UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

EVILASIO COSTA JUNIOR

SASeS: UM FRAMEWORK PARA O DESENVOLVIMENTO DE APLICAÇÕES AUTOADAPTATIVAS BASEADAS EM SERVIÇOS

FORTALEZA – CEARÁ

2015

EVILASIO COSTA JUNIOR

SASeS: UM FRAMEWORK PARA O DESENVOLVIMENTO DE APLICAÇÕES AUTOADAPTATIVAS BASEADAS EM SERVIÇOS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Orientadores: Jorge Luiz de Castro e Silva
e Paulo Henrique M. Maia

FORTALEZA – CEARÁ

2015

Dados Internacionais de CatalogaÃ§Ã£o na PublicaÃ§Ã£o

Universidade Estadual do CearÃ¡

Sistema de Bibliotecas

Costa Junior, Evilasio Costa Junior.
SASeS: Um framework para o desenvolvimento de aplicaÃ§Ãµes auto-adaptativas baseadas em serviÃ§os [recurso eletrï¿½nico] / Evilasio Costa Junior Costa Junior. ï¿½ 2015.
1 CD-ROM: il.; 4 ï¿½ pol.

CD-ROM contendo o arquivo no formato PDF do trabalho acadÃªmico com 65 folhas, acondicionado em caixa de DVD Slim (19 x 14 cm x 7 mm).

DissertaÃ§Ã£o (mestrado acadÃªmico) ï¿½ Universidade Estadual do CearÃ¡, Centro de CiÃªncias e Tecnologia, Mestrado AcadÃªmico em CiÃªncia da ComputaÃ§Ã£o, Fortaleza, 2015.
Ã¡rea de concentraÃ§Ã£o: CiÃªncia da ComputaÃ§Ã£o.
OrientaÃ§Ã£o: Prof. Ph.D. Jorge Luiz de Castro e Silva.
CoorientaÃ§Ã£o: Prof. Dr. Paulo Henrique Mendes Maia.

1. Sistemas autoadaptativos. 2. ServiÃ§os. 3. Sistemas autoadaptativos baseados em serviÃ§os. 4. Ciclo de AdaptaÃ§Ã£o. 5. Framework. I. TÃ-tulo.

EVILASIO COSTA JUNIOR

SASeS: UM FRAMEWORK PARA O DESENVOLVIMENTO DE APLICAÇÕES AUTOADAPTATIVAS BASEADAS EM SERVIÇOS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Aprovada em:

BANCA EXAMINADORA

Jorge Luiz de Castro e Silva (Orientador)
Universidade Estadual do Ceará – UECE

Paulo Henrique M. Maia (Orientador)
Universidade Estadual do Ceará – UECE

Mariela Inés Cortés
Universidade Estadual do Ceará – UECE

Frank José Affonso
Universidade Estadual Paulista - UNESP

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

Primeiramente a Deus que permitiu que tudo isso acontecesse, ao longo de minha vida, e não somente nestes anos como universitário, mas que em todos os momentos é o maior mestre que alguém pode conhecer.

Aos meus pais, pelo amor, incentivo e apoio incondicional.

Obrigado a meus irmãos, que sempre me apoiaram durante essa empreitada.

A esta universidade, seu corpo docente, direção e administração que sempre ofereceu um ambiente propício para meus estudos e pesquisas, demonstrando uma acendrada confiança no mérito e ética aqui presentes.

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender. A palavra mestre, nunca fará justiça aos professores dedicados aos quais sem nominar terão os meus eternos agradecimentos.

“O aspecto mais triste da vida de hoje é que a ciência ganha em conhecimento mais rapidamente que a sociedade em sabedoria.”

Isaac Asimov

RESUMO

Sistemas auto-adaptativos baseados em serviços modificam seu próprio comportamento em resposta a mudanças no seu ambiente operacional de tal modo que o serviço adequado pode ser selecionado de acordo com os requisitos do software. O desenvolvimento deste tipo de software é uma tarefa complexa e não trivial, uma vez que o ciclo de vida nunca termina e o software precisa estar em constante auto-avaliação. Este trabalho propõe o SASeS, um framework para o desenvolvimento de aplicações auto-adaptativas baseadas em serviços. Este é um framework genérico com base em padrões de web services e padrões de projeto que suporta tarefas comuns no desenvolvimento de cada sistema. O SASeS facilita a criação do gerenciador do ciclo de adaptação e da aplicação adaptável. A arquitetura do framework é descrita neste trabalho, bem como são apresentados estudos de caso que demonstram sua aplicabilidade.

Palavras-chave: Sistemas autoadaptativos. Serviços. Sistemas autoadaptativos baseados em serviços. Ciclo de Adaptação. Framework.

ABSTRACT

Self-adaptive service-based systems modify their own behavior in response to changes in its operating environment such that the most appropriate service can be selected according to requirements. The development of such software is a complex and non-trivial task, since its life cycle never ends and the software needs is constantly self-evaluating. This work proposes SASeS, a framework for developing self-adaptive service-based applications. It is a generic framework based on web services standards and design patterns that supports common tasks in the development of such systems. SASeS facilitates creating the management adaptation loop and the adaptable application. The framework architecture is described in this work and case studies are presented to demonstrate its aplicability.

Keywords: Self-adaptive systems. Services. Self-adaptive service-based systems. Adaptation Loop. Framework

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMT *Autonomic Management Toolkit*.

ASF *Adaptive Server Framework*.

CPU *Central Processing Unit*.

HTML *HyperText Markup Language*.

MAPE *Monitoring, Analysis, Planning and Execution*.

PCTL *Probabilistic Computation Tree Logic*.

SOAP *Simple Object Access Protocol*.

UDDI *Universal Description, Descovering and Integration*.

URL *Uniform Resource Locator*.

WSDL *Web Service Description Language*.

XML *eXtensible Markup Language*.

SUMÁRIO

1 INTRODUÇÃO

Tradicionalmente a Engenharia de Software tem focado em agir sobre o software em tempo de desenvolvimento, investigando como o processo de desenvolvimento de software pode ser entendido, melhorado e automatizado (BARESI; GHEZZI, 2010). As fases de desenvolvimento e execução de software são rigorosamente separadas uma da outra, e a modificação de softwares já em execução é geralmente feita por reimplementação dos componentes de software a serem alterados (SZVETITS; ZDUN, 2013).

No entanto, a complexidade dos sistemas de software atuais levou a comunidade científica a investigar maneiras de gerenciamento e evolução de softwares. Adicionalmente, os sistemas de software tornaram-se cada vez mais versáteis, flexíveis, resilientes, confiáveis, eficientes em energia, recuperáveis, customizáveis, configuráveis, e autootimizáveis através da adaptação às mudanças que podem ocorrer em seus contextos operacionais, ambientes e requisitos (KRAMER; MAGEE, 2007).

Muitos pesquisadores de Engenharia de Software tem sido motivados a desenvolver técnicas e ferramentas que permitam lidar com a complexidade de projetar, construir e testar sistemas de software de grande porte. Grande parte desses avanços têm dependido fortemente de intervenção manual (ANDERSSON et al., 2009). Porém, muitas vezes, as mudanças nas aplicações não podem ser manuseados off-line e exigem que o software adapte seu comportamento de forma dinâmica, para continuar a garantir a qualidade de serviço desejado (BARESI; GHEZZI, 2010).

O aparecimento de sistemas altamente distribuídos, móveis, e embarcados que são muitas vezes de longa duração, tornou cada vez mais inviável o gerenciamento e controle manual de sistemas. Isso fez com que surgisse uma nova classe de sistemas de software, o chamados sistemas autoadaptativos (ANDERSSON et al., 2009).

Os sistemas autoadaptativos são capazes de modificar o seu comportamento e/ou estrutura através do ajuste de parâmetros em resposta à sua percepção de meio ambiente e do próprio sistema sem a intervenção humana. O estudo desses sistemas tornou-se um tópico de pesquisa importante em muitas áreas, como computação autonômica, computação confiável (*dependable*), sistemas embarcados, redes móveis ad hoc, robótica, sistemas multi-agente, aplicações *peer-to-peer*, redes de sensores, arquiteturas orientadas a serviços, e computação ubíqua (CHENG et al., 2009a).

Outra tecnologia que vem sendo utilizada para trabalhar no complexo contexto das

aplicações atuais são os sistemas baseados em serviços, que de acordo com (CALINESCU et al., 2012), são aplicações de software construídas a partir de serviços de baixo acoplamento de vários provedores. Esses serviços são executados de maneira autônoma remotamente e podem ser descobertos de forma dinâmica, sendo invocados pela aplicação para executar alguma tarefa. Assim sendo, os projetistas podem construir aplicações através da composição de serviços, possivelmente oferecidos por terceiros (NITTO et al., 2008).

Os sistemas baseados em serviços são usados em vários domínios de aplicação, incluindo *e-commerce*, *online banking*, e *health care*. Esses sistemas operam em ambientes caracterizados por mudanças frequentes. Como resultado, sua eficácia depende cada vez mais da sua capacidade de autoadaptação.

Atualmente, trabalhos como Perini (2012) e Nitto et al. (2008), estudam o desenvolvimento de softwares autoadaptativos baseados em serviços, que são softwares baseados em serviços que utilizam as ideias de criação de softwares autoadaptativos. Uma forma de criar sistemas autoadaptativos baseados em serviços é selecionar dinamicamente os serviços que implementam as suas operações, a partir de conjuntos de serviços funcionalmente equivalentes associados à diferentes níveis de desempenho, confiabilidade e custo (CALINESCU et al., 2012).

1.1 MOTIVAÇÃO

Mesmo com o crescente número de trabalhos relacionados a softwares autoadaptativos e softwares autoadaptativos baseados em serviços, ainda existe bastante espaço para pesquisa na área, já que os modelos atuais de desenvolvimento não são adequados a trabalhar com estes tipos de softwares.

Alguns trabalhos propuseram arquiteturas (GARLAN et al., 2004), frameworks e ferramentas para o desenvolvimento de sistemas autoadaptativos (ASADOLLAHI et al., 2009)(INVERARDI et al., 2004)(GORTON et al., 2006)(ADAMCZYK et al., 2008)(BURG; DOLSTRA, 2011)(AUTILI et al., 2010) e, mais especificamente, sistemas autoadaptativos baseados em serviços (CALINESCU et al., 2013). Porém, segundo Salehie e Tahvildari (2009), ainda há oportunidades para mais pesquisa na área de desenvolvimento de novos frameworks, ferramentas e linguagens adequadas a criação de softwares autoadaptativos e softwares autoadaptativos baseados em serviços. Isso se deve a melhorias constantes nas técnicas que abordam o gerenciamento de recursos autônomos e monitoramento dos atributos de qualidade para estes sistemas.

Diante disso, este trabalho contribui para pesquisa de sistemas autoadaptativos e, mais especificamente, a de sistemas autoadaptativos baseados em serviços ao apresentar SASeS, um framework para o desenvolvimento de sistemas autoadaptativos baseados em serviços. O framework, que foi desenvolvido em Java, permite a criação de um módulo adaptador externo que implementa parte do ciclo MAPE (*Monitoring, Analysis, Planning and Execution*) (COMPUTING et al., 2006) e a aplicação baseada em serviços. O framework utiliza o padrão de projeto Observer (GAMMA, 2007) para auxiliar o gerenciamento das fases do ciclo MAPE. Além disso, SASeS provê uma interface para descrever os possíveis serviços utilizados que contém as funcionalidades do sistema. Por fim, é ilustrado o funcionamento do framework apresentando a utilização do mesmo em três estudos de caso.

1.2 OBJETIVOS

Nas subseções abaixo estão descritos os objetivos gerais e específicos do presente trabalho.

1.2.1 Objetivo Geral

Esse trabalho objetiva prover um framework para auxiliar o desenvolvimento de softwares autoadaptativos baseados em serviços.

1.2.2 Objetivos específicos

a) Análise e modeladem do framework proposto.
b) Implementação do framework proposto.
c) A implementação de uma ferramenta para construção de arquivos XML (*eXtensible Markup Language*) utilizados por aplicações criadas com base no framework proposto.
d) Demonstração da aplicabilidade do framework proposto através da aplicação do mesmo em estudos de caso.

1.3 ESTRUTURA DO TRABALHO

O restante deste trabalho está organizado da seguinte maneira:

- O **Capítulo 2** apresenta o referencial teórico, que aborda os conceitos de softwa-

res autoadaptativos, as fases do ciclo de adaptação do software, tipos de abordagens de adaptação, os conceitos de softwares baseados em serviços e de softwares autoadaptativos baseados em serviços.

- O **Capítulo 3** expõe os trabalhos relacionados e faz uma análise comparativa dos mesmos com este trabalho.
- O **Capítulo 4** mostra o framework proposto, sua arquitetura, quais os elementos que o compõem e como ele é utilizado.
- O **Capítulo 5** apresenta os estudos de caso que demonstram a aplicabilidade do framework SASeS.
- O **Capítulo 6** expõe as considerações finais, incluindo a autoavaliação do trabalho e as propostas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta o referencial teórico que fundamenta a proposta deste trabalho. Inicialmente, os conceitos de softwares autoadaptativos e as etapas de construção do mesmo são apresentados na seção 2.1. Em seguida, na seção 2.2, são expostos os conceitos sobre abordagens de adaptação internas e externas para softwares autoadaptativos. Por fim, na seção 2.3, são apresentados os conceitos de softwares baseados em serviços e de softwares autoadaptativos baseados em serviços.

2.1 SOFTWARES AUTOADAPTATIVOS

Os softwares de hoje em dia precisam atuar em ambientes cada vez mais dinâmicos e complexos. Nesses ambientes, a manutenção desses softwares tornou-se um gargalo devido ao alto custo associado. Assim sendo, o desenvolvimento e a evolução de tais sistemas tornaram-se um desafio para a academia e para a indústria. A autoadaptação surge como uma alternativa para minimizar a necessidade de intervenção humana e para promover um maior grau de confiabilidade a tais sistemas de software (INVERARDI; TIVOLI, 2009).

Segundo Oreizy et al. (1999), softwares autoadaptativos modificam seu próprio comportamento em resposta a mudanças no seu ambiente operacional. Pelo ambiente operacional entende-se qualquer coisa observável pelo sistema de software, tais como entradas de usuário, dispositivos externos de hardware e sensores. O ponto chave dos softwares autoadaptativos é que o ciclo de vida do software nunca termina, pois o software precisa estar constantemente se avaliando e alterando de acordo com as mudanças do ambiente e exigências de usuário.

Em um outro ponto de vista, a adaptação é relacionada à evolução. Buckley et al. (2005) fornecem uma taxonomia de evolução com base no objeto de mudança (onde - *where*), as propriedades do sistema (o que - *what*), as propriedades temporais (quando - *when*), e suporte a mudança (como - *how*).

Já Inverardi e Tivoli (2009) correlaciona a adaptação com base nas questões: (i) *why* - Porque é preciso adaptar? (ii) *what* - O que é preciso adaptar? (iii) *when* - Quando a adaptação deve acontecer? (iv) *who* - Quem gerencia a adaptação? . A primeira questão é referente à causa por traz da adaptação. Geralmente é necessário adaptar devido a algum requisito que não está sendo atendido. A segunda é referente a que componente do sistema deve ser alterado. A terceira está relacionada ao momento no ciclo de vida do software em que a adaptação deve ocorrer. Isso

não significa que a mudança ocorre em tempo de execução. Essa questão define se o sistema de software será estático ou dinâmico. Por fim, a quarta questão é relacionada a qual componente gerencia as etapas necessárias à adaptação do software.

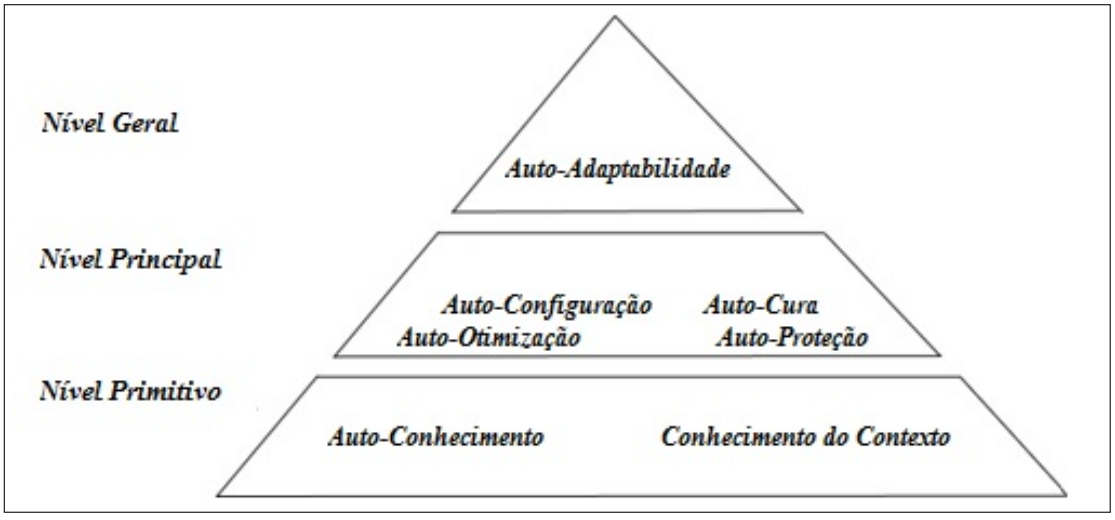
Em Salehie et al. (2009) foi apresentado um mapeamento da taxonomia proposta por Buckley et al. (2005) para o domínio de software autoadaptativo e foi proposto um modelo conceitual para mudanças de adaptação com base em mecanismos de autoadaptação biológica. Adaptações estáticas e dinâmicas, relacionadas à dimensão temporal desse ponto de vista, são mapeadas para evolução em tempo de compilação e evolução em tempo de execução, respectivamente. Por esta razão, a adaptação dinâmica é por muitas vezes chamada de evolução dinâmica.

Segundo Calinescu et al. (2012), para um software necessitar evoluir ou se adaptar deve ocorrer uma violação que pode acontecer por três razões: (i) a implementação do software não está satisfazendo as especificações; (ii) o conhecimento do ambiente diverge das suposições de domínio; e (iii) os requisitos não estão de acordo com as necessidades do usuário.

2.1.1 Propriedades Auto-*

Existem uma série de propriedades associadas aos softwares autoadaptativos, que são conhecidas como "propriedades auto-*". Em Salehie e Tahvildari (2009), uma hierarquia com base nas oito propriedades inicialmente introduzidas pela IBM é apresentada, como mostra a Figura 1.

Figura 1 – Hierarquia de Propriedades Auto-*



Fonte: (SALEHIE; TAHVILDARI, 2009)

Nesta hierarquia, como apresentado na Figura 1, o nível Geral contém as propriedades globais dos softwares autoadaptativos ligados à autoadaptação. O nível Principal contém as propriedades inerentes à computação autonômica, que são baseadas nos mecanismos de autoadaptação biológica. Por fim, o nível Primitivo está ligado ao conhecimento interno e externo do sistema.

A autoadaptabilidade pode ser compreendida como uma propriedade que engloba um conjunto de propriedades relacionadas ao gerenciamento de componentes do software e de suas funcionalidades. Dentre essas propriedades estão a autogestão, autocontrole, autoavaliaçao.

Segundo Kephart e Chess (2003), a autogestão é a capacidade que um sistema de software possui para se auto gerir com base em objetivos pré-definidos. Um software autogerenciável deve controlar as mudanças em seus componentes, condições externas e falhas de hardware e software, abrangendo também as capacidade de autoadministração e automanutenção.

Kokar et al. (1999) entendem o autocontrole como a capacidade que o software tem de controlar as mudanças em componentes internos, além dos estímulos externos captados por este software. Em Laddaga (2006), os autores definem a autoavaliação como a capacidade que o software tem de avaliar seu próprio comportamento a fim de definir se há como alcançar um desempenho melhor.

A autoconfiguração é a capacidade que software tem de reconfigurar automaticamente e de forma dinâmica em resposta a mudanças através da instalação, atualização, integração e composição ou decomposição de entidades de software (SALEHIE; TAHVILDARI, 2009). Segundo Sterritt e Bustard (2003), a autootimização é a capacidade que o software tem de gerir o desempenho e a alocação de recursos, a fim de satisfazer as necessidades dos diferentes usuários. O tempo de resposta, *throughput*, utilização e carga de trabalho são exemplos de preocupações ligadas a esta propriedade.

Em Gorla et al. (2010), os autores interpretam a autocura como a capacidade que um sistema de software tem de identificar, diagnosticar e corrigir falhas. Em Salehie e Tahvildari (2009), a autoproteção foi definida como a capacidade de detectar falhas de segurança e recuperar o sistema a partir dos seus efeitos. Essa propriedade tem duas funções: defender o sistema contra ataques maliciosos e antecipar problemas e tomar medidas para evitá-los ou para mitigar seus efeitos.

Segundo Hinchey e Sterritt (2006), quando um sistema de software possui a propriedade de autoconhecimento, isso significa que esse sistema é consciente de seus estados e

de seu comportamento. Essa propriedade é baseada no automonitoramento, que reflete o que é monitorado. Compreende-se que um sistema é consciente de seu contexto quando esse conhece seu ambiente operacional (PARASHAR; HARIRI, 2005).

Em (SALEHIE; TAHVILDARI, 2009) foram correlacionadas as propriedades auto-* com fatores de qualidade do software, ou seja, ao implementar cada propriedade auto-* é necessário levar em consideração diversos fatores de qualidade. Na Tabela 2 são apresentadas algumas correlações entre propriedades auto-* e fatores de qualidade.

Tabela 2 – Fatores de qualidade x Propriedades auto-*

Fator QoS/ Propriedade	Autoconfiguração	Autocura	Auto-otimização	Autoproteção
Manutenibilidade	x	x		
Funcionalidade	x		x	x
Portabilidade	x			
Usabilidade	x			
Confiabilidade	x	x		x
Disponibilidade		x		
Sobrevivência		x		
Eficiência			x	

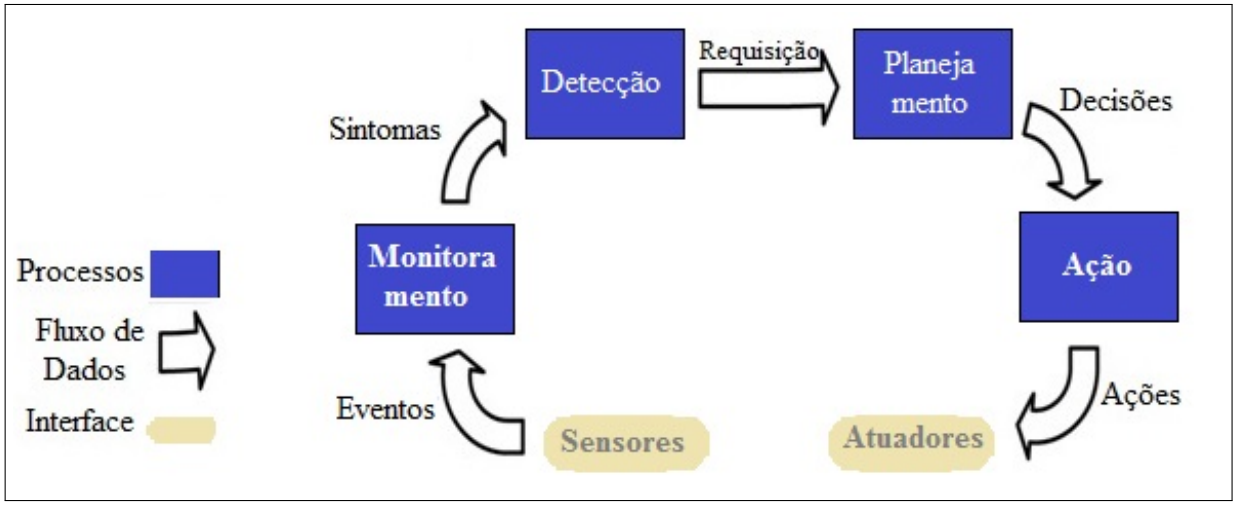
A autoconfiguração dos sistemas impacta principalmente a manutenibilidade, a funcionalidade, a portabilidade, a usabilidade e a confiabilidade do sistema. Um sistema com a propriedade da autocura deve garantir a manutenibilidade, a confiabilidade, a disponibilidade e a sobrevivência do sistema. A auto-otimização impacta diretamente a funcionalidade e a eficiência do sistema. Já a autoproteção impacta a funcionalidade e a cofiabilidade do sistema.

2.1.2 Fases da Adaptação

Sistemas autoadaptativos costumam adotar o ciclo de adaptação MAPE (SALEHIE; TAHVILDARI, 2009) para gerir o seu comportamento, como ilustra a Figura 2.

Esse ciclo de adaptação consiste em quatro fases: monitoramento, análise, planejamento e execução. A fase de monitoramento é responsável por coletar e correlacionar dados

Figura 2 – Ciclo de Adaptação



Fonte: (SALEHIE; TAHVILDARI, 2009)

de sensores, convertendo-os em padrões e sintomas comportamentais. Essa fase responde as questões de onde ocorreu a mudança, quando ocorreu a mudança, e o que foi alterado. A fase de análise (ou detecção) é responsável por analisar as respostas da fase anterior e identificar onde e quando deve ocorrer a mudança. A fase de planejamento (ou decisão) determina o que deve ser alterado e como deve ser feita a alteração para alcançar o estado desejável. Por fim, a fase de execução (ou ação) é responsável por aplicar as ações arquitetadas na fase de planejamento.

Os métodos mais populares para trabalhar com as fases de monitoramento e análise se baseiam na construção e verificação de modelos probabilísticos. Grunske e Zhang (2009) propuseram um framework para trabalhar com monitoramento e análise que estende técnicas de verificação de modelos probabilísticos. Já em (CÁMARA; LEMOS, 2012) e (SAMMAPUN et al., 2005), os autores apresentam abordagens para prover monitoramento em tempo de execução.

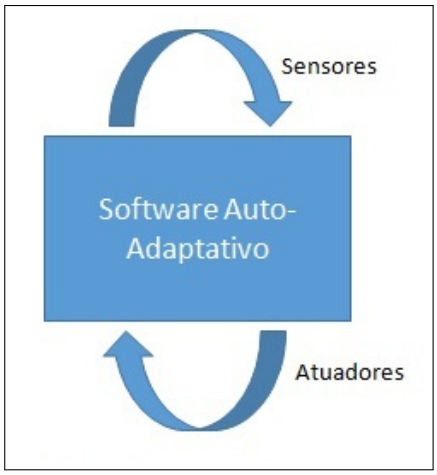
Ainda são poucos os trabalhos que apresentam técnicas para atuar especificamente nas fases de planejamento e execução. Em Salehie e Tahvildari (2009) é apresentada uma lista de possíveis atuadores que poderiam ser utilizados no auxílio destas fases, dentre os quais destaca-se o uso de padrões arquiteturais e de projeto.

2.1.3 Abordagens de Adaptação

Em (SALEHIE; TAHVILDARI, 2009), os autores classificam as abordagens de adaptação como internas ou externas. Na abordagem interna, aplicação e lógica de adaptação se misturam. Nessa abordagem o conjunto de sensores, atuadores e processos de adaptação são

misturados com o código do aplicativo, ocasionando uma má escalabilidade e a dificuldade de manutenção. Essa abordagem pode ser útil para lidar com adaptações locais (por exemplo, para o tratamento de exceção). No entanto, a adaptação muitas vezes precisa de informação global sobre a aplicação e sobre o ambiente. A Figura 3 ilustra a abordagem interna.

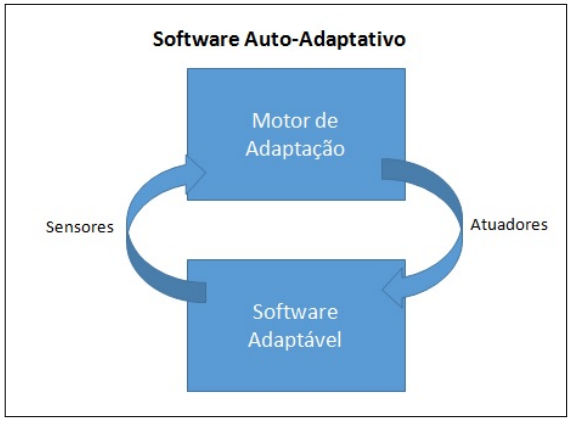
Figura 3 – Abordagem Interna



Fonte: (SALEHIE; TAHVILDARI, 2009)

Na abordagem externa é utilizado um motor externo de adaptação (ou gerenciador autonômico) que contém o processo de adaptação. Utilizando essa abordagem, o sistema de software autoadaptativo consiste em um adaptador e um software adaptável. Uma vantagem significativa da abordagem externa é a reutilização do adaptador, ou de alguns processos realizados, para várias aplicações. Isso significa que um adaptador pode ser personalizado e configurado para sistemas diferentes. A Figura 4 ilustra a abordagem externa.

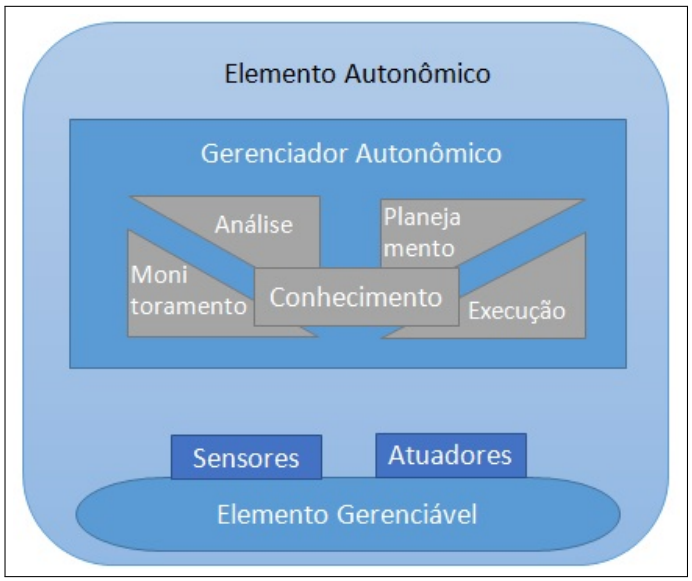
Figura 4 – Abordagem Externa



Fonte: (SALEHIE; TAHVILDARI, 2009)

O modelo da referência da IBM trabalha com um componente de adaptação chamado gerenciador autonômico (HUEBSCHER; MCCANN, 2008), como mostrado na Figura 5. Esse componente coleta e analisa dados, organiza as ações que devem ser tomadas em prol de alcançar os objetivos e controla a execução destas ações.

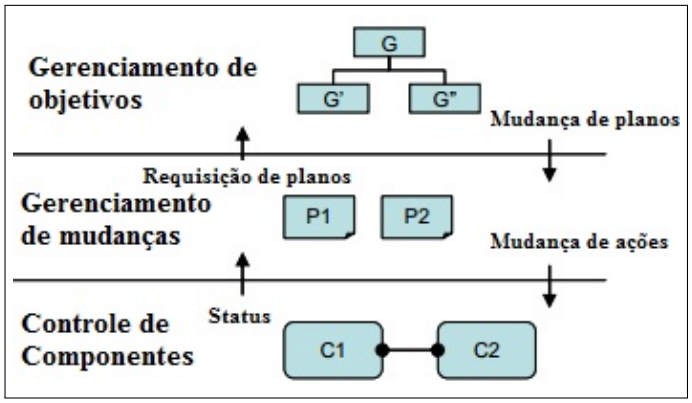
Figura 5 – Modelo de referência da IBM



Fonte: (HUEBSCHER; MCCANN, 2008)

Em Kramer e Magee (2007), os autores também defenderam o uso de uma arquitetura com o componente externo. Os autores dividiram esse componente em três camadas, como ilustra a Figura 6.

Figura 6 – As três camadas do componente externo proposto por Kramer e Magee (2007)



Fonte: (KRAMER; MAGEE, 2007)

A camada de controle de componentes é responsável por controlar os sensores e atuadores, além das adições e remoções de componentes. A camada de gerenciamento de

mudanças é responsável pela análise dos dados obtidos pela camada de controle e pela seleção de planos a serem adotados afim de alcançar a adaptação. A camada de gerencimento de objetivos contém os objetivos que se deseja alcançar, que são utilizados para auxiliar a criação de novos planos.

2.2 SOFTWARES BASEADOS EM SERVIÇOS

Sistemas baseados em serviços são aplicações de software construídas a partir de serviços de baixo acoplamento, ou seja, que independem de outros serviços ou aplicações, de vários fornecedores. Esses sistemas são utilizados em vários domínios de aplicação, incluindo *e-commerce*, *online banking*, e *health care*. Os softwares baseados em serviços operam em ambientes caracterizados por mudanças frequentes (CALINESCU et al., 2012).

Os softwares baseados em serviços seguem uma lógica similar a dos softwares baseados em componentes. Em geral, um software baseado em componentes é análogo a um jogo de montar onde cada bloco é um componente e que juntos geram a estrutura ou aplicação final. Porém, a retirada de um único componente ou adição de outros não necessariamente pode gerar um colapso na estrutura. Portanto, é comum que para essas aplicações haja uma estrutura principal, ou base (que pode ser uma classe principal), e a ela sejam adicionados os demais componentes, de modo que eles agregam novas funcionalidades ao software, mas não são fundamentais para seu funcionamento.

Segundo Perini (2012), é natural que aplicações baseadas em serviços sejam sistemas distribuídos, sem controle direto de seus desenvolvedores, pois serviços podem ser usados, mas são de propriedade de terceiros. Além disso, esses sistemas atuam em ambientes abertos, onde eventos externos podem criar situações imprevistas, e dinâmicas, já que novos serviços podem ser descobertos e selecionados em tempo de execução.

Em Francisco (2003) um serviço é definido como toda e qualquer fonte de informação disponibilizada através da Internet. Essas fontes podem ser ligadas diretamente a componentes de software (por exemplo, sistemas de gerência de bancos de dados) ou a tarefas realizadas por pessoas intermediadas por algum mecanismo de comunicação (por exemplo, uma reserva sendo feita através de e-mail em um hotel). Além disso, vale a pena ressaltar que a definição de aplicação baseada em serviços é recursiva e, portanto, um serviço pode também ser uma aplicação baseada em serviços.

Uma aplicação baseada em serviços tem como ponto-chave a escolha de uma tec-

nologia de componentes que permita o desacoplamento e a distribuição das partes através de uma rede. Uma vez escolhida essa tecnologia, cabe ao projetista da aplicação determinar qual a melhor maneira de realizar a composição dos dados e implementar as funcionalidades desses componentes de software (FRANCISCO, 2003). A principal tecnologia de componentes para representação de serviços são os *web services*.

Francisco (2003) também conclui que o desenvolvimento de aplicativos baseados em serviços permite a criação de software sem que os desenvolvedores saibam quais serviços irão utilizar, sua localização e sua implementação. Isso permite que componentes sejam dinamicamente selecionados em tempo de execução. Essa propriedade garante maior adaptabilidade, distribuição e tolerância a falhas por parte das aplicações.

2.2.1 Web services

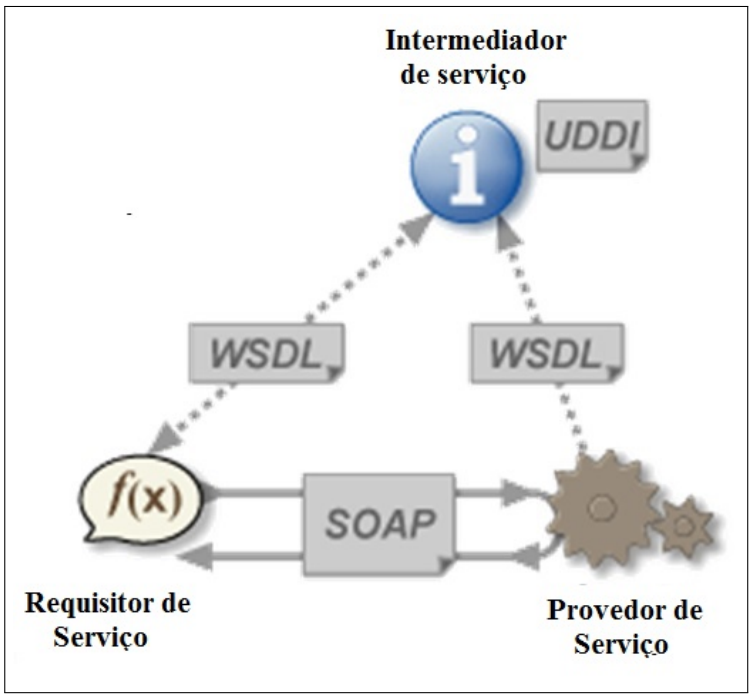
Os *web services* são aplicações autodescritivas, que utilizam o padrão XML, são acessíveis através de uma URL (*Uniform Resource Locator*), independentes das plataformas de desenvolvimento e que permitem a interação entre aplicações sem intervenção humana (LOPES; RAMALHO, 2004).

Segundo Francisco (2003), um *web service* é composto de três tipos básicos de entidades: provedor do serviço, intermediador do serviço e requisitante do serviço. Um determinado serviço é fornecido por um provedor do serviço, que publica suas funcionalidades em um sistema de busca de serviços e recebe o nome de intermediador do serviço.

Um requisitante do serviço é uma entidade que irá utilizar alguma funcionalidade de um *web service* para realizar alguma tarefa. Para tal, essa entidade entra em contato com um intermediador do serviço através de um padrão de descoberta, como por exemplo, o padrão *Universal Description, Descovering and Integration* (UDDI). Ao receber a resposta, o requisitante do serviço terá as interfaces para acesso aos serviços que atendem às suas necessidades. Estas interfaces fazem parte da descrição do serviço, que normalmente é feita utilizando a linguagem WSDL (*Web Service Description Language*). A partir desse ponto, os serviços fornecidos pelo provedor do serviço poderão ser acessados utilizando um protocolo de acesso a objetos remotos (por exemplo, o *Simple Object Access Protocol* - SOAP) (FRANCISCO, 2003). A Figura 7 apresenta um esquema básico da arquitetura de um *web service*.

Em Lopes e Ramalho (2004) é descrito de maneira simplificada o ciclo de vida de um *web service*. Esse ciclo segue os seguintes passos:

Figura 7 – Esquema básico da arquitetura de um *web service*



Fonte: Obtido em http://pt.slideshare.net/joaosavio/web-services-com-netbeans

- O fornecedor constrói o serviço utilizando uma linguagem de programação qualquer;
- Em seguida, especifica a interface/assinatura do serviço que definiu em WSDL;
- O fornecedor registra o serviço no *Service Broker* usando o padrão UDDI;
- O utilizador (aplicação cliente) pesquisa em um repositório UDDI e encontra o serviço;
- A aplicação cliente estabelece a ligação com o *web service* e estabelece um diálogo com este, via mensagens SOAP.

2.2.2 Softwares AutoAdaptativos Baseados em Serviços

Softwares autoadaptativos baseados em serviços podem ser compreendidos como softwares baseados em serviços capazes de se adaptar de forma automática, autônoma e dinâmica às mudanças no ambiente e/ou seus requisitos.

Segundo Nitto et al. (2008), tais alterações podem incluir a implantação de novas instâncias de um determinado tipo de serviço, a remoção de serviços já existentes, e até mesmos mudanças globais na aplicação. Os softwares autoadaptativos baseados em serviços devem ser capazes de continuar em funcionamento, mesmo quando ocorrem mudanças no comportamento

dos serviços que eles utilizam, sendo esses serviços controlados internamente ou fornecidos por provedores externos de serviços. A auto adaptação requer que softwares autoadaptativos baseados em serviços sejam capazes de automaticamente descobrir novos serviços, os escolhendo dentre os provedores de serviços disponíveis.

Segundo Calinescu et al. (2012), uma maneira de criar sistemas autoadaptativos baseados em serviços consiste em selecionar dinamicamente os serviços que implementam suas operações a partir de um conjuntos de serviços funcionalmente equivalentes associados a diferentes níveis de desempenho, confiabilidade e custo.

3 TRABALHOS RELACIONADOS

Existem diversos trabalhos que discutem e apresentam técnicas para solucionar os desafios relacionados ao desenvolvimento de softwares autoadaptativos. Outros trabalhos apresentam propostas para implementação desse tipo de software. Muitos desses focam em apresentar alternativas apenas para algumas etapas do ciclo de adaptação, destacando-se as etapas de Monitoramento e Análise. Há trabalhos que propuseram arquiteturas completas para desenvolvimento de softwares autoadaptativos, alguns desses também apresentam frameworks e outras bases para codificação dessas arquiteturas. Há trabalhos que apresentam propostas de abordagens para o desenvolvimento de softwares autoadaptativos baseados em serviços. Os trabalhos relacionados presentes nesse capítulo apresenta propostas a algumas das questões anteriormente expostas.

3.1 QUESTÕES SOBRE DESENVOLVIMENTO E FASES DE ADAPTAÇÃO

Salehie e Tahvildari (2009) apresentam uma série de desafios relacionados ao desenvolvimento de softwares autoadaptativos em geral. Dentre esses desafios, o mais correlacionado a este trabalho sugere que ainda há oportunidades para mais pesquisa na área de desenvolvimento de novos frameworks, ferramentas e linguagens adequadas a criação de softwares autoadaptativos.

Gorla et al. (2010) discutem os desafios para criação de sistemas com a propriedade de autocura focando em falhas funcionais. Os autores também apresentam técnicas para construção de softwares que podem curar automaticamente tais falhas. Esse trabalho abrange todas as etapas do ciclo de adaptação.

Calinescu et al. (2012) apresentam uma base para verificação quantitativa em tempo de execução para as etapas de monitoramento, análise e planejamento de softwares autoadaptativos usando cadeias de Markov de tempo discreto e *Probabilistic Computation Tree Logic* (PCTL).

Neste trabalho é apresentado um framework para auxiliar o desenvolvimento de softwares autoadaptativos, sem foco em nenhuma propriedade auto-* específica, diferentemente do trabalho de Gorla et al. (2010), mas este framework é genérico o bastante para possibilitar a utilização de quaisquer uma das propriedade auto-* e que também abrange as etapas do ciclo de adaptação. Além disso, o framework não define uma forma específica de trabalhar com cada

etapa do ciclo de adaptação, mas é utilizada a proposta de explorar o uso de serviços para a etapa de planejamento exposta em (CALINESCU et al., 2012) como uma das bases para nosso framework.

3.2 ARQUITETURAS DE DESENVOLVIMENTO DE SOFTWARES AUTOADPTATIVOS

Kramer e Magee (2007) apresentam uma base para a composição de arquiteturas de desenvolvimento de softwares autoadaptativos utilizando um componente externo responsável pela adaptação, que é dividido em três camadas que gerenciam objetivos, o ciclo de adaptação e os componentes de controle.

Garlan et al. (2004) apresentam Rainbow, uma arquitetura e framework, que suporta a autoadaptação de sistemas de software utilizando um mecanismo externo que gera a especificação de estratégias de adaptação para múltiplos sistemas. Esse trabalho é uma das principais referências base para o estudo de softwares autoadaptativos.

Affonso e Nakagawa (2013) apresentam uma arquitetura de referência que se utiliza do mecanismo de reflexão para prover a adaptação de software. Esse metamodelo pode ser alterado e a partir desse novo metamodelo pode-se gerar um novo código provendo assim a adaptação do software em tempo de execução.

Affonso et al. (2014) apresentam um estudo comparativo entre arquiteturas de referência para sistemas de softwares autogerenciáveis. A principal contribuição desse estudo foi detalhar o estado da arte das arquiteturas de referência para sistemas autogerenciáveis. A técnica de Revisão Sistemática da Literatura foi adotado para a coleta de dados e extração de informações de bancos de dados de trabalhos publicados.

Oliveira e Barbosa (2014) apresentam uma estratégia de autoadaptação para arquiteturas baseadas em serviços que utiliza um modelo para estratégias de reconfiguração da arquitetura, planejadas em tempo de projeto. Esse modelo contém um sistema de transição cujos estados são possíveis configurações originalmente previstas para arquitetura, e as arestas representam as reconfigurações, ou seja, os caminhos de uma configuração para outra.

O framework SASeS também utiliza uma arquitetura com um componente de adaptação externo, tal como proposto por Kramer e Magee (2007) e leva em consideração algumas das questões propostas nesse artigo. Para este trabalho foi escohido não utilizar metamodelos nem reflexão, como em (AFFONSO; NAKAGAWA, 2013), por crermos que a utilização de serviços, também utilizados em (OLIVEIRA; BARBOSA, 2014) é uma maneira robusta e mais

simples de prover um nova adaptação em tempo de execução. Por fim, o trabalho de Affonso et al. (2014) foi estudado como referência sobre arquiteturas de sistemas autoadaptativos.

3.3 FRAMEWORKS E OUTRAS ABORDAGENS PARA O DESENVOLVIMENTO DE SOFTWARES AUTOADAPTATIVOS E PARA SOFTWARES AUTOADAPTATIVOS BASEADOS EM SERVIÇOS

Inverardi et al. (2004) apresentam um framework para trabalhar com sistemas pervasivos que promove uma adaptação da aplicação para que essa possa trabalhar com dispositivos diferentes. A aplicação desenvolvida com esse framework deve verificar as incompatibilidades que não permitem que certos dispositivos possam utilizar a aplicação ou serem utilizados por essa e adapta aplicação para que esta se torne compatível ao dispositivo. O framework usa uma abordagem declarativa desenvolvida em Java que utiliza metodologias conhecidas. Essas metodologias usam fórmulas lógicas para poder descrever a incompatibilidade e tornar possível a adaptação.

Gorton et al. (2006) apresentam o *Adaptive Server Framework* (ASF). Esse framework é usado para construir servidores de aplicação com comportamento adaptativo. O ASF prover uma separação clara entre implementação de comportamentos adaptativos e o servidor de aplicação da lógica de negócio. Além disso, o ASF é uma arquitetura leve em que incorre baixa sobrecarga de CPU (*Central Processing Unit*) e uso de memória.

Adamczyk et al. (2008) apresentam uma arquitetura de framework chamada *Autonomic Management Toolkit* (AMT), que emprega mecanismos de regras para o raciocínio e tomada de decisão. Este conjunto de ferramentas foi implementado para suportar desenvolvimento dinâmico e gerenciamento de loops de adaptação. Isso requer a descoberta automática de recursos, instrumentação e acessórios para gerenciamento autonômico, além disso há um módulo de tomada de decisão escalável e de fácil alteração, que é a parte principal do gerenciamento autonômico.

Asadollahi et al. (2009) propuseram um framework genérico desenvolvido em Java para criação de sistemas autogerenciáveis chamado StarMX. Ele suporta diversas propriedades auto-* e utiliza diversos padrões de desenvolvimento.

Autili et al. (2010) propuseram um modelo que visa auxiliar o desenvolvimento de aplicações Java em ambientes pervasivos. Esse modelo estende a linguagem Java com base em um framework chamado CHAMALEON e apresenta um código genérico que é usado como base

para a adaptação em conjunto com classes e métodos adaptáveis.

Yang et al. (2010) propuseram um framework para construção de softwares autoadaptativos que utiliza lógica fuzzy. O framework apresenta um controle fuzzy que utiliza informação heurística para a construção de estratégias de controle e controladores da adaptação, que são tolerantes a dados imprecisos.

Burg e Dolstra (2011) desenvolveram um framework de desenvolvimento autoadaptativo para suportar a construção de sistemas orientados as serviços desenvolvidos com base em uma ferramenta chamada Disnix. Esse framework descobre dinamicamente máquinas na rede e cria um mapa de componentes para as máquinas baseado em propriedades não funcionais. O Disnix é então invocado para automaticamente, confiavelmente e eficientemente reimplantar o sistema.

Derakhshanmanesh et al. (2011) elaboraram o GRAF, um framework para adaptação de sistemas em tempo de execução baseado na notação de grafos, que permite adaptação através da criação, gerenciamento e interpretação de modelos de software baseados em tal notação. Esse framework apresenta um elemento externo ao software adaptável, que utiliza o ciclo MAPE e que é dividido em três camadas, a primeira contém um *middleware* que gerencia os sensores e atuadores, a segunda é responsável pela gerencia dos modelos e a terceira é responsável por gerenciar a adaptação.

O framework FUSION, proposto por Elkhodary et al. (2010), visa resolver o problema da previsão das mudanças do ambiente, o que facilita a adaptação em tempo de execução para sistemas baseados em funcionalidades utilizando técnicas de aprendizagem de máquina.

Calinescu e Rafiq (2013) propuseram a utilização de *proxies* inteligentes para o desenvolvimento de sistemas autoadaptativos baseados em serviços. Em (CALINESCU et al., 2013), os mesmos autores utilizam esses proxies na composição de um framework para desenvolvimento de softwares autoadaptativos baseados em serviços com a propriedade de autoverificação.

Abufouda (2014) propôs um framework para prover a autoadaptabilidade em sistemas de software utilizando uma abordagem externa. Essa abordagem utiliza raciocínio baseado em casos (AAMODT; PLAZA, 1994) no componente responsável pela adaptação. A abordagem também tenta gerenciar a incerteza referente a mudanças no ambiente e internas ao software adaptável utilizando terias probabilísticas e função de utilidade.

O framework SASeS possui uma abordagem genérica simples que auxilia a criação

de softwares autoadaptativos baseados em serviços. O SASeS foca na criação de um gerenciador externo (Módulo Adaptador) responsável pelo gerenciamento do ciclo de adaptação que ao fim de sua execução seleciona as funcionalidades (serviços) utilizadas pelo software baseado em serviço. Além disso, o framework também apresenta um modelo para criação da aplicação adaptável. Os serviços selecionados pelo Módulo Adaptador não precisam ser pré-definidos, assim é possível remover e adicionar novos serviços. Na fase atual o SASeS provê a opção de se utilizar arquivos XML ou *sockets* para configuração e troca de mensagens entre o Módulo Adaptador e a Aplicação.

Os trabalhos Calinescu e Rafiq (2013) e (CALINESCU et al., 2013) foram utilizados como base para construção da abordagem que evolui e se torno o framework proposto neste trabalho. Parte formatação do trabalho (ASADOLLAHI et al., 2009) também foi adaptado para construção das seções deste trabalho e mais especificamente de artigos provenientes desta pesquisa.

Grande parte dos demais frameworks apresentados nessa seção foram estudados e utilizados como comparação ao SASeS, mas como não foi possível obter o código fonte desses frameworks, não foi feita uma análise mais profunda, que seria importante para avaliar tanto as vantagens de se utilizar o SASeS em detrimento desses trabalhos, quanto para avaliar o que seria interessante aproveitar desses trabalhos para aplicar no SASeS.

A Tabela 3 apresenta um sumário dos trabalhos relacionados nessa subseção e inclui também este trabalho. A tabela tem dois aspectos de comparação (1) os aspectos de pesquisa e (2) aspectos relacionados a softwares autoadaptativos e softwares autoadaptativos baseados em serviços, similar ao proposto em (ABUFOUDA, 2014). Os aspectos de pesquisa representam indicações sobre a maturidade da pesquisa e indica se o trabalho contém as seguintes informações: (i) o trabalho estabelece o problema a ser resolvido; (ii) o trabalho propôs um framework; (iii) o trabalho apresenta estudo de caso; (iv) o trabalho apresenta uma autoavaliação do framework ou abordagem proposta; (v) é disponibilizado uma maneira para que o leitor obtenha o código fonte do framework ou abordagem apresentada; e (vi) são apresentadas limitações do trabalho.

Os aspectos relacionados a softwares autoadaptativos e softwares autoadaptativos baseados em serviços estão relacionado ao tópico de pesquisa deste trabalho e indica se o trabalho contém as seguintes informações: (i) o trabalho apresenta como é feita a adaptação utilizando o framework ou abordagem proposta; (ii) o framework ou abordagem proposta possui um componente responsável pelo ciclo de adaptação; (iii) o framework ou abordagem proposta

trata incertezas do ambiente; (iv) o framework ou abordagem proposta trata de sistemas baseados em serviços; (v) qual o estilo de adaptação utilizada pelo framework ou abordagem proposta; e (vi) qual a abordagem de adaptação utilizada pelo framework ou abordagem proposta.

Tabela 3 – Sumário de trabalhos relacionados

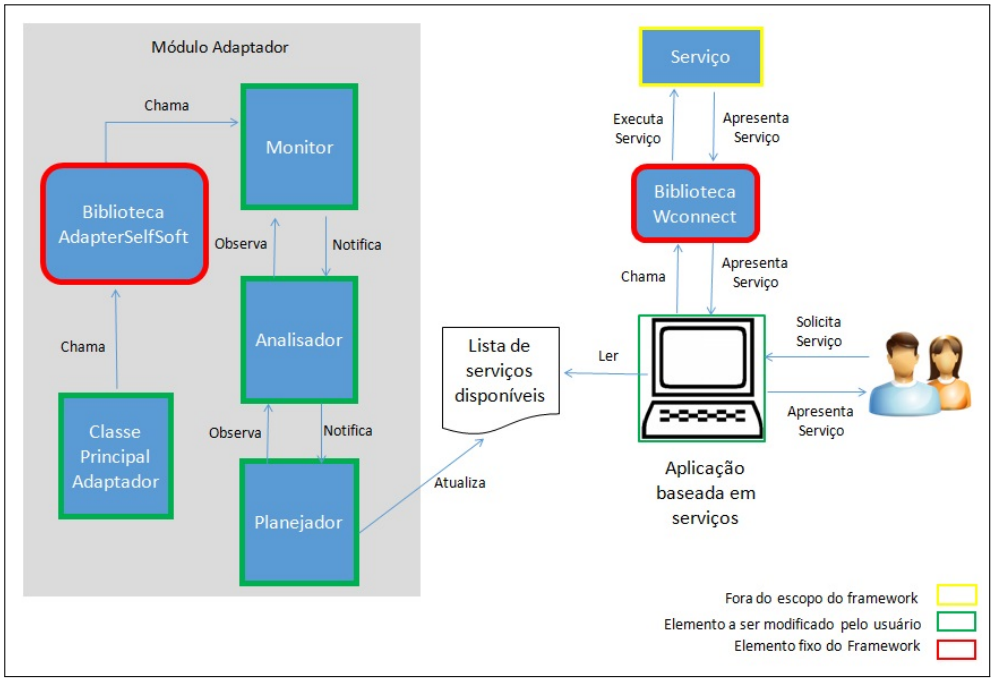
Trabalhos	Aspectos de Pesquisa						Aspectos de Sistemas de Softwares Auto-Adaptativos e Baseados em Serviços					
	Estabelece o problema	Framework	Estudo de Caso	Auto Avaliação	Disponibilidade de Código Fonte	Limitações	Apresenta a Adaptação	Ciclo de Adaptação	Incertezas do Ambiente	Sistemas Baseados 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	X	Dinâmica	Externa
AUTILI et al., 2010	√	X	√	√	X	X	√	√	X	X	Dinâmica	Externa
YANG et al., 2010	√	√	X	√	X	X	√	X	√	X	Dinâmica	Externa
BURG e DOLSTRA, 2011	√	√	√	√	X	X	√	√	X	√	Dinâmica	Externa
DERAKHSHANMANESH et al., 2011	√	√	√	√	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
ABUFOUDA, 2014	√	√	X	√	X	√	√	√	√	X	Dinâmica	Externa
Framework SASeS	√	√	√	√	√	√	√	√	X	√	Dinâmica	Externa

4 FRAMEWORK SASeS

O framework SASeS foi criado para auxiliar o desenvolvimento de softwares auto adaptativos baseados em serviços, que consistem em aplicações cujas funcionalidades são contidas em serviços (*web services*) e que pode se autoadaptar. O framework SASeS permite a criação de um (ou mais) componente de adaptação externo (módulo adaptador) que seleciona e informa a lista de serviços que serão utilizados pela aplicação. Uma vez selecionados os serviços, esse componente ainda pode, em tempo de execução, alterar quais são esses serviços, retirando ou adicionando novos serviços dessa lista, de acordo com os requisitos do software. O framework ainda auxilia a criação da aplicação baseada em serviços[1].

A Figura 8 mostra uma visão geral do framework, que é composto pelo módulo Adaptador, pelo módulo de aplicação, e pela biblioteca WConnect, que provê classes que generalizam *web services* e que permitem a conexão das classes dos módulos com os *web services*. O módulo Adaptador, a Aplicação e a biblioteca são detalhados a seguir.

Figura 8 – Visão geral do framework SASeS



Fonte: Elaborado pelo autor

Na Figura 8, as bibliotecas estão circundadas por uma borda vermelha, indicando que são elementos internos do framework que não necessitam devem ser modificados. Os elemenos circundados de verde são elementos que deverão ser alterados pelo usuário ao desenvolver

[1] O código da versão atual do framework está disponível em: http://goo.gl/r3igB5

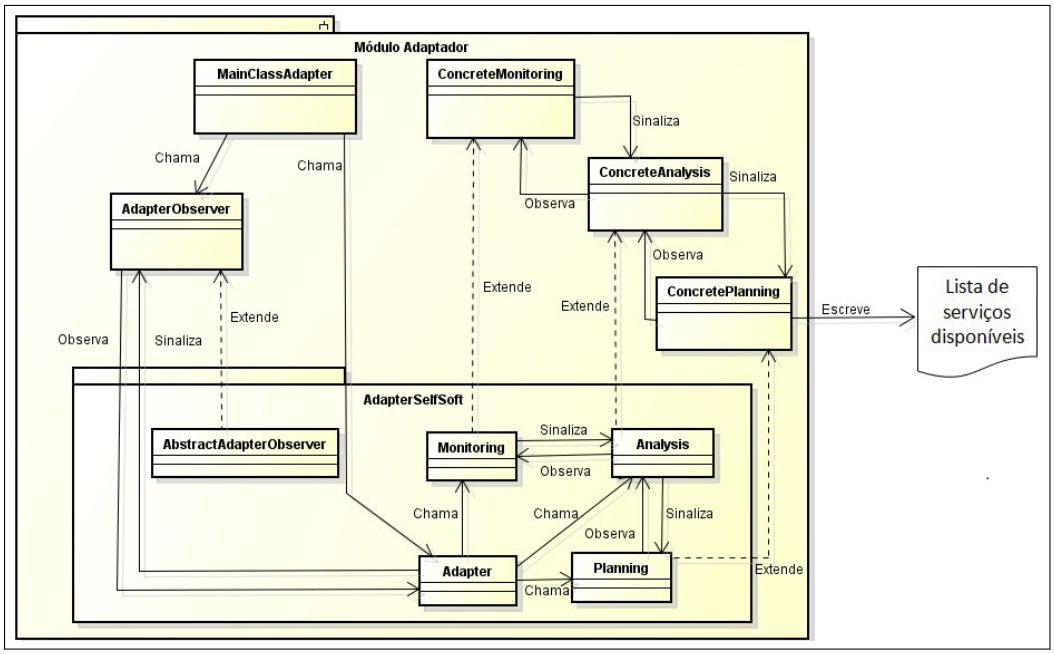
o sistema baseada em serviços. Os serviços estão circundados com a cor amarela, porque o desenvolvimento desses não faz parte do escopo do framework.

Importante destacar que o módulo adaptador não apresenta um componente específico para a etapa de Execução do ciclo MAPE, pois foi observado que dentro da lógica do framework não haveria necessidade de mais um componente de modo que o próprio componente relacionado ao Planejamento atualiza a lista de serviços. Além disso, esta lista pode ser um arquivo XML ou pode ser informada diretamente a aplicação via *sockets*. Para trabalhar com os arquivos XML no framework SASeS foi utilizada a biblioteca *XStream* (WONG et al., 2003).

4.1 O MÓDULO ADAPTADOR

Este módulo provê as classes necessárias para criação do componente de adaptação externo. Ele é composto pela biblioteca AdapterSelfSoft, que consiste de classes abstratas que representam os três primeiros componentes do ciclo de adaptação (Monitoramento, Análise e Planejamento), e a classe Adapter, que de fato é o gerenciador da adaptação, que usa os objetos do ciclo de adaptação mencionados anteriormente. Além disso, um exemplo concreto genérico das classes de Monitoramento, Análise e Planejamento é também fornecido. Deste modo o usuário pode apenas modificar estas classes e seus comportamentos específicos. A Figura 9 apresenta o Diagrama de classes do Módulo Adaptador.

Figura 9 – Diagrama de classes do Módulo Adaptador



Fonte: Elaborado pelo autor

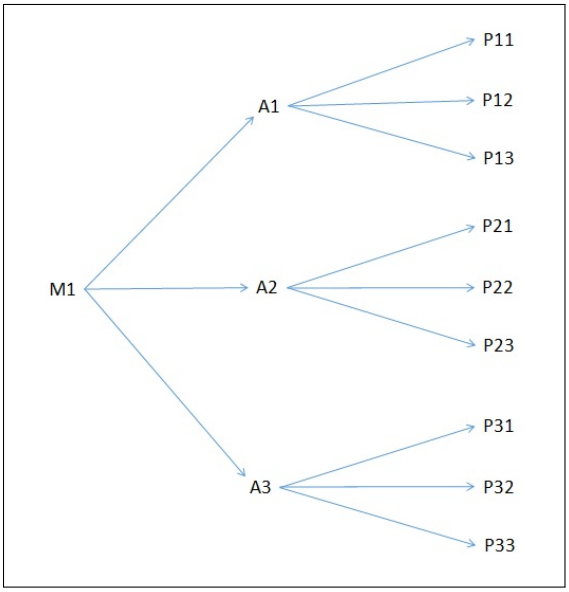
O SASeS assume que a comunicação entre as classes do ciclo de adaptação é realizada utilizando, por padrão, arquivos. Isto significa que, quando o monitor concreto é executado, o resultado de sua tarefa de monitoramento é escrito em um arquivo de monitoramento. O analisador concreto lê o arquivo de monitoramento e começa a tarefa de análise, mas o resultado é escrito em outro arquivo, que é lido pelo planejador concreto. Outra possibilidade para a comunicação, tanto entre classes do ciclo de adaptação quanto entre o módulo adaptador e a aplicação, seria a utilização de *sockets* em vez de arquivos.

Para sincronizar a execução entre as classes concretas, o padrão de projeto *Observer* (GAMMA et al., 1994) foi implementado. Assim, quando a saída de uma classe apresenta um resultado de sua tarefa específica (monitoramento, análise ou planejamento), ela notifica a próxima classe que precisa desse resultado para executar. Diferentemente do StarMX (ASADOLLAHI et al., 2009), que usa um padrão de projeto *Chain of Responsability* (GAMMA et al., 1994) para permitir a comunicação entre os elementos do ciclo de adaptação, optou-se por utilizar o *Observer*, já que ele permite que múltiplos elementos acessem o mesmo recurso.

Para implementar o padrão *Observer*, as classes abstratas de Monitoramento e Análise têm uma lista de observadores de Análise e Planejamento, respectivamente, que são notificados via método NOTIFY, quando o arquivo de leitura é atualizado. A notificação faz os analisadores e planejadores observadores executarem seus métodos UPDATE, que de fato chamam os métodos ANALYZE e PLAN, respectivamente. A lógica desses métodos, juntamente com o método MONITORING da classe de Monitoramento, devem ser implementadas pelo usuário na classe concreta correspondente. Além desses métodos, a classe abstrata tem também métodos para adicionar e remover observadores.

Com a lógica do padrão *Observer*, é possível que um monitor sejam associados vários analisadores e a um analisador sejam associados vários planejadores. A Figura 10 mostra um exemplo da árvore de associações possíveis entre monitores, analisadores e planejadores. A execução segue a regra de uma busca em profundidade, assim ao terminar a execução do monitor (M1) é notificado o primeiro Analisador (A1), que por sua vez notifica o primeiro planejador (P11). Após P11 terminar sua execução, o segundo planejador (P12) será executado e após P12 terminar sua execução, o terceiro planejador (P13) será executado. Ao fim da execução de P13 começa a execução do Analisador 2 (A2), que irá notificar o quarto planejador (P21) e assim por diante. Ao fim da execução de cada planejador pode-se considerar que uma adaptação foi executada.

Figura 10 – Exemplo de associações possíveis entre monitores, analisadores e planejadores



Fonte: Elaborado pelo autor

A última classe na biblioteca AdapterSelfSoft é a Adapter. Essa classe implementa o comportamento concreto do módulo adaptador e tem uma lista de Monitores, Analisadores e Planejadores. Além disso, essa classe tem o método RUN, que inicia a execução dos monitores e, consequentemente, todo o processo do ciclo de adaptação.

A classe Adapter também usa o padrão *Observer* associado a uma classe chamada AdapterObserver que é parte do Módulo Adaptador. A classe AdapterObsever implementa a classe abstrata AbstractAdapterObserver, contida na biblioteca AdapterSelfSoft. Essa associação da classe Adapter com o AdapterObserver é importante porque a última contém a função que determina o critério de parada da execução da classe Adapter que irá ser checado depois de cada ciclo de adaptação.

Finalmente, a classe MainClassAdapter apresentada na Figura 9 é a classe executável que é usada para iniciar o módulo adaptador.

4.2 O MÓDULO DE APLICAÇÃO

O módulo de aplicação é composto por uma classe executável (*main*) e uma classe de apresentação (*view*). A classe principal é, de fato, a classe de execução da aplicação real baseada em serviços, que é composta pelos serviços que fornecem as principais funcionalidades do sistema. Nesta classe existe um laço de execução em que os serviços serão chamados.

A classe View é usada para separar a lógica da aplicação da lógica de apresentação.

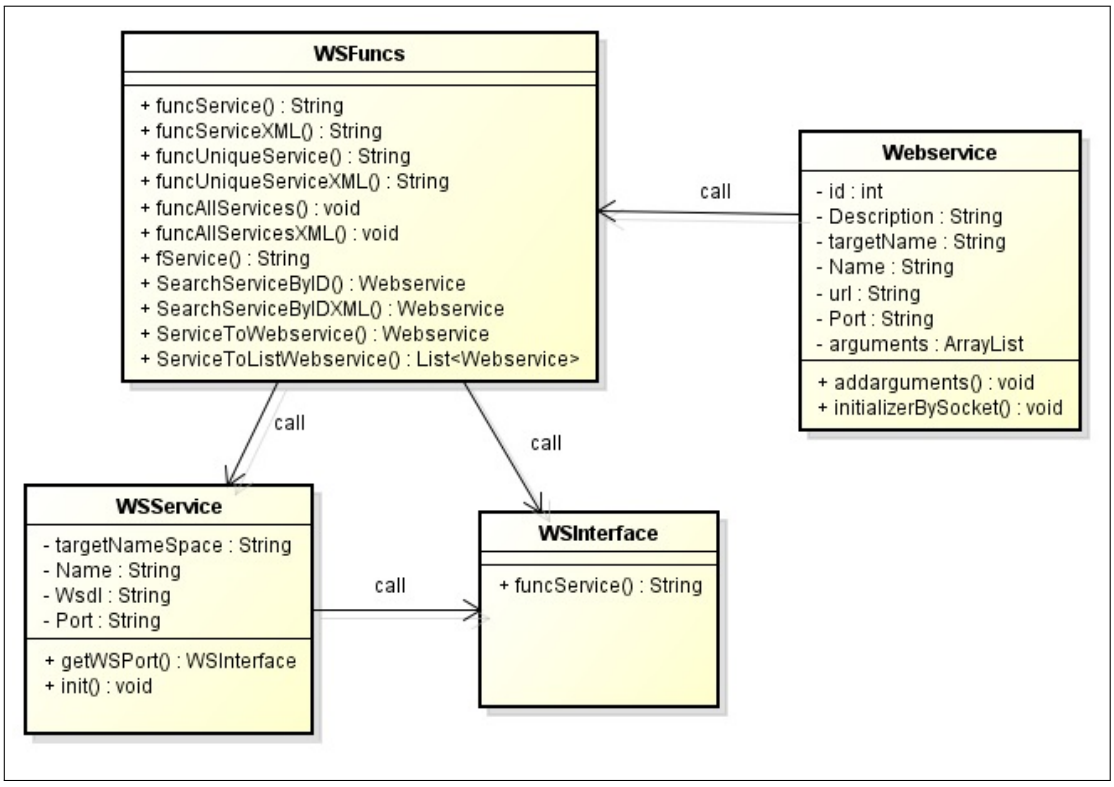
Esta classe é opcional e o usuário pode criar a interface usando outras tecnologias de criação de interfaces gráficas, como o pacote *java swing* ou *JavaFX*.

A fim de tornar a aplicação desacoplada dos serviços, permitindo assim que o serviço seja dinâmico, podendo alterá-lo sem parar a aplicação, a classe principal usa somente os serviços que são informados para essa pela classe responsável pelo planejamento presente no módulo adaptador.

4.3 A BIBLIOTECA WCONNECT

Esta biblioteca é composta por três classes (Webservice, WSServices e WFuncs) e uma interface (WSInterface), como mostra a Figura 11. A classe Webservice representa um *web service*, cujos atributos (`id`, `description`, `targetName`, `Name`, `url`, `Port`, `arguments`) contém as informações necessárias para acessar o serviço. Essa classe tem um método que instancia atributos via *socket*, chamada `initializerBySocket`, que é necessária se o desenvolvedor não quer trabalhar com arquivos em sua implementação.

Figura 11 – Diagrama de classe da biblioteca WConnect



Fonte: Elaborado pelo autor

A classe WSService e a interface WSIterface são classes criadas pelo framework

JAX-WS para permitir a conexão com serviços.

A classe WSFuncs contém um conjunto de métodos que recebem as informações que identificam um ou mais serviços e os atributos que esses precisam para executar, então são usadas as classes WSService e a WSInterface para acessar o serviço. A classe WSFuncs é usada pelas classes que precisam se conectar com um ou mais serviços. Além disso, para os métodos dessa classe permitirem acesso simplificado aos serviços, há outros métodos, como `SearchServiceByID`, `SearchServiceByIDXML`, `ServiceToWebservice` e `ServiceToListWebservice`, que são usados para ler arquivos e preencher com os dados desses arquivos objetos Webservice.

A Figura 12 mostra um exemplo de arquivo que contém os atributos de um objeto Webservice que representa o serviço e que pode ser usado pelos métodos da classe WSFuncs. Este arquivo contém as informações dos atributos da classe Webservice. As linhas 3 e 4 apresentam atributos usados para facilitar a manipulação de arquivos XML, as linhas 5 a 8 apresentam atributos que são parte do arquivo *wsdl* do *web service* e as linhas 9 a 11 apresentam a lista de argumentos que um *web service* específico precisa para executar.

Figura 12 – Exemplo de arquivo XML com informações da lista de serviços

```
1  <list>
2    <webservice.Webservice>
3      <id>1</id>
4      <Description>Service 1</Description>
5      <targetName>http://jr.com/</targetName>
6      <Name>ServiceTwoService</Name>
7      <url>http://localhost:8080/AirPortServices/ServiceTwo?wsdl</url>
8      <Port>ServiceTwoPort</Port>
9      <arguments>
10       <string></string>
11     </arguments>
12   </webservice.Webservice>
13 </list>
```

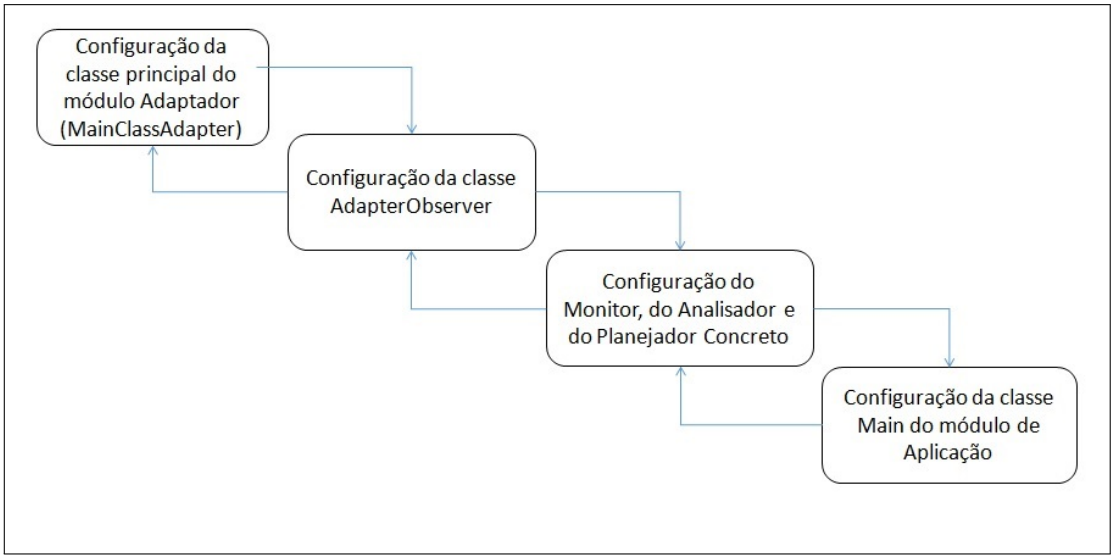
Fonte: Elaborado pelo autor

A classe WSFuncs foi construída para facilitar a conexão aos *web services*, evitando, assim, a obrigação do usuário saber trabalhar com o framework JAX-WS. Entretanto, é possível usar o framework JAX-WS, se o usuário preferir, mas é necessário fazer customizações para que não haja conflito quando usá-lo com o framework SASeS.

4.4 PASSO A PASSO DE COMO UTILIZAR O SASES PARA CRIAR UMA SOFTWARE AUTOADAPTATIVO BASEADO EM SERVIÇOS

Nessa subseção são apresentadas as etapas para o desenvolvimento das classes do framework SASeS que devem ser criadas pelo desenvolvedor. Na versão atual do framework consta uma versão genérica de cada uma dessas classes. A Figura 13 ilustra as etapas a serem seguidas pelo desenvolvedor.

Figura 13 – Etapas a serem seguidas pelo desenvolvedor quando for utilizar o SASeS



Fonte: Elaborado pelo autor

Nas três primeiras etapas são configuradas as classes do módulo Adaptador e na quarta etapa é configurada a classe principal do módulo de aplicação. Os códigos de exemplos dessas classes são apresentados e descritos a seguir.

Será omitida a lógica da classe *view* do módulo de aplicação, pois, como foi citado na subseção 4.2, é possível criar uma interface gráfica com outras tecnologias, mas uma versão genérica dessa classe também consta no pacote da versão atual do framework. É importante informar que a versão atual do framework foi desenvolvido utilizando a IDE Netbeans e alguns instruções podem necessitar de uma adaptação caso utilize outra IDE.

É importante destacar que ao obter o pacote da versão atual do framework SASeS, antes de começar a modificar os projetos genéricos contidos nele, é importante adicionar as bibliotecas WConnect, AdapterSelfSoft e a XStream.

Existem duas maneiras de implementar o módulo Adaptador, incluindo a lógica de cada fase de adaptação nas classes concretas de monitoramento, análise e planejamento ou

adicionando as lógicas a *web services*, sendo a função de cada classe concreta apenas executar esses serviços.

A Figura 14 apresenta um exemplo de configuração da classe MainClassAdapter, no qual a lógica de cada fase de adaptação constará nas classes concretas de monitoramento, análise e planejamento.

Figura 14 – Exemplo de configuração da classe MainClassAdapter

```
13          AdapterObserver sf = new AdapterObserver();
14
15          Monitoring monitor = new ConcreteMonitoring();
16          Analysis analyzer = new ConcreteAnalysis();
17          Planning planner = new ConcretePlanning();
18          Monitoring monitor2 = new ConcreteMonitoring();
19          Analysis analyzer2 = new ConcreteAnalysis();
20          Planning planner2 = new ConcretePlanning();
21
22          List<Monitoring> monitors = new ArrayList<Monitoring>();
23          List<Analysis> analyzers = new ArrayList<Analysis>();
24          List<Planning> planners = new ArrayList<Planning>();
25          monitors.add(monitor); monitors.add(monitor2);
26          analyzers.add(analyzer); analyzers.add(analyzer2);
27          planners.add(planner); planners.add(planner2);
28
29          monitor.setValuesMonitor("ValuesMonitor.xml");
30          analyzer.setValuesAnalysis("ValuesAnalysis.xml");
31          monitor2.setValuesMonitor("ValuesMonitora.xml");
32          analyzer2.setValuesAnalysis("ValuesAnalysisa.xml");
33          monitor.setTime(0);
34          analyzer.setObserver(monitor); analyzer.setObserver(monitor2);
35          planner.setObserver(analyzer); planner.setObserver(analyzer2);
36
37          Adapter adapter = new Adapter(monitors, analyzers, planners);
38
39          sf.setObserver(adapter);
40
41          adapter.run();
```

Fonte: Elaborado pelo autor

Na linha 13 é instanciado um objeto do tipo AdapterObserver. Nas linhas 15 a 20 são instanciados os monitores, analisadores e planejadores concretos. Nas linha 22 a 27 são criadas as listas de objetos de monitoramento, análise e planejamento e em seguida os monitores, analisadores e planejadores concretos são adicionados a estas listas. Nas linhas 29 a 32 são declarados os nomes dos arquivos de saída dos monitores e analisadores. Na linha 33 é especificado o tempo entre a execução dos monitores (isto é opcional). Nas linhas 34 e 35 são setados os observadores. Na linha 37 é instanciado um objeto do tipo Adapter e na linha 39 é setado o padrão *Observer* para o objeto do tipo AdapterObserver. Finalmente, na linha 41 é executado o método `run` do objeto do tipo Adapter que inicializa a adaptação.

A diferença para o caso onde são utilizados *web services* é apenas a necessidade de instanciar também objetos do tipo Webservice ou lista de objetos deste tipo. Neste caso, também apenas haverá um monitor, um analisador e um planejador concreto, que será responsável por chamar os serviços e mesmo utilizando uma lista de serviços, como a lógica dessas classes concretas não muda, não há necessidade de utilizar mais de uma dessas classes.

A Figura 15 apresenta o código da classe AdapterObserver que contém a lógica de parada do adaptador e pode conter outras funções específicas desse adaptador. Na linha 27 é declarado um objeto do tipo Adapter. Na linha 29 é declarado o construtor da classe. Nas linhas 31 a 34 é declarada a função que inicializa o *Observer*. Nas linhas 37 a 41 consta a função que é disparada pelo Adapter. Nas linhas 43 a 46 é declara a função que contém a lógica do critério de parada do adaptador.

Figura 15 – Exemplo de configuração da classe AdapterObserver

```
26  public class AdapterObserver implements AbstractAdapterObserver{
27      private Adapter adapterObserved;
28
29      public AdapterObserver(){}
30
31      public void setObserver(Adapter a){
32          this.adapterObserved = a;
33          adapterObserved.addAdapterObserver(this);
34      }
35
36      @Override
37      public void update(Adapter a) {
38                  if(a == adapterObserved){
39                      adapterObserved.setStop(verify());
40                  }
41      }
42      public Boolean verify(){
43          Boolean baux = true;
44          return baux;
45      }
46  }
```

Fonte: Elaborado pelo autor

A Figura 16 apresenta um exemplo de configuração do monitor concreto. Nas linhas 16 a 18 é declarado um construtor para o caso em que são utilizados *web services*, ou seja, quando a lógica de monitoramento está contida em um ou mais webservices. Caso não se utilize webservices a lógica de monitoramento deve ser adicionada ao método `monitoring` do monitor concreto. Nas linhas 20 e 21 é declarado um construtor vazio. As linhas 25 a 35 apresentam

a função que contém a lógica de monitoramento. Da linha 25 até à linha anterior a linha em que é executado o método `Notify` deve constar a lógica de monitoramento. Se a lógica de monitoramento estiver contida em um serviço basta executar o método `monitoringWebservice`. O método `Notify` dispara o observador, no caso, o analisador concreto. Finalmente, nas linhas 31 a 34 é instanciado o tempo até iniciar o próximo monitoramento.

Figura 16 – Exemplo de configuração do monitor concreto

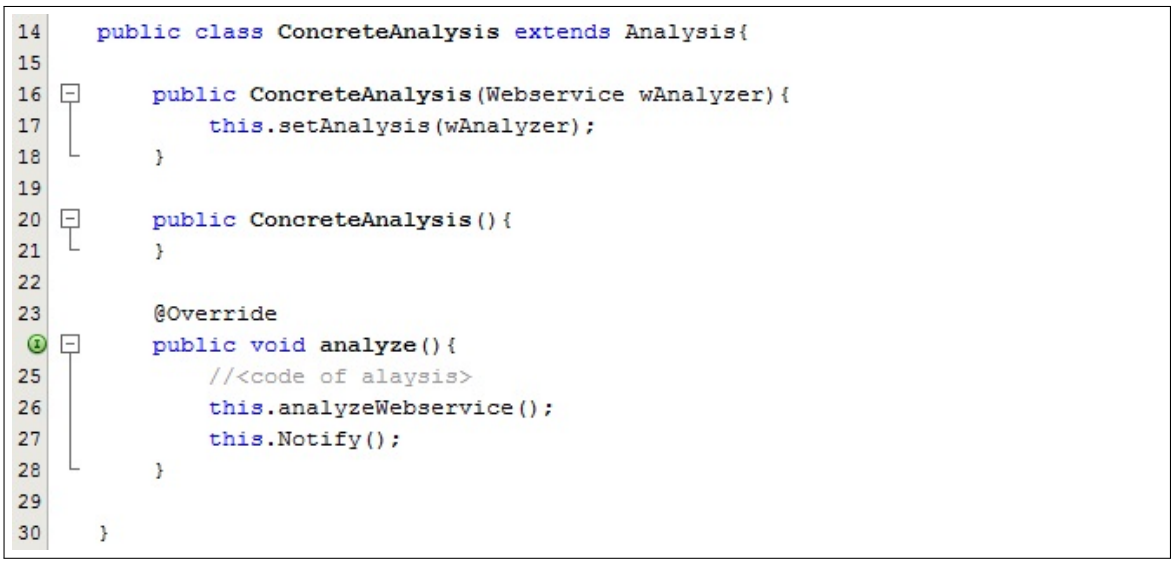
```
14  public class ConcreteMonitoring extends Monitoring{
15
16      public ConcreteMonitoring(Webservice wMonitor){
17          this.setMonitoring(wMonitor);
18      }
19
20      public ConcreteMonitoring(){
21      }
22
23      @Override
24      public void monitoring(){
25           //<code of monitoring>
26          this.monitoringWebservice();
27          //Alert Observers
28          this.Notify();
29
30          //Time until next read
31          try {
32             Thread.sleep(getTime()*1000);  //each getTime() seconds
33          } catch (Exception e) {
34          }
35      }
36
37  }
```

Fonte: Elaborado pelo autor

A Figura 17 apresenta um exemplo de configuração do analisador concreto. Nas linhas 16 a 18 é declarado um construtor para o caso em que são utilizados *web services*. Nas linhas 20 e 21 é declarado um construtor vazio. As linhas 24 a 28 apresentam a função que contém a lógica de análise. Da linha 25 até a linha anterior à linha em que é executado o método `Notify` deve constar a lógica de análise. Se a lógica de monitoramento estiver contida em um serviço basta executar o método `analyzeWebservice` que irá executar esse *webservice*. Caso não se utilize webservices a lógica de análise deve ser adicionada ao método `analyze` do analisador concreto. Nas linhas 20 e 21 é declarado um construtor vazio. Por fim, o método `Notify` dispara o observador, no caso, o planejador concreto.

A Figura 18 apresenta um exemplo de configuração do planejador concreto. Nas

Figura 17 – Exemplo de configuração do analisador concreto

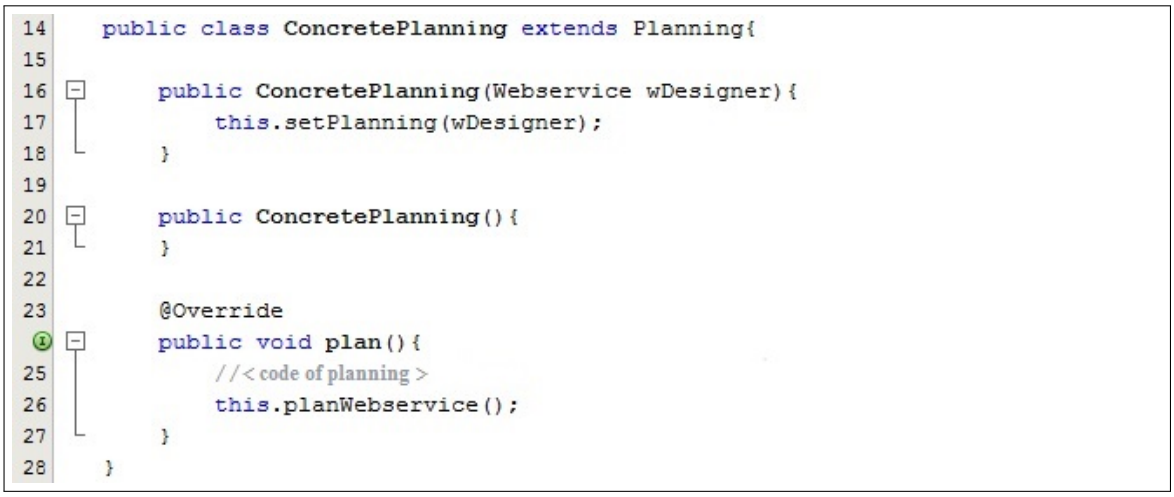


```
14  public class ConcreteAnalysis extends Analysis{
15
16      public ConcreteAnalysis(Webservice wAnalyzer){
17          this.setAnalysis(wAnalyzer);
18      }
19
20      public ConcreteAnalysis(){
21      }
22
23      @Override
24      public void analyze(){
25          //<code of alaysis>
26          this.analyzeWebservice();
27          this.Notify();
28      }
29
30  }
```

Fonte: Elaborado pelo autor

linhas 16 a 18 é declarado um construtor para o caso em que são utilizados *web services*. Nas linhas 20 e 21 é declarado um construtor vazio. As linhas 24 a 27 apresentam a função que contém a lógica de planejamento. Da linha 25 até o final deve constar a lógica de planejamento. Se a lógica de planejamento estiver contida em um serviço basta executar o método `planWebservice`. Caso não se utilize webservices a lógica de planejamento deve ser adicionada ao método `plan` do planejador concreto. Nas linhas 20 e 21 é declarado um construtor vazio.

Figura 18 – Exemplo de configuração do planejador concreto



```
14  public class ConcretePlanning extends Planning{
15
16      public ConcretePlanning(Webservice wDesigner){
17          this.setPlanning(wDesigner);
18      }
19
20      public ConcretePlanning(){
21      }
22
23      @Override
24      public void plan(){
25          //< code of planning >
26          this.planWebservice();
27      }
28  }
```

Fonte: Elaborado pelo autor

Por fim, a Figura 19 apresenta um exemplo de configuração da classe main do módulo de aplicação. Na linha 25 é instanciado um objeto do tipo Webservice. Na linha 26 é

instanciado um objeto do tipo File que contém a lista de serviços informada pelo planejador concreto. Na linha 27 é instanciado um objeto do tipo WSFuncs. Na linha 28 é instanciado a View e na linha 29 é instanciada uma variável auxiliar utilizada para identificar serviços na lista de serviços contido no objeto do tipo File. As linhas 31 e 32 são utilizadas para iniciar a execução do módulo Adaptador. As linhas 34 a 52 contém o loop de execução desta aplicação. A linha 36 chama a função que apresenta a interface gráfica da aplicação. Nas linhas 38 e 39 é verificada a condição de parada do loop. Na linha 41 é inicializado o objeto do tipo Webservice selecionado da lista de serviços contido no objeto do tipo File. Nas linhas 44 a 52 é verificado se o serviço escolhido realmente está na lista de serviços, caso esteja, esse serviço é executado.

Figura 19 – Exemplo de configuração da classe main do módulo de aplicação

```
23      public static void main(String[] args) throws IOException {
24
25          Webservice service = new Webservice();
26          File Services = new File("Services.xml");
27          WSFuncs funcs = new WSFuncs();
28          GenericView view = new GenericView();
29          String idstop = "";
30
31          String commands = "java" + "- jar" + "Path the GenericAdapterProgram.jar";
32          Process p  = Runtime.getRuntime().exec(commands);
33
34          while(!idstop.equals("0")) //Loop of execution
35          {
36              idstop = view.WriteinScreen();
37
38              if(idstop.equals("0"))
39                  continue;
40
41              service = funcs.SearchServiceByID(idstop,Services);
42
43              if(service!=null){
44                  if(service.getarguments().size() > 0)
45                      funcs.fService(service, service.getarguments().toArray(
46                                      new String[service.getarguments().size()]));
47                  else
48                      funcs.fService(service, service.getarguments().toArray(
49                                      new String[service.getarguments().size()]));
50              }
51              else
52                  System.out.printf("Requested service does not exist. "
53                                      + "Please choose another service\n");
54          }
```

Fonte: Elaborado pelo autor

Não é apresentado como criar os *web services*, pois tal etapa não faz parte do escopo do framework SASeS, já que cada serviço possui uma lógica própria que vai variar de aplicação para aplicação. Como a adaptação proposta pelo SASeS não altera a lógica dos serviços, a

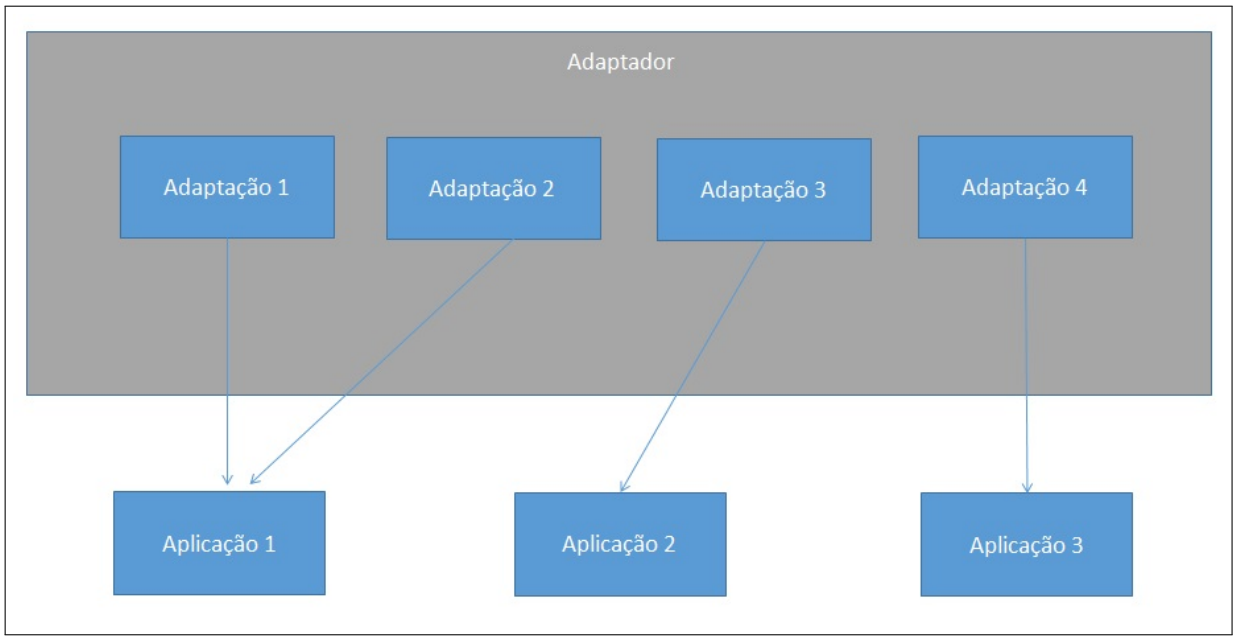
criação destes pode ser a última etapa do desenvolvimento ou pode ser executada em paralelo com as etapas anteriores.

4.5 CENÁRIOS

O SASeS permite uma flexibilidade de criação de cenários. A princípio o mais comum seria onde uma aplicação baseada em serviços está associada a um único adaptador. É possível também que uma única aplicação esteja associada a mais de um adaptador e também que um adaptador esteja associado a mais de uma aplicação. Dessa forma, pode-se reutilizar a lógica de um mesmo adaptador para várias aplicações ou modularizar o desenvolvimento de possíveis adaptações relacionadas a uma aplicação separando as adaptações em adaptadores diferentes.

A Figura 20 mostra um adaptador que gera adaptações para várias aplicações. Nesse caso são quatro adaptações, as duas primeiras são relacionadas a uma mesma aplicação e as outras duas são relacionadas a uma aplicação cada.

Figura 20 – Primeiro exemplo de cenário de criação de sistemas usando o SASeS

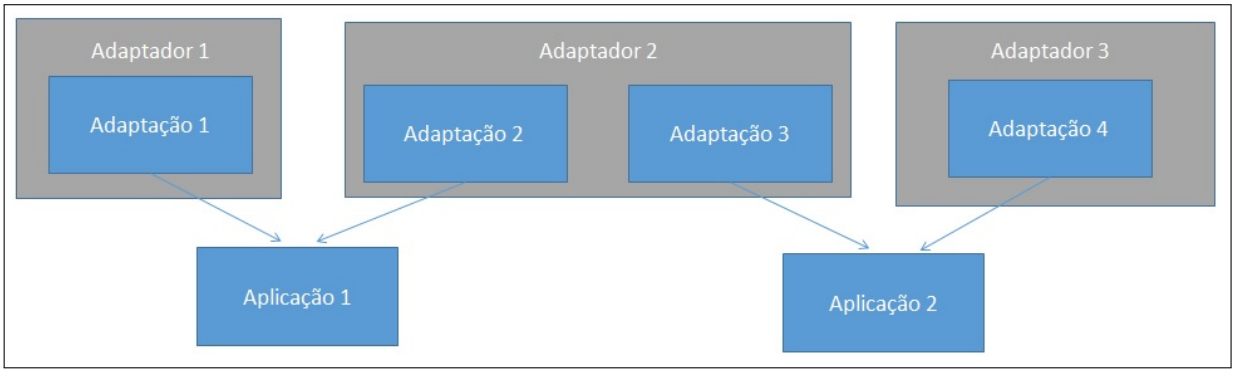


Fonte: Elaborado pelo autor

A Figura 21 mostra aplicações que utilizam adaptações geradas por mais de um adaptador. Nesse caso, o primeiro adaptador gera apenas uma adaptação que é associada a uma aplicação. O segundo adaptador gera duas adaptações, a primeira associada a mesma aplicação

que é associada a adaptação gerada pelo primeiro adaptador, e a segunda adaptação é associada a um outra aplicação que por sua vez também associada si está outra adaptação que é gerada por um terceiro adaptador.

Figura 21 – Segundo exemplo de cenário de criação de sistemas usando o SASeS



Fonte: Elaborado pelo autor

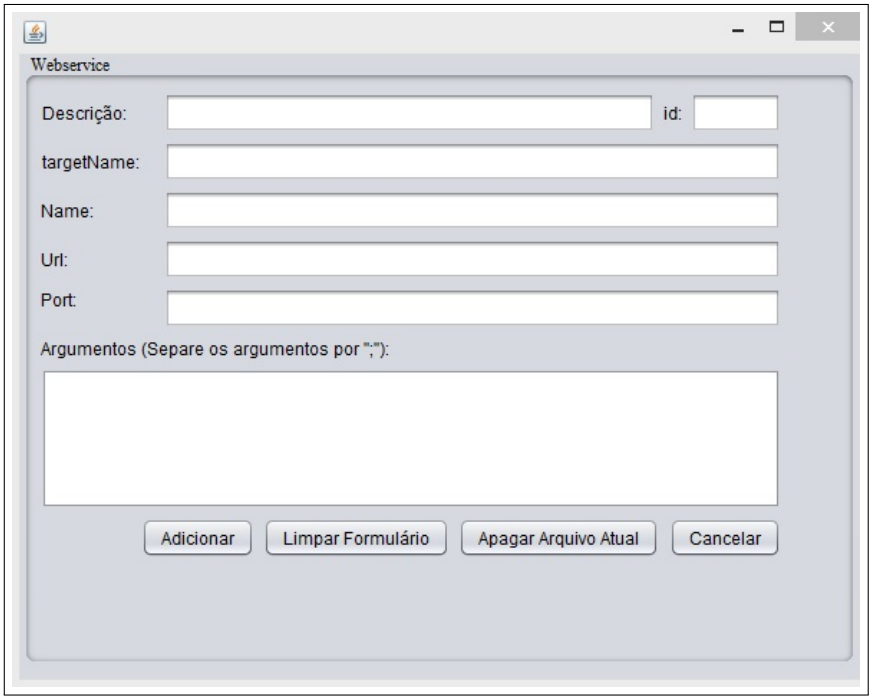
Os estudos de caso apresentados na próxima seção utilizam o primeiro cenário. Um exemplo de aplicação para o segundo cenário seria quando existe um módulo adaptador já desenvolvido para outra aplicação e uma das adaptações que ele executa pode ser utilizada por uma nova aplicação que o desenvolvedor irá criar. Deste modo poderia ser utilizado o adaptador que já está pronto. Caso haja outras adaptações possíveis para aplicação, que o módulo adaptador já existente não contempla, seria criado um outro módulo adaptador para contemplar essas adaptações, logo a aplicação utilizaria dois adaptadores diferentes.

4.6 A FERRAMENTA WebserviceXMLCreator

Para facilitar a criação de arquivos XML contendo listas de serviços (*web services*) foi criada a ferramenta WebserviceXMLCreator, que é uma interface composta de um formulário que deve ser preenchido com as informações referentes a cada *web service* que constará na lista final. A Figura 12, apresentada anteriormente, mostra um exemplo de arquivo ".xml"gerado por esta ferramenta. A Figura 22 mostra a interface da versão atual da ferramenta WebserviceXMLCreator.

Esta ferramenta cria um arquivo XML com nome "WebserviceXML.xml"na pasta em que ela se encontra. O botão `Adicionar` adiciona um *web service* com as informações que estão nos campos do formulário. Caso o campo não seja preenchido é adicionada a informação vazia e se o arquivo "WebserviceXML.xml"ainda não existir, ele é criado. O botão `Limpar`

Figura 22 – Ferramenta WebserviceXMLCreator



Fonte: Elaborado pelo autor

`Formulário` limpa os campos do formulário. O botão `Apagar arquivo atual` deleta o arquivo "WebserviceXML.xml". Por fim, o botão `Cancelar` fecha a aplicação.

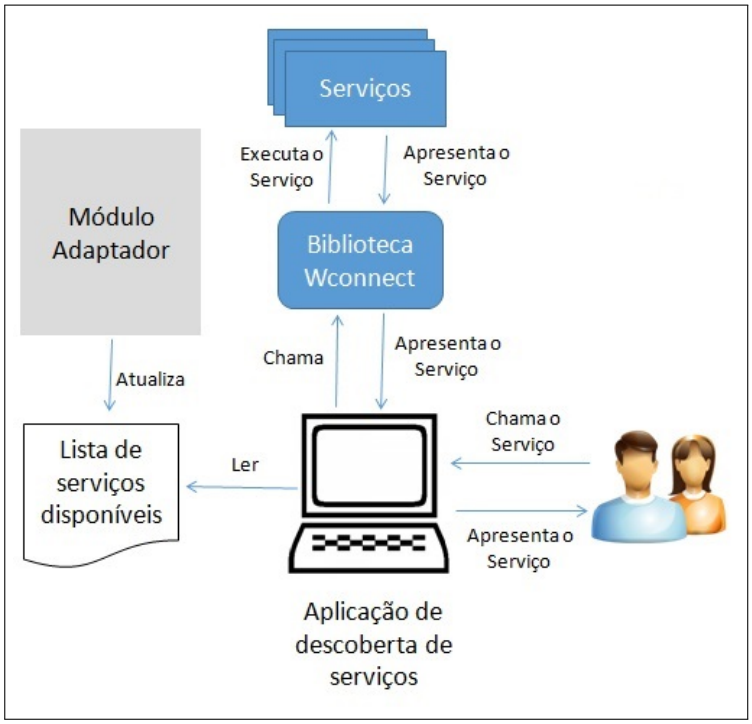
5 ESTUDOS DE CASO

Este capítulo apresenta três estudos de caso que ilustram a aplicabilidade do framework SASeS.

5.1 SISTEMA DE DESCOBERTA DE SERVIÇOS

Este estudo de caso[1] simula o exemplo mostrado por Camara et al. (2009) onde um sistema de descoberta de serviços em um aeroporto é proposto. Neste trabalho, é apresentado um sistema de descoberta de serviços genérico. O sistema tem como objetivo checar 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. A seguir é descrito como o framework SASeS foi usado na construção do módulo adaptador e da aplicação para o sistema de descoberta de serviços.

Figura 23 – Arquitetura simplificada da aplicação de descoberta de serviços



Fonte: Elaborado pelo autor

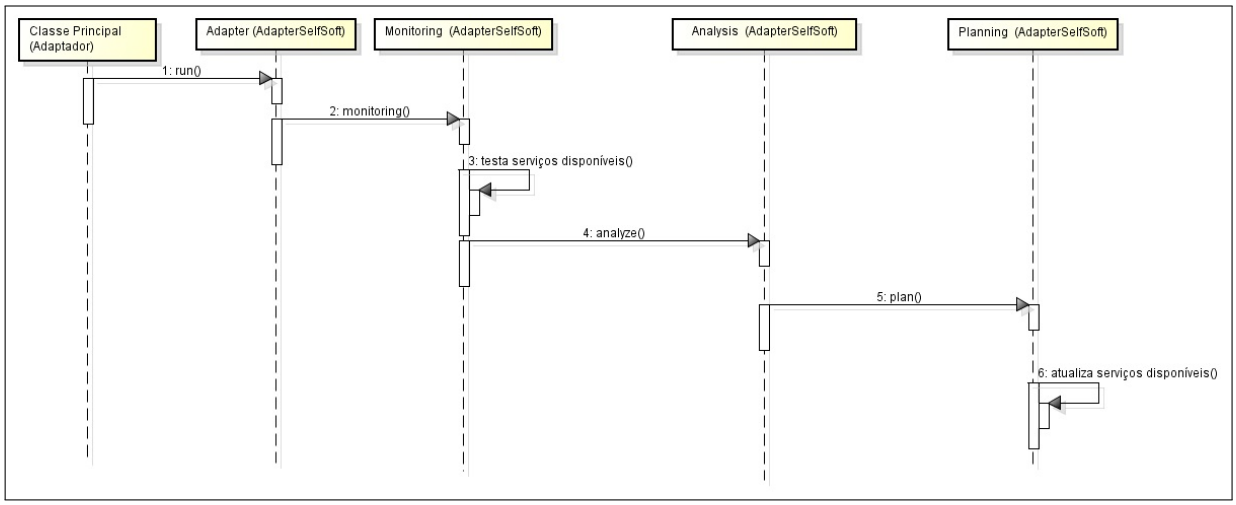
Neste estudo de caso, dois tipos de adaptação são necessárias: remoção de serviços que pararam de funcionar e adição de novos serviços funcionais. Para tal, duas classes de cada

[1] Veja o código do estudo de caso em: http://goo.gl/DsU05x

componente do ciclo de adaptação foram criadas: um para tratar o primeiro tipo de adaptação (remoção de serviços), e o outro para o segundo tipo. Para simular a descoberta de serviços, é usado um arquivo XML que contém uma lista de serviços disponíveis. A Figura 23 apresenta a arquitetura simplificada do sistema.

A Figura 24 mostra o diagrama de sequência do caso da remoção de serviços. O processo começa quando a classe principal do módulo adaptador executa o método `run` da classe Adapter. Isso dispara a execução do monitor concreto que lê a lista de serviços disponíveis e testa se eles estão funcionando. Os serviços não operacionais são escritos no arquivo de saída do monitor indicando que devem ser removidos. Em seguida, o analisador concreto é notificado que a lista de serviços não disponíveis foi atualizada. Como o analisador já sabe os serviços que tem de ser removidos, ele simplesmente notifica o planejador concreto. Finalmente, a última classe efetivamente remove os serviços da lista de serviços disponíveis.

Figura 24 – Diagrama de sequência da remoção de serviços



Fonte: Elaborado pelo autor

O processo de adição de serviços é muito similar à sua contraparte. Ele começa quando o método `run` do Adapter que dispara a execução do segundo monitor concreto, que checa se há novos serviços disponíveis no ambiente, e caso haja, salva eles no arquivo de saída do monitor. Subsequentemente, é disparado a execução do analisador, que lê o arquivo, testa os novos serviços descobertos pelo monitor e passa a nova lista com os serviços funcionais para o planejador. Finalmente, o planejador adiciona os serviços reportados pelo analisador a lista de serviços disponíveis que irão ser usados pela aplicação.

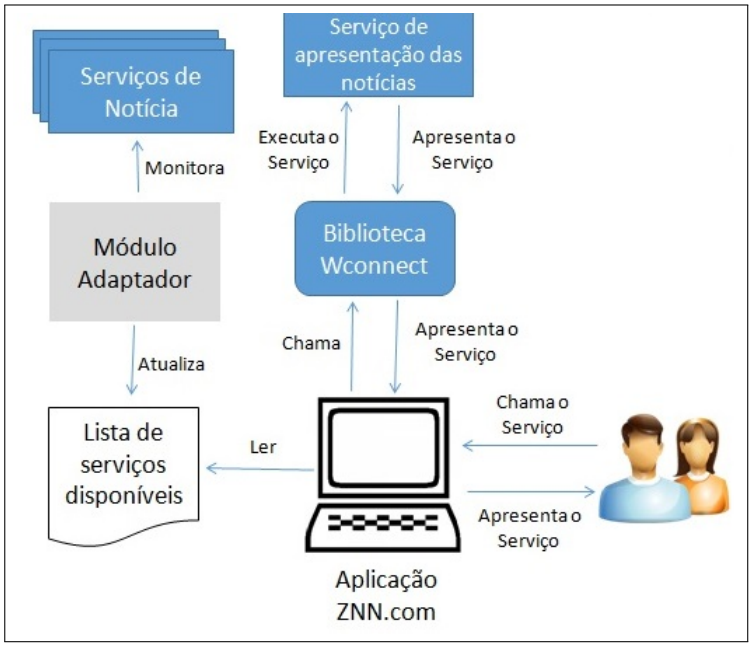
A aplicação baseada em serviços consiste na customização da classe principal do

módulo de aplicação. Ela se mantém constantemente lendo a lista de serviços disponíveis que está sendo atualizada pelo Módulo adaptador e apresenta a lista ao usuário, que pode executar qualquer um dos serviços ou pode simplesmente atualizar a lista para verificar se há novos serviços.

5.2 APLICAÇÃO ZNN.COM

Este estudo de caso[2] adapta o estudo de caso apresentado em (CHENG et al., 2009b) que é baseado em sites reais como o *cnn.com* e o *RockyMountainNews.com.* O Znn.com original é um serviço de notícias que serve novos conteúdos de multimídia para seus clientes. Para esse trabalho foi criada uma aplicação que tem a mesma funcionalidade que o Znn.com original. A aplicação Znn.com lê uma lista de novos serviços (*web services*) multimídia e apresenta a notícia selecionada ao usuário.

Figura 25 – Arquitetura simplificada da aplicação Znn.com



Fonte: Elaborado pelo autor

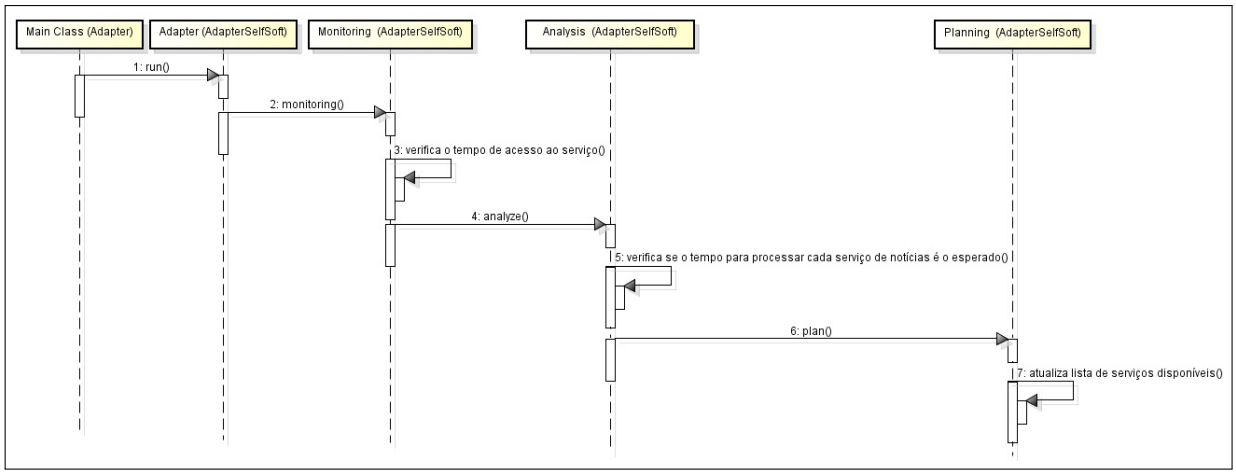
Neste estudo de caso, para adicionar e remover novos serviços pode-se utilizar a mesma lógica do Sistema de Descoberta de Serviços. Então, somente é implementada uma adaptação para este sistema quando o acesso aos novos serviços é muito demorado. Nesse caso, o sistema apenas exibe o texto da notícia e outros elementos multimídia são omitidos. Uma classe

[2] Veja o código do estudo de caso em: http://goo.gl/4OuzRf

de cada componente do ciclo de adaptação foi criada. Para simular a descoberta de serviços, é usado um arquivo XML que contém a lista de novos serviços. A Figura 25 apresenta a arquitetura simplificada do sistema.

A Figura 26 mostra o diagrama de sequência deste caso. O processo começa quando a classe principal do módulo adaptador executa o método `run` da classe Adapter. Ele dispara a execução do monitor concreto que lê a lista de novos serviços de notícias disponíveis e verifica o tempo necessário para acessar cada serviço. Os serviços de notícia são escritos no arquivo de saída do monitor junto com a informação do tempo necessário para acessar cada um deles e em seguida o analisador é notificado.

Figura 26 – Diagrama de sequência do módulo adaptador da aplicação Znn.com



Fonte: Elaborado pelo autor

O analisador lê o arquivo e verifica se o tempo para processar os novos serviços está dentro do esperado. Depois é adicionado ao arquivo de saída do analisador os novos serviços de notícia e a informação se o tempo de acesso a cada um deles está dentro do esperado e então o planejador é notificado.

Finalmente o planejador, usando a informação passada pelo analisador, informa à aplicação a lista de notícias e o código de geração da página HTML (*HyperText Markup Language*), se o tempo de acesso para o serviço de notícia é maior que o esperado este código irá gerar um página apenas com texto, caso contrário, a página irá também apresentar os componentes de multimídia (imagens e vídeos) associados às notícias.

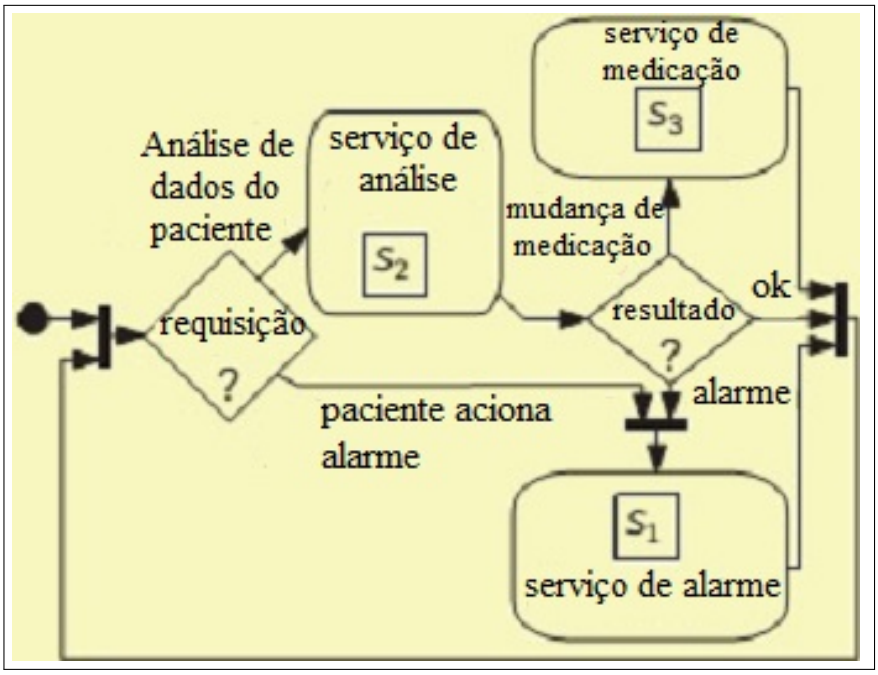
Além da lista de notícias e o código de geração da página HTML, o planejador também informa o serviço (*web bservice*) que a aplicação utiliza para, com base no código informado, gerar o arquivo HTML e apresentar a notícia selecionada ao usuário.

A aplicação baseada em serviços consiste de uma customização da classe principal do módulo de aplicação. Esta classe apresenta a lista de notícias e chama o serviço que gera a página HTML da notícia de acordo com o código inserido no arquivo que contém a lista de notícias, que é atualizado pelo módulo adaptador, e apresenta a notícia selecionada pelo usuário.

5.3 SISTEMA DE ASSISTÊNCIA MÉDICA À DISTÂNCIA

Este estudo de caso[3] é um exemplo similar ao proposto em (CALINESCU et al., 2012), ilustrado na Figura 27, que consiste de um sistema de assistência médica à distância.

Figura 27 – Sistema de Assistência Médica à distância



Fonte: (CALINESCU et al., 2012)

A aplicação desenvolvida tem 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 a função de informar a dosagem do medicamento a ser utilizado pelo paciente, caso este subsistema seja acionado; e (iii) o subsistema de alarme, que é acionado quando o paciente encontra-se em estado grave ou caso o paciente o acione. Uma vez acionado o subsistema de alarme é necessário uma intervenção médica local. Considera-se que há chances de cada um desses subsistemas falhar.

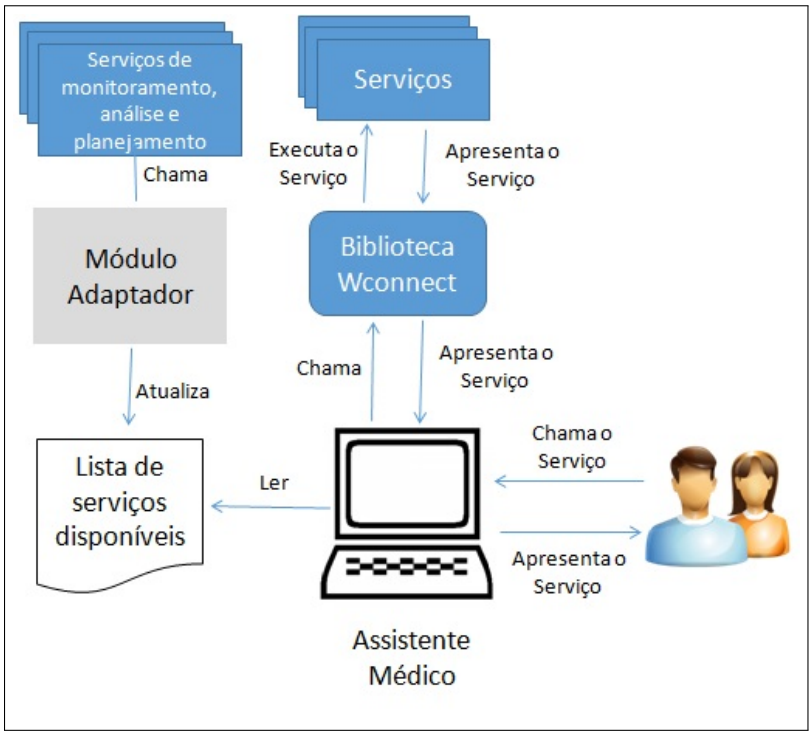
Neste estudo de caso, foi escolhido trabalhar com uma possível adaptação relacionada

[3] Veja o código do estudo de caso em: http://goo.gl/HDw3sy

ao subsistema de alarme a fim de evitar ou minimizar problemas decorrentes de uma falha nesse subsistema. A adaptação consiste em selecionar um dos três *web services* que implementam o subsistema de alarme. Esta adaptação pode ser melhor compreendida como uma forma de autoverificação. O mesmo estudo de caso é usado em (CALINESCU et al., 2013).

Neste estudo de caso a lógica de monitoramento, análise e planejamento estão contidos em *web services* que são disparados pelo monitor concreto, analisador concreto e planejador concreto, respectivamente. Esses *web services* são informados na classe principal do módulo adaptador. A Figura 28 apresenta a arquitetura simplificada do sistema.

Figura 28 – Arquitetura simplificada do sistema de assistência médica

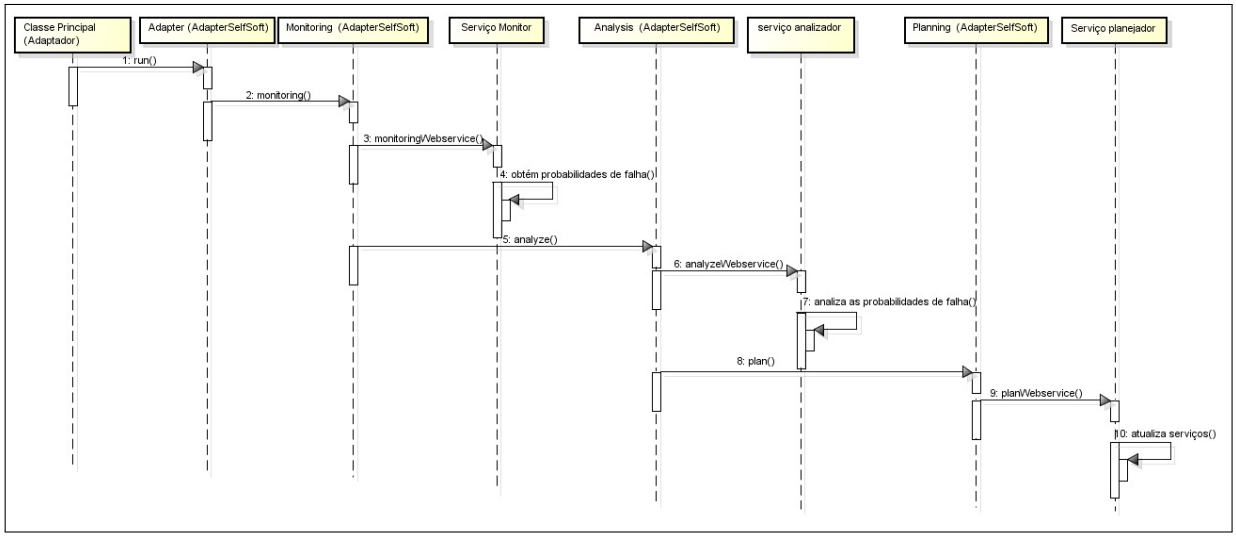


Fonte: Elaborado pelo autor

A Figura 29 mostra o diagrama de sequência desse caso. O processo começa quando a classe principal do módulo adaptador executa o método `run` da classe Adapter. Ele 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. Depois o monitor concreto notifica o analisador.

O analisador concreto executa o método `analyzeWebservice` para chamar o serviço analisador. O serviço analisador usa uma ferramenta de checagem de modelos probabilísticos

Figura 29 – Diagrama de sequência do módulo adaptador do Sistema de Assistência Médica

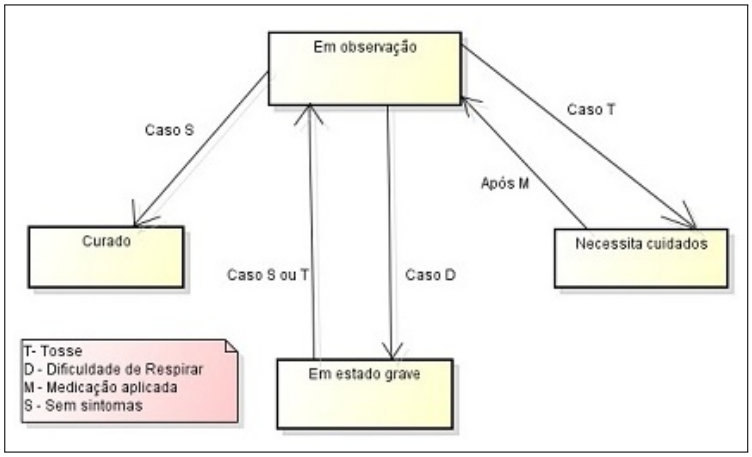


Fonte: Elaborado pelo autor

chamado PRISM (HINTON et al., 2006), que lê uma cadeia de Markov de tempo discreto (a mesma sugerida em (CALINESCU et al., 2012)) e adiciona a probabilidade de falha de cada subsistema de alarme obtido pelo serviço monitor e, caso essa probabilidade seja maior ou menor que um certo valor, a ferramenta gera um retorno verdadeiro ou falso. Então, o serviço analisador seleciona o serviço de subsistema de alarme que apresenta um valor verdadeiro e adiciona essa informação ao arquivo de saída do analisador. Depois o analisador concreto notifica o planejador.

O planejador executa o método `planWebservice` para chamar o serviço planejador. O serviço planejador informa a aplicação por *sockets* que serviços implementam o subsistema da aplicação incluindo o serviço de subsistema de alarme informado pelo serviço analisador.

Figura 30 – Mudanças de situação do paciente no sistema de assistência médica



Fonte: Elaborado pelo autor

A aplicação baseada em serviço consiste na customização da classe principal do módulo de aplicação. Esta classe recebe a informação via *sockets* do serviço planejador de quais serviços implementam os subsistemas da aplicação.

A aplicação recebe a informação de qual sintoma o paciente está apresentando e, em seguida, a aplicação chama o serviço de subsistema de análise que informa se deve ser chamado o serviço de subsistema de medicação ou o serviço de subsistema de alarme, ou se o paciente está curado, neste último caso o sistema encerra sua execução. A Figura 30 apresenta as mudanças de situação do paciente de acordo com os sintomas.

6 CONSIDERAÇÕES FINAIS

Nesta dissertação foi apresentado o SASeS, um framework para auxiliar o desenvolvimento de sistemas autoadaptativos baseados em serviços. O framework provê um conjunto de classes para construção de um módulo adaptador externo, a aplicação baseada em serviços, e os serviços usados pela aplicação. O framework também provê uma biblioteca que facilita a comunicação entre os *web services*, a aplicação e o gerenciador autonômico.

São apresentados também três estudos de caso que utilizam o framework SASeS. O primeiro estudo de caso simula um sistema de descoberta de serviços, o segundo apresenta um agregador de serviços de notícias e o terceiro apresenta um sistema de assistência médica a distância.

6.1 CONCLUSÕES

Ao ser apresentado e descrito o framework SASeS no capítulo 4, o primeiro objetivo específico do trabalho foi alcançado. Ao final do capítulo 4 também é apresentada a ferramenta WebserviceXMLCreator, que comprova que o segundo objetivo também foi alcançado. No capítulo 5, são apresentados os estudos de casos utilizados para demonstrar a aplicabilidade do framework, deste modo o terceiro e último objetivo específico também é alcançado.

O framework SASeS se mostrou uma ferramenta simples para auxiliar o desenvolvimento de sistemas autoadaptativos baseados em serviços, utilizando uma abordagem externa, na qual um componente chamado Módulo Adaptador implementa as etapas do ciclo de adaptação.

O framework SASeS encontra-se em fase de evolução e várias melhorias ainda podem ser adicionadas. Em princípio é interessante verificar a necessidade de uma possível refatoração do código, para torná-lo ainda mais simples. A versão atual o framework possibilita ampla reusabilidade e fornece vários cenários para criação de sistemas autoadaptativos baseados em serviços.

Outras possíveis melhorias são apresentadas na sessão 6.3, que contém os trabalhos futuros. É importante destacar que o código da versão atual do framework é aberto.

6.2 LIMITAÇÕES

Uma das limitações deste trabalho e que serve como um desafio futuro é falta de uma avaliação de como o framework SASeS pode garantir que requisições relacionadas aos fatores de

qualidade de serviço (QoS) foram alcançadas. Nesse caso, cabe ao projetista de cada aplicação tomar as precauções necessárias para garantir que esses requisitos sejam alcançados.

Além disso, como o SASeS foi elaborado com foco em sistemas baseados em serviços, não foi avaliado que modificações seriam necessárias para que o mesmo possa ser utilizado por outros tipos de sistemas.

6.3 TRABALHOS FUTUROS

Existem alguns trabalhos futuros que poderão ser desenvolvidos com o intuito de dar continuidade ao trabalho apresentado nesta dissertação. Dentre eles:

a) Uso de *proxies* inteligentes para seleção de serviços operacionais de diferentes fontes, como proposto em (CALINESCU et al., 2013).
b) Criação de uma versão mobile do framework, a fim de criar aplicações reais sensíveis à contexto.
c) Extensão do framework SASeS para usar *web services* desenvolvidos em outros frameworks que não apenas o JAX-WS.
d) Avaliação da vantagem de trabalhar com uma camada objetivo, como proposto em (KRAMER; MAGEE, 2007).
e) Utilização de uma abordagem MDE integrada ao SASeS para gerar sistemas autoadaptativos baseados em serviços de maneira automática.
f) Também é interessante avaliar como implementar cada propriedade auto-* utilizando o SASeS. A avaliação de cada propriedade pode ser entendida com um tema de pesquisa.

REFERÊNCIAS

AAMODT, A.; PLAZA, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. **AI communications**, IOS Press, v. 7, n. 1, p. 39–59, 1994.

ABUFOUDA, M. Quality-aware approach for engineering self-adaptive software systems. **Third International Conference on Information Technology Convergence and Services**, 2014.

ADAMCZYK, J.; CHOJNACKI, R.; JARZAB, M.; ZIELIŃSKI, K. Rule engine based lightweight framework for adaptive and autonomic computing. In: **Computational Science–ICCS 2008**. [S.l.]: Springer, 2008. p. 355–364.

AFFONSO, F. J.; NAKAGAWA, E. Y. A reference architecture based on reflection for self-adaptive software. In: IEEE. **Software Components, Architectures and Reuse (SBCARS), 2013 VII Brazilian Symposium on**. [S.l.], 2013. p. 129–138.

AFFONSO, F. J.; SCANNAVINO, K. R.; OLIVEIRA, L. B.; NAKAGAWA, E. Y. Reference architectures for self-managed software systems: A systematic literature review. In: IEEE. **Software Components, Architectures and Reuse (SBCARS), 2014 Eighth Brazilian Symposium on**. [S.l.], 2014. p. 21–31.

ANDERSSON, J.; LEMOS, R. D.; MALEK, S.; WEYNS, D. Modeling dimensions of self-adaptive software systems. In: **Software engineering for self-adaptive systems**. [S.l.]: Springer, 2009. p. 27–47.

ASADOLLAHI, R.; SALEHIE, M.; TAHVILDARI, L. Starmx: A framework for developing self-managing java-based systems. In: IEEE. **Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on**. [S.l.], 2009. p. 58–67.

AUTILI, M.; BENEDETTO, P. D.; INVERARDI, P. A programming model for adaptable java applications. In: ACM. **Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java**. [S.l.], 2010. p. 119–128.

BARESI, L.; GHEZZI, C. The disappearing boundary between development-time and run-time. In: **Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research**. New York, NY, USA: ACM, 2010. (FoSER '10), p. 17–22. ISBN 978-1-4503-0427-6. Disponível em: <http://doi.acm.org/10.1145/1882362.1882367>.

BUCKLEY, J.; MENS, T.; ZENGER, M.; RASHID, A.; KNIESEL, G. Towards a taxonomy of software change. **Journal of Software Maintenance and Evolution: Research and Practice**, Wiley Online Library, v. 17, n. 5, p. 309–332, 2005.

BURG, S. van der; DOLSTRA, E. A self-adaptive deployment framework for service-oriented systems. In: ACM. **Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems**. [S.l.], 2011. p. 208–217.

CALINESCU, R.; GHEZZI, C.; KWIATKOWSKA, M.; MIRANDOLA, R. Self-adaptive software needs quantitative verification at runtime. **Communications of the ACM**, ACM, v. 55, n. 9, p. 69–77, 2012.

CALINESCU, R.; JOHNSON, K.; RAFIQ, Y. Developing self-verifying service-based systems. In: IEEE. **Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on**. [S.l.], 2013. p. 734–737.

CALINESCU, R.; RAFIQ, Y. Using intelligent proxies to develop self-adaptive service-based systems. In: IEEE. **Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on**. [S.l.], 2013. p. 131–134.

CAMARA, J.; CANAL, C.; SALAUN, G. Behavioural self-adaptation of services in ubiquitous computing environments. In: IEEE. **Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on**. [S.l.], 2009. p. 28–37.

CÁMARA, J.; LEMOS, R. de. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In: IEEE. **Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on**. [S.l.], 2012. p. 53–62.

CHENG, B. H.; LEMOS, R. D.; GIESE, H.; INVERARDI, P.; MAGEE, J.; ANDERSSON, J.; BECKER, B.; BENCOMO, N.; BRUN, Y.; CUKIC, B. et al. Software engineering for self-adaptive systems: A research roadmap. In: **Software engineering for self-adaptive systems**. [S.l.]: Springer, 2009. p. 1–26.

CHENG, S.-W.; GARLAN, D.; SCHMERL, B. Evaluating the effectiveness of the rainbow self-adaptive system. In: IEEE. **Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on**. [S.l.], 2009. p. 132–141.

COMPUTING, A. et al. An architectural blueprint for autonomic computing. **IBM White Paper**, Citeseer, 2006.

DERAKHSHANMANESH, M.; AMOUI, M.; O'GRADY, G.; EBERT, J.; TAHVILDARI, L. Graf: graph-based runtime adaptation framework. In: ACM. **Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems**. [S.l.], 2011. p. 128–137.

ELKHODARY, A.; ESFAHANI, N.; MALEK, S. Fusion: a framework for engineering self-tuning self-adaptive software systems. In: ACM. **Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering**. [S.l.], 2010. p. 7–16.

FRANCISCO, E. d. R. F. **DEVELOPING SERVICE-ORIENTED APPLICATIONS ON THE SEMANTIC WEB**. Tese (Mestrado) — PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, 2003.

GAMMA, E. **Padrões de Projetos: Soluções Reutilizáveis**. [S.l.]: Bookman, 2007.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns: elements of reusable object-oriented software**. [S.l.]: Pearson Education, 1994.

GARLAN, D.; CHENG, S.-W.; HUANG, A.-C.; SCHMERL, B.; STEENKISTE, P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. **Computer**, IEEE, v. 37, n. 10, p. 46–54, 2004.

GORLA, A.; PEZZE, M.; WUTTKE, J.; MARIANI, L.; PASTORE, F. Achieving cost-effective software reliability through self-healing. **Computing & Informatics**, v. 29, n. 1, 2010.

GORTON, I.; LIU, Y.; TRIVEDI, N. An extensible, lightweight architecture for adaptive j2ee applications. In: ACM. **Proceedings of the 6th international workshop on Software engineering and middleware**. [S.l.], 2006. p. 47–54.

GRUNSKE, L.; ZHANG, P. Monitoring probabilistic properties. In: ACM. **Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering**. [S.l.], 2009. p. 183–192.

HINCHEY, M. G.; STERRITT, R. Self-managing software. **Computer**, IEEE, v. 39, n. 2, p. 107–109, 2006.

HINTON, A.; KWIATKOWSKA, M.; NORMAN, G.; PARKER, D. Prism: A tool for automatic verification of probabilistic systems. In: **Tools and Algorithms for the Construction and Analysis of Systems**. [S.l.]: Springer, 2006. p. 441–444.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing—degrees, models, and applications. **ACM Computing Surveys (CSUR)**, ACM, v. 40, n. 3, p. 7, 2008.

INVERARDI, P.; MANCINELLI, F.; NESI, M. A declarative framework for adaptable applications in heterogeneous environments. In: ACM. **Proceedings of the 2004 ACM symposium on Applied computing**. [S.l.], 2004. p. 1177–1183.

INVERARDI, P.; TIVOLI, M. The future of software: Adaptation and dependability. In: **Software Engineering**. [S.l.]: Springer, 2009. p. 1–31.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. **Computer**, IEEE, v. 36, n. 1, p. 41–50, 2003.

KOKAR, M. M.; BACLAWSKI, K.; ERACAR, Y. A. Control theory-based foundations of self-controlling software. **Intelligent Systems and their Applications, IEEE**, IEEE, v. 14, n. 3, p. 37–45, 1999.

KRAMER, J.; MAGEE, J. Self-managed systems: an architectural challenge. In: IEEE. **Future of Software Engineering, 2007. FOSE'07**. [S.l.], 2007. p. 259–268.

LADDAGA, R. Self adaptive software problems and projects. In: **Software Evolvability, 2006. SE\'06. Second International IEEE Workshop on**. [S.l.: s.n.], 2006.

LOPES, C. J. F.; RAMALHO, J. C. Web services: Metodologias de desenvolvimento. **XML, Aplicações e Tecnologias Associadas**, 2004.

NITTO, E. D.; GHEZZI, C.; METZGER, A.; PAPAZOGLOU, M.; POHL, K. A journey to highly dynamic, self-adaptive service-based applications. **Automated Software Engineering**, Springer, v. 15, n. 3-4, p. 313–341, 2008.

OLIVEIRA, N.; BARBOSA, L. S. A self-adaptation strategy for service-based architectures. In: **VIII Brazilian Symposium on Software Components, Architectures and Reuse**. Maceió, Alagoas: [s.n.], 2014. (SBCARS'2014, v. 2), p. 44–53. ISSN 2175-7356.

OREIZY, P.; HEIMBIGNER, D.; JOHNSON, G.; GORLICK, M. M.; TAYLOR, R. N.; WOLF, A. L.; MEDVIDOVIC, N.; ROSENBLUM, D. S.; QUILICI, A. An architecture-based approach to self-adaptive software. **IEEE Intelligent systems**, IEEE Computer Society, v. 14, n. 3, p. 54–62, 1999.

PARASHAR, M.; HARIRI, S. Autonomic computing: An overview. In: **Unconventional Programming Paradigms**. [S.l.]: Springer, 2005. p. 257–269.

PERINI, A. Self-adaptive service based applications: Challenges in requirements engineering. In: **RCIS**. [S.l.: s.n.], 2012. p. 1.

SALEHIE, M.; LI, S.; TAHVILDARI, L. Employing aspect composition in adaptive software systems: A case study. In: ACM. **Proceedings of the 1st workshop on Linking aspect technology and evolution**. [S.l.], 2009. p. 17–21.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. **ACM Trans. Auton. Adapt. Syst.**, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, maio 2009. ISSN 1556-4665.

SAMMAPUN, U.; LEE, I.; SOKOLSKY, O. Rt-mac: runtime monitoring and checking of quantitative and probabilistic properties. In: IEEE. **Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on**. [S.l.], 2005. p. 147–153.

STERRITT, R.; BUSTARD, D. Autonomic computing—a means of achieving dependability? In: IEEE COMPUTER SOCIETY. **Engineering of Computer-Based Systems, IEEE International Conference on the**. [S.l.], 2003. p. 247–247.

SZVETITS, M.; ZDUN, U. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. **Software & Systems Modeling**, Springer, p. 1–39, 2013.

WONG, E. Y. C.; CHAN, A. T. S.; LEONG, H. V. Xstream: A framework for the efficient streaming of xml documents over a wireless environment. In: KING, I.; MáRAY, T. (Ed.). **WWW (Posters)**. [s.n.], 2003. Disponível em: <http://dblp.uni-trier.de/db/conf/www/www2003p.html#WongCL03>.

YANG, Q.; LÜ, J.; LI, J.; MA, X.; SONG, W.; ZOU, Y. Toward a fuzzy control-based approach to design of self-adaptive software. In: ACM. **Proceedings of the Second Asia-Pacific Symposium on Internetware**. [S.l.], 2010. p. 15.