensagens entre o Módulo Adaptador e a Aplicação, que flexibiliza ainda mais seu uso pelos desenvolvedores.

A Tabela 1 apresenta um breve sumário sobre os trabalhos relacionados a este. Essa tabela apresenta dois aspectos de comparação, conforme exposto em [21]: (1) pesquisa e (2) softwares autoadaptativos e softwares autoadaptativos baseados em serviços. O primeiro representa indicações sobre a maturidade do trabalho e se o mesmo: (i) propôs um *framework*; (ii) apresenta estudo de caso; (iii) apresenta uma autoavaliação de sua proposta; (iv) disponibilizada uma maneira de acesso ao código fonte; e, por fim, (v) apresentada alguma limitação. O segundo está relacionado ao tópico de pesquisa deste artigo e indica se o trabalho: (i) apresenta como é feita a adaptação utilizando o *framework* ou abordagem proposta; (ii) possui um componente responsável pelo ciclo de adaptação; (iii) trata incertezas do ambiente; (iv) trata de sistemas baseados em serviços; (v) mostra o estilo de adaptação utilizado; e, por fim, (vi) apresenta a abordagem de adaptação utilizada.

TABELA I
COMPARATIVO ENTRE O SASES E OS TRABALHOS RELACIONADOS

Trabalhos	Aspectos de Pesquisa					Aspectos de Sistemas de Softwares Auto-Adaptativos e Baseados em					
	Framework	Estudo de Caso	Auto Avaliação	Disponibilidade de Código Fonte	Limitações	Apresenta a Adaptação	Ciclo de Adaptação	Incertezas do Ambiente	Sistemas Baseados em Serviços	Estilo de Adaptação	Abordagem de Adaptação
INVERARDI et al., 2004	√	X	√	X	X	√	X	X	X	Dinâmica	Externa
GORTON et al., 2006	√	√	√	X	X	√	√	√	X	Dinâmica	Externa
ADAMCZYK et al., 2008	√	√	√	X	X	√	√	X	X	Dinâmica	Externa
ASADOLLAHI et al., 2009	√	√	√	X	X	√	√	X	X	Dinâmica	Externa
ELKHODARY et al., 2010	√	√	√	X	X	√	√	√	X	Dinâmica	Externa
CALINESCU E RAFIQ, 2013	X	√	X	X	X	√	√	X	√	Dinâmica	Externa
CALINESCU et al., 2013	√	√	√	X	X	√	√	X	√	Dinâmica	Externa
Framework SASeS	√	√	√	√	√	√	√	X	√	Dinâmica	Externa

Diante do exposto na Tabela 1, os outros trabalhos apresentados nesta seção apresentam limitações, sejam no aspecto da pesquisa ou de softwares autoadaptativos. O SASeS contempla praticamente todos os critérios, exceto ao tratamento de incertezas do ambiente. Isso destaca o potencial desse trabalho em relação aos demais.

III. *FRAMEWORK* SASES

O *framework* SASeS foi projetado para auxiliar o engenheiro de software no desenvolvimento de sistemas de softwares autoadaptativos baseados em serviços. Para isso, o SASeS requer que um módulo adaptador seja acoplado à aplicação que se pretende monitorar em uma abordagem "supervisor-supervisionado". Esse adaptador é responsável por gerenciar, em tempo de execução, uma lista de serviços para cada aplica-

ção. Vale ressaltar ainda que, dependendo da necessidade da aplicação, essa lista pode ser modificada (ou seja, serviços podem ser inseridos ou removidos). Também vale destacar que o mesmo módulo adaptador pode ser utilizado por diversas aplicações que mantenham lógicas de adaptação similares.

A Fig. 1 mostra uma visão geral do SASeS, que é composto pelos seguintes componentes: (i) Módulo Adaptador, que implementa as fases do *loop* de controle MAPE-K sem a gestão do conhecimento. O conhecimento está representado pela lista de serviços disponíveis para cada aplicação; (ii) Aplicação baseada em serviços que será fornecida aos clientes (ou seja, os consumidores de serviços); (iii) Biblioteca *Wconnect*, cujo papel é disponibilizar os serviços dos provedores aos interessados (ou seja, aplicação orientada a serviços). A seguir, uma breve descrição de cada componente do SASeS é apresentada.

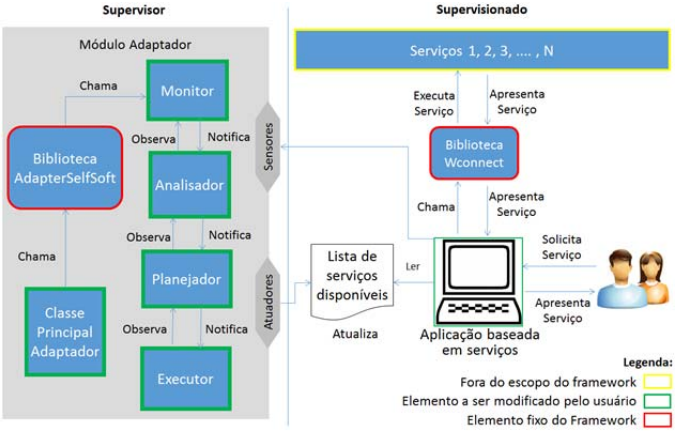

Figura 1. Visão geral do *framework* SASeS.

A. O Módulo Adaptador

Inicialmente o módulo adaptador deve ser acoplado à aplicação baseada em serviços para que possa monitorar os serviços que estão sendo consumidos (Componente Serviço - linha amarela - Figura 1). Os sensores desse módulo ficam coletando informações referentes aos serviços utilizados pela aplicação. Assim, dependendo da condição de um serviço, a lista de serviço pode ser atualizada e a aplicação modificada, sem a percepção de seus clientes. Para conduzir a adaptação, esse módulo implementa o *loop* de controle MAPE. Os componentes, Monitor, Analisador, Planejador e Executor são os responsáveis por gerenciar os ciclos de adaptação (linha verde). Além disso, esse módulo possui a biblioteca AdapterSelfSoft (linha vermelha), cujo objetivo é coordenar as atividades do módulo. O mecanismo utilizado pelo módulo para realizar a troca de mensagens entre as classes é arquivo XML (*eXtensible Markup Language*). O diagrama de classe dessa biblioteca pode ser visto na Figura 2.

A biblioteca AdapterSelfSoft contém classes abstratas que representam os componentes do ciclo MAPE (Monitoring, Analysis, Planning e Execution) e a classe Adapter, que é o gerenciador da adaptação. Assim, o desenvolvedor deve trabalhar na implementação das classes concretas (ou seja, módulos Monitor, Analisador, Planejador e Executor da Figura 1). As classes concretas herdam das classes abstratas uma série de funções que as permitem interagir com o software baseado em serviços.

Sobre a implementação, o módulo de adaptação foi projetado com base no padrão de projeto *Observer* [8]. Assim, cada classe concreta pode notificar e ser notificada sobre a execução de uma atividade de adaptação. Esse padrão permite que múltiplos elementos acessem o mesmo recurso simultaneamente. Além disso, toda classe concreta terá que implementar um dos métodos monitoring, analyze, plan ou execute, referente a fase do ciclo de adaptação associada a essa classe (Fig. 2). Dentro de cada método implementado em uma classe concreta, o desenvolvedor deve adicionar a lógica específica do componente do ciclo MAPE. Para que haja notificação entre os componentes desse ciclo, um método notify foi implementado. Dessa forma, quando um componente é notificado, o método update é invocado para que os métodos analyze, plan e execute sejam invocados. Sobre essa funcionalidade vale destacar que as classes Monitoring e Executing, por representarem a primeira e última fase do ciclo, não possuem os métodos uptade e notify respectivamente, como ilustra a Fig. 2.

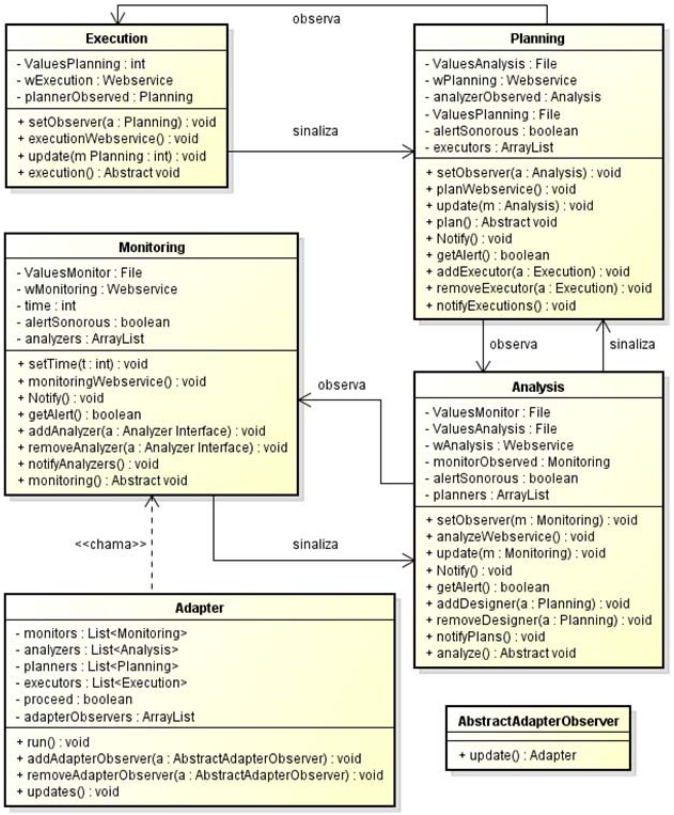

Figura 2. Diagrama de classe da biblioteca AdapterSelfSoft.

Outro diferencial no uso desse padrão *Observer* é a maneira como estão organizados os componentes do módulo. Por exemplo, um monitor pode ser observado e pode notificar vários analisadores, cada um desses analisadores, por sua vez, pode ser observado e pode notificar vários planejadores. Além disso, um planejador pode ser observado e notificar vários executores, no entanto é comum que apenas um executor observe e seja notificado por apenas um planejador.

A Fig. 3 mostra um exemplo da árvore de associações possíveis entre monitores, analisadores e planejadores. A execução segue a regra de uma busca em profundidade. No exemplo ilustrado nessa figura é possível observar que exis-

tem três analisadores que observam o mesmo monitor e para cada um dos analisadores têm-se três planejadores que os observam. Nessa figura foram omitidos os executores, mas pode considerar que cada planejador é observado por um ou mais executores. Toda vez que um planejador for executado, os executores que o observam também serão executados. Assim, quando a execução do monitor (M1) for finalizada, o primeiro Analisador (A1) é notificado, que por sua vez notifica o primeiro planejador (P11). Quando P11 terminar sua execução, o segundo planejador (P12) será executado e, após P12 concluir sua execução, o terceiro planejador (P13) será executado. Ao fim da execução de P13 começa a execução do Analisador 2 (A2), que será executado seguindo a mesma estratégia utilizada por A1. Ao fim da execução de cada planejador, e, por conseguinte, dos executores que os observam, pode-se considerar que uma adaptação foi executada.

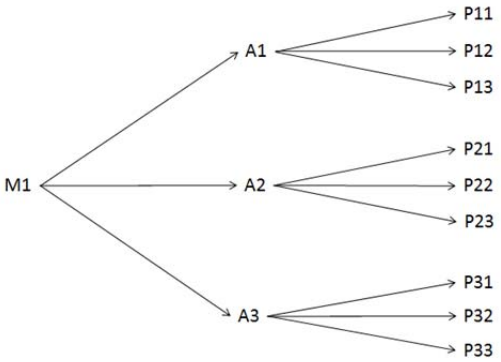

Figura 3. Exemplo de possíveis associações entre monitores, analisadores e planejadores.

Por fim, vale ressaltar que a biblioteca AdapterSelfSoft possui a lista de Monitores, Analisadores, Planejadores e Executores do sistema para que uma atividade seja por ela conduzida. Além disso, pode-se dizer que essa biblioteca representa o comportamento concreto do módulo adaptador. Para isso, a classe Adapter implementa o método run, que tem por objetivo inicializar a execução dos monitores e, consequentemente, todo o processo do ciclo de adaptação.

B. O módulo de aplicação

O módulo de aplicação é composto por uma classe principal de execução (Main) e uma classe de apresentação (View). A classe Main é composta pelos serviços que fornecem as principais funcionalidades do sistema, ou seja, representa a execução da aplicação baseada em serviços. Nessa classe existe um laço de execução em que os serviços são chamados para adaptação. A classe View é usada para separar a lógica da aplicação da lógica de apresentação. Essa classe é opcional e o usuário pode confeccionar a interface usando outras tecnologias (por exemplo, *Java Swing* ou *JavaFX*) [20]. Por fim, vale destacar que a classe principal usa somente os serviços que são informados pela classe responsável pelo planejamento, presente no módulo adaptador. Essa estratégia permite que a aplicação seja desacoplada dos serviços, permitindo assim que o serviço seja dinâmico, ou seja, que possa ser alterado sem que a execução da aplicação seja interrompida.

C. A Biblioteca Wconnect

Essa biblioteca é composta por três classes (Webservice, WSServices e WFuncs) e uma interface (WSInterface), como mostra a Fig. 4. A classe Webservice representa um serviço web, cujos atributos representam as informações necessárias para acessar o serviço. Essa classe possui um método chamado initializerBySocket, que instancia tais atributos via *socket*. Esse método permite que o desenvolvedor não utilize arquivos XML em sua implementação para adaptação de serviços.

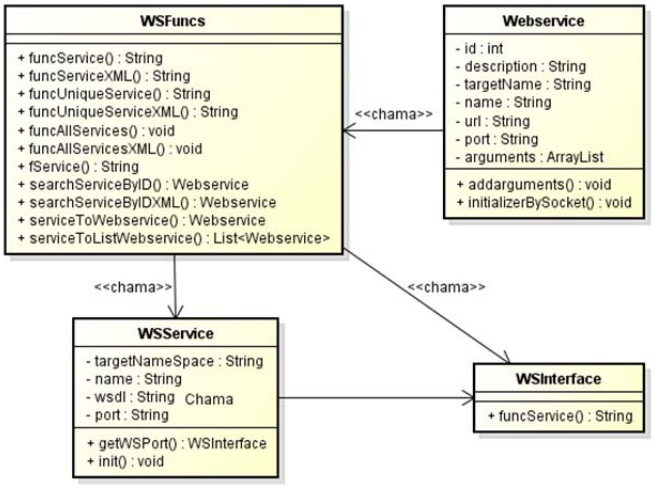

Figura 4. Diagrama de classe da biblioteca Wconnect.

A classe WSService e a interface WSIterface são instanciadas pelo *framework* JAX-WS (do inglês, *Java API for XML Web Services*) para permitir a comunicação com os serviços. A classe WSFuncs contém um conjunto de métodos que identificam os serviços e os atributos que estes necessitam para serem executados/invocados. Essa classe é usada para realizar a conexão com os serviços. Por fim, vale destacar que a classe WSFuncs foi desenvolvida para abstrair a complexidade de implementação do *framework* JAX-WS, pois o desenvolvedor pode utilizar serviços web sem que conhecimentos prévios sobre tal *framework* sejam adquiridos.

D. Metodologia de Utilização do SASeS

Para facilitar a atividade de desenvolvimento de sistemas autoadaptativos orientados a serviços, assim como orientar os engenheiros de software no uso do *framework* proposto neste artigo, uma metodologia de desenvolvimento foi elaborada. A Fig. 5 ilustra as etapas a serem seguidas pelo desenvolvedor. Nas três primeiras etapas são configuradas as classes do módulo Adaptador e na quarta etapa é configurada a classe principal do módulo de aplicação. A seguir uma breve descrição de cada etapa é apresentada.

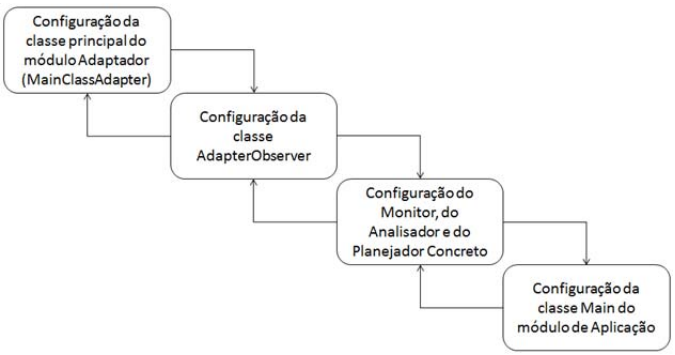

Figura 5. Metodologia para utilização do SASeS.

Como passo preliminar, o engenheiro de software deve acoplar as bibliotecas Wconnect, AdapterSelfSoft e a XStream ao projeto da aplicação que está sendo desenvolvido. Em seguida, deve sistematicamente conduzir a configuração das seguintes etapas:

- **Módulo adaptador:** esse módulo pode ser configurado de duas maneiras pelo engenheiro de software: (i) incluir a lógica de cada fase de adaptação nas classes concretas de monitoramento, análise, planejamento e execução; ou (ii) adicionar as lógicas aos serviços web para que cada classe concreta execute apenas tais serviços;
- **Classe AdapterObserver:** essa classe contém a lógica de parada do adaptador, além de outras funções específicas desse adaptador (incluindo o critério de parada do processo de adaptação);
- **Monitor, Analisador, Planejador e Executor:** estes representam os objetos concretos do ciclo MAPE [15]. Além disso, vale ressaltar que esses objetos implementam o padrão *Observer* e possuem métodos específicos para notificar e serem notificados durante o ciclo de adaptação;
- **Módulo de adaptação:** esse módulo contém a lista de serviços informada pelo planejador concreto. Esse módulo permite que serviços sejam identificados nessa lista para que objetos, do tipo Webservice, possam ser instanciados e devolvidos aos interessados (ou seja, solicitante da adaptação).

IV. Estudo de Caso

Para mostrar a aplicabilidade do *framework* SASeS, dois estudos de casos foram elaborados. Sobre tais estudos, vale destacar que os mesmos foram replicados ou adaptados, conforme especificação fornecida pelos autores (trabalhos relacionados). Ao contrário do presente trabalho, os demais não disponibilizam código fonte de seus *frameworks* e/ou ferramentas. Portanto uma análise comparativa, qualitativa ou quantitativa, não foi realizada em função dessa limitação. A Seção IV-A apresenta um estudo de caso para um sistema de descoberta dinâmica de serviços, enquanto a Seção IV-B mostra um sistema orientado a serviços de assistência médica remota. Vale destacar que ambos os estudos estão organizados conforme arquitetura apresentada na Fig. 1. A lógica de adaptação (ou seja, monitoramento, análise, planejamento) é disparada pelo módulo adaptador. Dessa forma, as listas de serviços são atualizadas e disponibilizadas para futuras adaptações. Para essas aplicações não foi utilizado o componente de execução. As atualizações das listas de serviços foram feitas pelo componente de planejamento.

A. Sistema de Descoberta de Serviços

Esse estudo de caso (disponível em: http://goo.gl/DsU05x) é uma adaptação do estudo reportado por [18], que aborda um sistema de descoberta de serviços em um aeroporto. O sistema tem por objetivo verificar os serviços disponíveis no ambiente e apresentar ao usuário somente aqueles que realmente estão funcionando. Assim, o sistema autogerencia a lista de serviços, prevenindo o usuário de acessar um serviço indisponível. Para isso, dois tipos de adaptação são abordados: (i) remoção de serviços que tiveram sua execução interrompida; e (ii) adição de novos serviços. Por motivos de escopo, apenas o primeiro caso será apresentado nesta Seção, como mostra a Fig. 6. A seguir, uma breve descrição do fluxo de execução é apresentada.

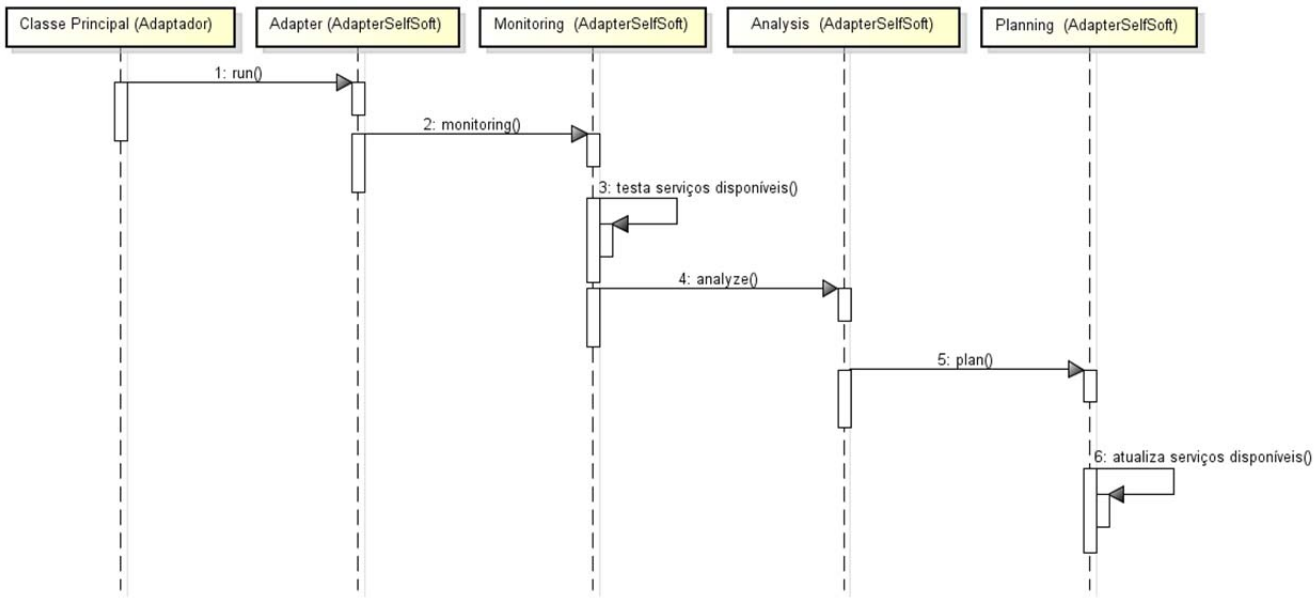

Figura 6. Diagrama de sequência da remoção de serviços.

O processo começa quando a classe principal do módulo adaptador executa o método run da classe Adapter. Esse método dispara a execução do monitor concreto, que lê a lista de serviços disponíveis e testa se todos estão disponíveis para execução. Os serviços não operacionais são escritos no arquivo de saída do monitor, indicando que devem ser removidos. Em seguida, o analisador concreto é notificado para que a lista de serviços não disponíveis seja atualizada. Como o ana-

lisador tem conhecimento sobre os serviços que necessitam ser removidos, o planejador concreto é apenas notificado para que essa operação seja efetivada.

Embora não tenha sido representada por meio de um diagrama, a adição de serviços ocorre de maneira similar. Os serviços disponíveis no ambiente são coletados e inseridos no arquivo de saída do monitor. Em seguida, o planejador é notificado e os serviços reportados pelo analisador são adicionados à lista de serviços disponíveis que irão ser usados pela aplicação.

B. Sistema de Assistência Médica a Distância

Este estudo de caso (disponível em: http://goo.gl/HDw3sy) é uma replicação do estudo apresentado por [30]. Basicamente, esse estudo trata de um sistema de assistência médica a distância, que é composto por três subsistemas: (i) o subsistema de análise, que analisa a situação do paciente e indica se ele está curado, ou se algum outro subsistema deve ser acionado; (ii) o subsistema de medicação que tem por objetivo informar, quando acionado, a dosagem do medicamento a ser utilizado pelo paciente; e (iii) o subsistema de alarme, que pode ser acionado (automaticamente) quando o paciente encontra-se em estado grave ou caso o paciente faz o acionamento manualmente. Quando o subsistema de alarme é acionado, a intervenção médica local é uma condição necessária. Caso um desses subsistemas falhe, outro serviço deve ser localizado como forma de preservar a integridade do sistema.

Diante do cenário apresentado, optou-se por tratar falhas especificamente no subsistema de alarme, por este ser considerado um subsistema mais crítico. Basicamente, a adaptação proposta consiste em selecionar um dos três serviços que implementam o subsistema de alarme. Segundo [31], esse tipo de adaptação pode ser definido como uma maneira/abordagem de autoverificação.

A sequência de operações executadas pelo módulo adaptador é ilustrada na Fig. 7. O processo começa quando a classe principal do módulo adaptador executa o método run da classe Adapter. Esse método dispara a execução do monitor concreto que executa o método monitoringWebservice para chamar o serviço monitor. O serviço monitor obtém a probabilidade de falha de cada serviço de subsistema de alarme e adiciona essa informação ao arquivo de saída do monitor. Em seguida, o monitor concreto notifica o analisador para que o método analyzeWebservice seja invocado. O serviço analisador usa uma ferramenta de checagem de modelos probabilísticos chamada PRISM [2], que lê uma cadeia de Markov de tempo discreto e adiciona a probabilidade de falha de cada subsistema de alarme obtido pelo serviço monitor. Caso essa probabilidade seja maior ou menor do que determinado valor, a ferramenta gera um valor retorno booleano (verdadeiro ou falso) [30]. Assim, o serviço analisador seleciona o serviço do subsistema de alarme que apresenta um valor verdadeiro e adiciona essa informação ao arquivo de saída do analisador. Em seguida, o analisador concreto notifica o planejador para que o método planWebservice seja invocado e a aplicação seja informada que os serviços foram atualizados.

Por fim, a aplicação recebe a informação sobre os sintomas do paciente para que o serviço de subsistema de análise seja invocado. Em seguida, o serviço do subsistema de medicação ou de alarme deve ser chamado. Caso o paciente esteja curado, o sistema encerra sua execução.

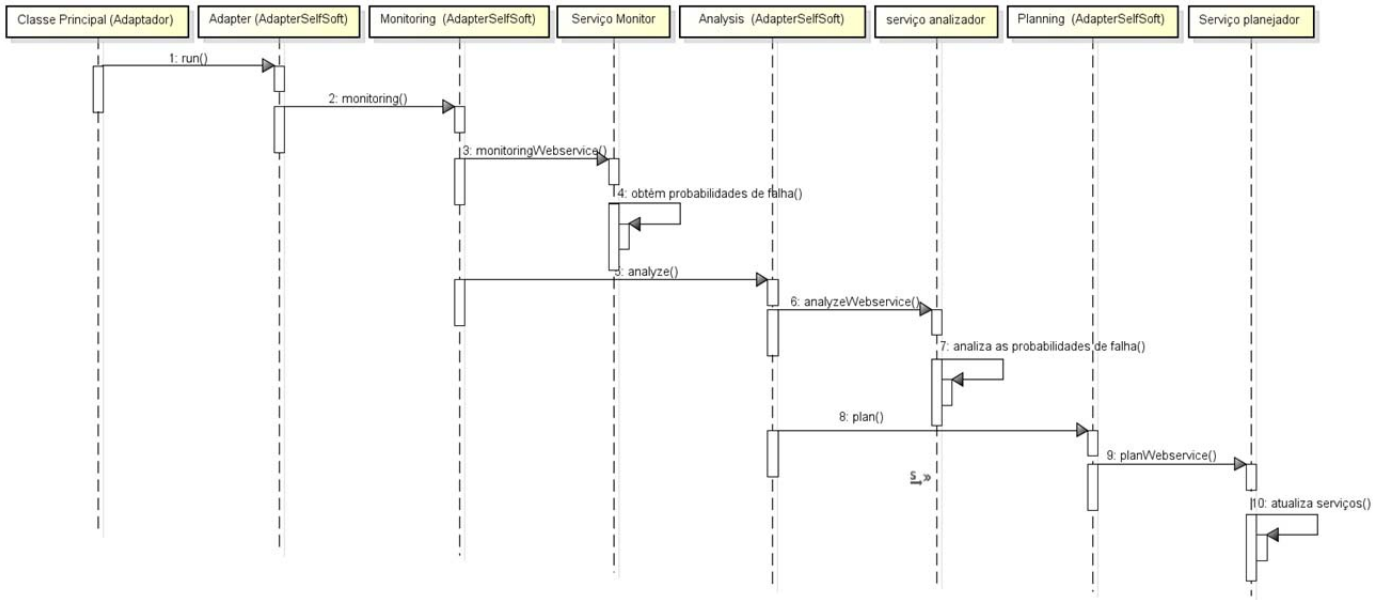

Figura 7. Diagrama de sequência do módulo adaptador do Sistema de Assistência Médica.

V. CONCLUSÕES

Este artigo apresentou o SASeS, um *framework* para auxiliar o desenvolvimento de sistemas autoadaptativos baseados em serviços. Basicamente, o *framework* permite que uma aplicação possa adicionar, remover e trocar, em tempo de execução, um serviço que esteja disponível nas seguintes condições: (i) quando o mesmo apresente falha de execução; ou (ii) quando não possa atender mais as necessidades de seus interessados. Sobre a implementação, o *framework* é baseado no *loop* de controle MAPE-K [15], sendo utilizando somente o ciclo MAPE, e no padrão de projeto *Observer* [8]. A fonte de conhecimento (K) está representada pela lista de serviços, que

é mantida/gerenciada pelo módulo adaptador. Serviços podem ser adicionados ou removidos durante os ciclos de adaptações. A seguir são destacadas as principais contribuições desse artigo:

- Para a comunidade de engenharia de software, provendo um meio aos interessados em desenvolver aplicações autoadaptativas orientadas a serviços;
- Para a área de sistemas autoadaptativos, pois uma estratégia de autoadaptação de serviços foi apresentada neste artigo;
- Para a área de desenvolvimento, pois um *framework* de fácil utilização foi apresentado. Neste sentido, espera-se que sua utilização e, consequentemente, sua difusão seja facilitada, pois tanto o *framework* quanto os estudos conduzidos estão disponíveis para livre acesso;

Em relação às limitações, que também podem ser visualizadas como desafios futuros, falta uma avaliação sobre a garantia de qualidade (do inglês, *Quality of Service* - QoS) nas requisições de serviços. Atualmente, o projetista da aplicação deve garantir, por meios próprios, que tal requisito seja implementado. Por fim, vale ressaltar que o SASeS foi elaborado para atuar em sistemas baseados em serviços. Portanto, não foi feita avaliações sobre adaptabilidade do *framework* para que outros tipos de sistemas possam ser atendidos.

Como trabalhos futuros, três objetivos são apresentados: (i) estender o SASeS para permitir o uso serviços web desenvolvidos em outros *frameworks* (ou seja, não apenas o JAX-WS); (ii) utilizar uma abordagem MDE (do inglês, *Model-Driven Engineering*) integrada ao SASeS para gerar sistemas autoadaptativos baseados em serviços de maneira automática; e (iii) utilizá-lo na implementação de aplicações móveis sensíveis ao contexto e baseadas em serviço.

AGRADECIMENTOS

Os autores receberam suporte financeiro da CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) e PROPe/UNESP (Pró-reitoria de Pesquisa da Universidade Estadual Paulista) para realização deste trabalho.

REFERÊNCIAS

[1] A. Elkhodary, N. Esfahani, and S. Malek. “FUSION: a framework for engineering self-tuning self-adaptive software systems”. In Proceedings of the eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, New York, NY, USA, pp.7-16. 2010;

[2] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, “Prism: A tool for automatic verification of probabilistic systems. In Tools and Algorithms for the Construction and Analysis of Systems, pp 441-444. Springer, 2006;

[3] A. Perini, “Self-adaptive service based applications: Challenges in requirements engineering”, In Sixth International Conference on Research Challenges in Information Science (RCIS), pp. 16-18, 2012;

[4] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, “Software engineering for self-adaptive systems: A research roadmap”. In Software engineering for self-adaptive systems, pages 1–26. Springer. 2009;

[5] 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;

[6] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, “Rainbow: Architecture-based self-adaptation with reusable infrastructure”. Computer, pp. 46-54, 2004;

[7] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, “A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engineering, pages 313–341, 2012;

[8] E. Gamma, R. Helm, R. Johnson and J. Vlissides, “Design patterns: elements of reusable object-oriented software. Pearson Education. 1994;

[9] 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;

[10] F. J. Affonso and E. Y. Nakagawa. “A reference architecture based on reflection for self-adaptive software”. In 7th Brazilian Symposium on Software Components, Architectures and Reuse, pp. 129-138, Brasília/DF, Brazil, 2013.

[11] F. J. Affonso, G. Leite, R. A. P. Oliveira, E. Y. Nakagawa, “A Framework Based on Learning Techniques for Decision-making in Self-adaptive Software”, In . SEKE 2015, pp. 24-29, 2015;

[12] F. J. Affonso, M. C. V. S. Carneiro, E. L. L. Rodrigues, E. Y. Nakagawa, “A Reference Model as Automated Process for Software Adaptation at Runtime”, In IEEE Latin America Transactions, vol.13, no.1, pp. 214-221, Jan. 2015;

[13] 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;

[14] I. Gorton, Y. Liu, and N. Trivedi. “An extensible, lightweight architecture for adaptive J2EE applications”. In Proceedings of the 6th international workshop on Software engineering and middleware (SEM '06). ACM, New York, NY, USA, 47-54, 2006.

[15] 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: 24 de nov. de 2015;

[16] J. Adamczyk, R. Chojnacki, M. Jarząb, and K. Zieliński. “Rule Engine Based Lightweight Framework for Adaptive and Autonomic Computing”. In Proceedings of the 8th international conference on Computational Science, Marian Bubak, Geert Dick Albada, Jack Dongarra, and Peter M. Sloot (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 355-364. 2008;

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

[18] J. Camara, C. Canal, G. Salaun, “Behavioural self-adaptation of services in ubiquitous computing environments” In ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, pp.28-37, 18-19 May 2009;

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

[20] JAVA, “Java Platform, Standard Edition (Java SE)”, on-line, 2015. Disponível em: http://www.oracle.com/us/technologies/java/standard-edition/overview/index.html, Acessado em: 24 de nov. de 2015;

[21] M. Abufouda, “Quality-aware approach for engineering self-adaptive software systems.”, Third International Conference on Information Technology Convergence and Services, 2014;

[22] 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;

[23] P. Inverardi, F. Mancinelli, and M. Nesi. “A declarative framework for adaptable applications in heterogeneous environments”. In Proceedings of the ACM Symposium on Applied Computing, pages 1177–1183, 2004;

[24] P. Inverardi and M. Mori. “Model checking requirements at run-time in adaptive systems”. In Proceedings of the 8th workshop on Assurances for self-adaptive systems, p. 5-9. ACM. 2011.

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

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

[27] P. Oreizy, D. Heimbigner, G. Johnson, M. M. Gorlick, R. N. Taylor, A. L. Wolf, N. Medvidovic, D. S. Rosenblum, and A. Quilici, “An architecture-based approach to self-adaptive software”. IEEE Intelligent systems, pp. 54-62, 1999.

[28] R. Asadollahi, M. Salehie, and L. Tahvildari, L. “Starmx: A framework for developing self-managing java-based systems”. ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, pages 58-67. 2009;

[29] R. Calinescu, and Y. Rafiq, Y. “Using intelligent proxies to develop

self-adaptive service-based systems". In International Symposium on Theoretical Aspects of Software Engineering (TASE), ages 131–134. 2013;
[30] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, R. "Self-adaptive software needs quantitative verification at runtime. Communications of the ACM, 55(9): pages 69–77. 2012;
[31] R. Calinescu, K. Johnson, and Y. Rafiq, Y. "Developing self-verifying service-based systems. In IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), pages 734–737. 2013;
[32] S. Vinoski, "A time for reflection [software reflection]", Internet Computing, IEEE, vol. 9, no. 1, pp. 86-89, 2005;
[33] 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;
[34] 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;
[35] Y. Shi, L. ZaoQing, W. JunLi, and W. FuDi, "A reflection mechanism for reusing software architecture", QSIC 2006, pp. 235-243, oct. 2006;

Evilasio Costa Junior, possui graduação em Ciência da Computação pela Universidade Estadual do Ceará(2010) e mestrado em Ciências da Computação pela Universidade Estadual do Ceará(2015). Atualmente é Professor Substituto do Curso de Bacharelado em Ciência da Computação da Universidade Estadual do Ceará. Tem experiência na área de Ciência da Computação. Atuando principalmente nos seguintes temas: Sistemas autoadaptativos, Serviços e Sistemas autoadaptativos baseados em serviços.

Paulo Henrique Mendes Maia, possui graduação em Ciência da Computação pela Universidade Federal do Ceará (2002), mestrado em Ciências da Computação pela Universidade Federal do Ceará (2004) e doutorado em Computação pelo Imperial College London (2010). Atualmente é professor adjunto do curso de Ciência da Computação da Universidade Estadual do Ceará (UECE) e pesquisador/orientador do Mestrado Acadêmico em Ciência da Computação (MACC) da UECE. Coordena o Grupo de Engenharia de Software Autoadaptativo e Distribuído (GESAD). Tem experiência na área de Ciências da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: engenharia dirigida por modelos, reengenharia e engenharia reversa de software, desenvolvimento orientado a aspectos, computação em nuvens e sistemas autoadaptativos.

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; UML; Framework de Persistência (Hibernate/JPA) e Sistemas de Software Autoadaptativos.