Pós-Graduação em Ciência da Computação

Fábio Nogueira de Souza

DSOA: uma plataforma para composição dinâmica de serviços cientes de qualidade

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2018

Fábio Nogueira de Souza

DSOA: uma plataforma para composição dinâmica de serviços cientes de qualidade

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Sistemas Distribuídos
Orientador: Nelson Souto Rosa

Recife
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S729d Souza, Fábio Nogueira de
DSOA: uma plataforma para composição dinâmica de serviços cientes de qualidade / Fábio Nogueira de Souza. – 2018.
158 f.: il., fig., tab.

Orientador: Nelson Souto Rosa.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências.

1. Sistemas distribuídos. 2. Composição de serviços. I. Rosa, Nelson Souto (orientador). II. Título.

004.36 CDD (23. ed.) UFPE- MEI 2018-124

Tese de Doutorado apresentada por **Fábio Nogueira de Souza** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**DSOA: uma plataforma para composição dinâmica de serviços cientes de qualidade**" **Orientador: Nelson Souto Rosa** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Dr. Paulo Romero Martins Maciel
Centro de Informática/ UFPE

Prof. Dr. Jaelson Freire Brelaz de Castro
Centro de Informática/ UFPE

Prof. Dr. Kiev dos Santos Gama
Centro de Informática / UFPE

Prof. Dr. Alfredo Goldman vel Leijbman
Departamento de Ciências da Computação / USP

Prof. Dr. Fábio Moreira Costa
Instituto de Informática / UFG

Visto e permitida a impressão.
Recife, 19 de Fevereiro de 2018.

Prof. Aluizio Fausto Ribeiro Araújo
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

AGRADECIMENTOS

Aos meus pais, Denize e Francisco José, pelo imenso amor, carinho e dedicação demonstrados ao longo de toda a minha existência. Agradeço também pelo exemplo de vida e por me ensinarem que a educação é a riqueza mais importante que um ser humano pode ter.

À minha esposa Andrea, cuja simples existência ilumina os meus caminhos, pelo amor, apoio, e compreensão presentes em todos os momentos no decorrer desta importante jornada. Aos meus filhos Thiago e Ana Carolina por me ensinarem que a vida não se esvai a cada dia, mas sim se renova!

Aos meus irmãos Bruno, Gabriel, Vinicius, Pedro e Gabriela pelo carinho, amizade, respeito, e apoio demonstrados. À minha avó Nanci pelo amor e carinho, e por representar para mim um exemplo de garra e sabedoria. Aos demais familiares por me mostrarem que a família é muito mais do que pais e irmãos fazendo-se sempre presentes em minha vida. Ao meu sogro, José Vasconcelos, e à minha sogra, Luisa Carmen, por me receberem como filho.

Aos meus amigos Tarcisio Coutinho, Francisco Madeiro, e David Cavalcanti por demonstrarem o real sentido da palavra amizade e por todo o apoio que recebi, sem o qual essa jornada seria, sem dúvidas, muito mais difícil.

Ao meu orientador, Nelson Rosa, pela oportunidade que me foi dada. Ao Banco Central do Brasil pelo suporte sem o qual a realização deste sonho não seria possível.

RESUMO

No mundo atual, há uma demanda crescente por uma nova geração de aplicações capazes de se adaptar em função de variações na qualidade dos serviços, sem a necessária intervenção humana. Aplicações com essa capacidade, denominadas aplicações auto-adaptativas, são normalmente implementadas através da introdução de um gerente de adaptação externo, que realiza um laço fechado de controle baseado em modelos mantidos em tempo de execução. A despeito de toda a evolução nessa área, questões importantes continuam em aberto. Em particular, observa-se que as soluções atuais projetam aplicações auto-adaptativas com base em modelos, representando conceitos dos domínios de serviço e qualidade. Em geral, esses modelos são reflexivos, sendo utilizados tanto para descrever os serviços consumidos e providos pela aplicação (e o nível de qualidade correspondente), quanto para viabilizar a sua reconfiguração dinâmica. Embora os modelos citados representem elementos essenciais, eles não possuem informações suficientes para configurar os processos internos de um gerente de adaptação. Em especial, deve-se destacar que os modelos descritos não incorporam nenhuma informação acerca de como o nível de qualidade dos serviços pode ser aferido em tempo de execução. A ausência desse tipo de informação limita a utilidade desses modelos do ponto de vista dos processos de monitoração e análise. Visando preencher essa lacuna, a presente tese propõe a introdução dos conceitos de eventos no espaço de modelagem, e implementa essa visão em uma nova plataforma de apoio ao desenvolvimento e execução de aplicações auto-adaptativas baseadas em serviço e cientes de qualidade, referenciada como *Dynamic Service Oriented Architecture* (DSOA). Na plataforma DSOA, a exposição dos eventos em nível de modelo permite que novas métricas de qualidade sejam definidas em função dos eventos que ocorrem em execução. Mais ainda, uma vez que a plataforma permite definir modelos que determinam como os eventos são processados, o próprio algoritmo de computação das métricas pode ser especificado pelos desenvolvedores. Por fim, como esses modelos são mantidos durante a execução, as métricas podem ser redefinidas e os seus algoritmos de computação dinamicamente modificados. Em suma, a representação integrada dos domínios de serviço, qualidade, e evento viabiliza a construção de uma plataforma orientada a serviços mais flexível, sendo capaz de suportar não somente a adaptação das aplicações em execução, mas também a reconfiguração dinâmica dos próprios gerentes responsáveis pela condução do processo de adaptação.

Palavras-chaves: Aplicações-auto adaptativas. Qualidade de serviço. Models@run.time.

ABSTRACT

Nowadays, there is a growing demand for applications able to adapt themselves at runtime as a result of variations in the expected quality of services. In this context, several research projects propose the utilization of an external manager, which carries out the adaptation process implementing a closed control loop based on a collection of models kept at runtime. In spite of the progress in this area, important issues remain open. In particular, current solutions conceive self-adaptive applications through models representing the concepts of the service and quality domains. Although these models are fundamental, they can not be used to configure the internal processes of an adaptation manager, since they do not contain information concerning how the quality level can be measured at runtime. The absence of the type of information limits the usefulness of these models from the monitoring and analysis points of view. To address this gap, this thesis proposes *Dynamic Service Oriented Architecture* (DSOA) platform, which incorporates the event domain concepts in the applications' modeling space. In this context, new quality metrics can be defined by mapping those metrics to models representing the events that can happen at runtime. Moreover, since the platform allows defining models that determine how the events are processed, the metrics computation algorithms can be specified by the developers. Finally, since these models are maintained during execution, the metrics can be redefined and their computation algorithms dynamically modified. In short, the joint representation of the service, quality and event domains makes it possible to build a truly flexible service-oriented platform capable of supporting not only the adaptation of running applications but also the dynamic reconfiguration of the adaptation managers themselves.

Key-words: Auto-adaptive applications. Quality of service. Models@run.time.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADL	*Architecture Description Language*
API	*Application Programming Interface*
BNF	*Backus-Naur Form*
CBD	*Component-Based Development*
DSL	*Domain-Specific Language* - Linguagem Específica de Domínio
DSOA	*Dynamic Service Oriented Architecture*
DSOA-ML	*DSOA Monitoring Language*
DSOA-EL	*DSOA Event Language*
DSOA-AL	*DSOA Architecture Language*
DSOA-QL	*DSOA Quality Language*
ECA	Evento-Condição-Ação
EMF	*Eclipse Modeling Framework*
LDAP	*Lightweight Directory Access Protocol*
MDE	*Model-Driven Engineering*
MOF	*Meta-Object Facility*
POJO	*Plain Old Java Objects*
QM	*Quality Model*
QSBA	*QoS-aware Service-Based Application*
QSD	*Quality-based Service Description*
SBA	*Service-Based Application*
SCA	*Service Component Architecture*
SOA	*Service-Oriented Architecture*
SOC	*Service-Oriented Computing*
SQMM	*Service Quality Meta-Model*

SUMÁRIO

1 INTRODUÇÃO

> *I keep six honest serving men. They taught me all I knew. Their names are What and Why and When and How and Where and Who.*
>
> Rudyard Kipling,

1.1 CONTEXTO E MOTIVAÇÃO

Service-Oriented Computing (SOC) (PAPAZOGLOU et al., 2007) é um paradigma que visa promover o desenvolvimento de aplicações flexíveis a partir da composição dinâmica de unidades funcionais elementares e auto-contidas referenciadas como serviços. O elemento central de SOC é um estilo arquitetural próprio, conhecido como *Service-Oriented Architecture* (SOA) (PAPAZOGLOU et al., 2007), o qual está fundamentado em três papéis: provedor, consumidor e registro de serviços. De acordo com o modelo de interação implícito no referido padrão arquitetural, os provedores publicam em um registro as descrições dos serviços fornecidos. Esse registro é consultado dinamicamente por consumidores, visando descobrir serviços capazes de desempenhar as funcionalidades requeridas. Uma vez identificados os serviços candidatos, os consumidores podem usar as descrições correspondentes para selecionar aqueles a serem efetivamente utilizados. Neste contexto, provedores, consumidores, e registro compõem o chamado triângulo central de SOA.

A introdução das ideias de SOC e, mais particularmente, a proposição do conceito de serviço, tem um forte impacto na noção convencional de reuso de software. De fato, quando uma aplicação utiliza um serviço disponível na rede ela não está somente reutilizando um elemento distribuído previamente desenvolvido; ela está também delegando a responsabilidade pelo gerenciamento e execução deste elemento para o provedor do serviço (Di Nitto et al., 2008). Nesse contexto, nem a aplicação que consome o serviço, nem os administradores responsáveis pela sua manutenção tem, normalmente, qualquer controle sobre o serviço utilizado, que pode se tornar indisponível ou ser modificado sem que a aplicação consumidora tenha efetiva ciência. Esse cenário de incerteza demanda uma atenção maior dos desenvolvedores, os quais devem projetar soluções mais robustas e idealmente capazes de se adaptar em tempo de execução.

Em suma, a natureza inerentemente distribuída e o elevado grau de dinamismo presentes nos ambientes de suporte às aplicações baseadas em serviço (*Service-Based Application* (SBA)) demandam uma nova geração de aplicações, as quais devem incorporar características e comportamentos auto-adaptativos (NITTO, 2012), permitindo que essas sejam capazes de selecionar serviços adequados e de se ajustar às variações em seu ambiente sem a necessária intervenção humana.

Embora o padrão arquitetural SOA forneça uma base interessante para o desenvolvimento dessa nova geração de aplicações, fomentando o reuso de unidades funcionais coesas e dinamicamente descobertas, a concepção de aplicações auto-adaptativas representa um enorme desafio. Para que se tenha uma ideia concreta acerca das dificuldades envolvidas no desenvolvimento dessas soluções, pode-se fazer uma breve análise dos aspectos com os quais as aplicações devem tratar ao longo das diferentes fases do seu ciclo de vida, incluindo a descoberta e seleção de serviços, monitoração, e adaptação (KRITIKOS et al., 2013). Em particular, a análise será focada nos atributos de qualidade que são considerados fatores-chave no desenvolvimento de aplicações adaptativas baseadas em serviço.

No contexto de SOC, a execução de uma aplicação é iniciada com a descoberta e seleção dos serviços a serem utilizados. Nessa fase, a aplicação necessita interagir com um registro de forma a descobrir, em tempo de execução, quais serviços fornecem funcionalidades com as características necessárias. Para cada funcionalidade, múltiplos candidatos podem ser descobertos, tornando-se essencial que a aplicação seja capaz de distingui-los e de selecionar aquele que ela considera o "melhor". Dentre os diversos critérios adotados nesse processo de seleção, um amplamente utilizado corresponde ao nível de qualidade de serviço, que é normalmente avaliado através de um conjunto de métricas relacionadas às características (atributos) de qualidade, tais como tempo de resposta e disponibilidade. O desenvolvimento de aplicações com essa capacidade (referenciadas como *QoS-aware Service-Based Application*s (QSBAs)) requer uma definição precisa dos atributos e métricas de qualidade considerados relevantes.

Após a descoberta e seleção, uma aplicação baseada em serviço está pronta para entrar em operação e, eventualmente, prover seus próprios serviços. Contudo, a aplicação não tem nenhum controle sobre os serviços que ela consome. Um exemplo rotineiro dos potenciais problemas dessa falta de controle diz respeito à disponibilidade dinâmica (CERVANTES; HALL, 2004). Basicamente, essa característica indica que os serviços consumidos por uma aplicação podem se tornar disponíveis ou indisponíveis a qualquer momento, independentemente de estarem ou não sendo utilizados por ela. A despeito do exemplo acima ser baseado na característica de disponibilidade, outros atributos de qualidade, tais como tempo de resposta e taxa de requisições processadas, apresentam uma natureza semelhante no que diz respeito à dinamicidade, podendo variar largamente ao longo do tempo.

Um requisito fundamental para que as QSBAs suportem variações dinâmicas no nível de qualidade dos serviços é a utilização de um mecanismo de monitoração que permita a identificação de situações indesejadas. Mais ainda, nessas situações, uma QSBA deve ser capaz de se adaptar dinamicamente, eventualmente substituindo um serviço consumido por outro compatível, obtido através de um novo ciclo de descoberta e seleção. A despeito da breve descrição apresentada, a substituição dinâmica de um serviço em uso é potencialmente complexa, podendo levar a aplicação a um estado inconsistente.

A inclusão do tratamento dos diferentes aspectos relacionados ao dinamismo diretamente no código de uma aplicação apresenta diversos problemas. Em primeiro lugar, as aplicações tornam-se mais complexas e, consequentemente, de difícil manutenção. Em segundo lugar, a solução não é compatível com o princípio de separação de interesses (HURSCH; LOPES, 1995), uma vez que a lógica de negócio se mistura com a lógica responsável pela monitoração e adaptação. Um terceiro ponto a ser destacado diz respeito ao reuso. Ao embutir a lógica responsável pela descoberta e seleção de serviços e pelo processo de adaptação diretamente nas aplicações, essa lógica não é efetivamente reutilizável, sendo necessária a replicação de código quando do desenvolvimento de aplicações semelhantes. Por fim, a solução não é extensível, sendo as aplicações incapazes de utilizar novos atributos de qualidade sem que haja uma interrupção na própria aplicação.

Atentos aos aspectos levantados, diferentes projetos propõem que as QSBAs sejam desenvolvidas utilizando-se modelos de componentes baseado em serviços (ESCOFFIER; HALL; LALANDA, 2007; WALLS, 2009; SEINTURIER et al., 2012). A ideia central consiste em fazer com que essas aplicações sejam gerenciadas por contêineres embutidos em infraestruturas orientadas a serviços. No contexto das QSBAs, esses contêineres seriam responsáveis por resolver as dependências das aplicações em termos dos recursos e serviços por elas requeridos.

Em um ambiente ideal, além de descobrir e selecionar os serviços requeridos por uma aplicação em tempo de instalação, os contêineres devem ser capazes de tratar os aspectos dinâmicos inerentes aos ambientes SOA. Em particular, um contêiner deve monitorar os serviços consumidos por uma aplicação e adaptá-la dinamicamente quando a qualidade desses serviços não atender aos níveis requeridos. Essa adaptação deve ser transparente do ponto de vista do código de negócio (e.g., tratada através de mecanismos de injeção de dependência). Neste cenário ideal, o conjunto formado pelo contêiner e pelo código de negócio compõe uma aplicação auto-adaptativa baseada em serviços e ciente de qualidade (i.e., QSBA).

Para desempenhar as atividades descritas acima, os contêineres precisam de diversas informações, tais como: Quais serviços uma aplicação fornece e/ou requer? Como os serviços necessários são selecionados? Quais atributos de qualidade são relevantes? Como esses atributos podem ser descritos, monitorados e avaliados? Como e quando uma aplicação baseada em serviços deve ser adaptada? Uma plataforma que se proponha a suportar o desenvolvimento e execução de QSBAs deve oferecer uma resposta para cada uma dessas perguntas.

1.2 PROBLEMA

A despeito dos avanços obtidos nas últimas duas décadas, o desenvolvimento de aplicações auto-adaptativas ainda representa um enorme desafio (CHENG et al., 2009; LEMOS et al., 2013). Grande parte dessa dificuldade decorre da necessidade de tomar decisões em tempo

de execução. De fato, para que estas decisões possam ser tomadas, é necessário que se tenha à disposição um amplo conjunto de informações, compreendendo desde requisitos funcionais e não-funcionais, até informações concernentes ao estado atual da aplicação e do seu ambiente. Embora uma parte dessas informações possa ser identificada ainda em tempo de desenvolvimento, outra parte somente está disponível ao longo da execução, sendo essencial definir como ela será representada e mantida na plataforma de execução.

Nesse sentido, uma abordagem promissora consiste na utilização de modelos em tempo de execução (FRANCE; RUMPE, 2007; BLAIR; BENCOMO; FRANCE, 2009) que propõe que as informações sejam organizadas através de um conjunto de modelos, representando as diferentes visões necessárias para suportar a adaptação dinâmica. Esses modelos devem evoluir com as mudanças percebidas na aplicação e em seu ambiente, garantindo que as informações representadas estejam devidamente atualizadas. Mais ainda, os modelos em tempo de execução usualmente mantém uma conexão causal com a aplicação que eles representam, de forma que modificações nesses modelos se refletem na aplicação em execução e vice-versa.

No contexto das QSBAs, os modelos normalmente utilizados representam conceitos pertencentes a dois domínios: serviços e qualidade. De fato, uma QSBA é essencialmente uma composição de serviços, sendo natural a sua representação através de um modelo que explicita os componentes que constituem a aplicação, e os serviços consumidos e providos por esses componentes. Os modelos assim estabelecidos podem ser utilizados tanto para representar a configuração atual da aplicação, quanto para promover adaptações a partir da substituição dos serviços utilizados e da conexão causal existente entre esses modelos e a aplicação em execução.

Para que as aplicações sejam cientes de qualidade, o nível de qualidade dos serviços também deve ser representado nos modelos, sendo essencial a utilização de um modelo de qualidade para este fim. Em essência, um modelo de qualidade estabelece uma taxonomia que representa os atributos e métricas que podem ser utilizados pelas aplicações para especificar restrições no nível de qualidade esperado de cada serviço.

Embora sejam essenciais, os conceitos e modelos descritos não são suficientes para representar os aspectos dinâmicos relacionados à própria natureza das QSBAs. Nesse contexto, um problema está relacionado à adoção de um modelo de qualidade específico. A despeito da ampla variedade de modelos de qualidade propostos na literatura (KRITIKOS et al., 2013; ORIOL; MARCO; FRANCH, 2014), nenhum deles se estabeleceu como padrão de fato.

Dentre as limitações frequentemente observadas nos modelos propostos, destaca-se a falta de extensibilidade. De fato, a maioria desses modelos se limita a estabelecer uma taxonomia de atributos e métricas de qualidade previamente determinados, não sendo possível a definição de novas características de qualidade relevantes no contexto de uma aplicação em particular.

Observe-se também que os modelos de qualidade normalmente não explicitam a forma como as métricas podem ser computadas, ficando a sua utilidade limitada a permitir a expressão de restrições que somente podem ser utilizadas no processo de seleção "estática" de serviços. Em outras palavras, não há nesses modelos informação suficiente para que uma plataforma de execução saiba como monitorar os serviços utilizados e verificar se esses estão mantendo o nível de qualidade esperado.

Do ponto de vista da capacidade de adaptação, as plataformas atuais também apresentam importantes limitações. Em geral, embora os contêineres embutidos nessas plataformas sejam configuráveis durante a instalação das aplicações, poucos suportam a adaptação dinâmica. Quando esse suporte existe, os gatilhos utilizados no disparo das regras de adaptação são predefinidos e normalmente baseados em disponibilidade (ESCOFFIER; HALL; LALANDA, 2007; WALLS, 2009).

Em geral, nas aplicações auto-adaptativas, o processo de adaptação é disparado a partir da ocorrência de eventos detectados durante a etapa de monitoração. Após a detecção, esses eventos são analisados e, caso haja necessidade, a aplicação é adaptada. Nas plataformas atuais, os desenvolvedores ficam usualmente limitados a especificar esse processo de adaptação em termos de eventos simples, os quais são produzidos diretamente por sensores responsáveis por monitorar a aplicação e/ou o seu ambiente de execução. Neste contexto, para que eventos complexos (gerados a partir de combinações de outros eventos) possam ser utilizados, eles devem ser gerados a partir de código embutido na própria aplicação, tornando o código mais complexo e as possibilidades de reuso mais restritas. Essa abordagem não oferece um nível de abstração adequado, uma vez que um desenvolvedor não é capaz de especificar, em alto nível, eventos complexos potencialmente utilizados como gatilhos do processo de adaptação.

Em suma, os modelos normalmente utilizados nas plataformas de suporte às aplicações auto-adaptativas atuais possuem importantes limitações uma vez que: pré-determinam o universo das características de qualidade que podem ser representadas, não permitem a especificação do processo de computação das métricas de qualidade, e não suportam a definição de eventos complexos.

1.3 CONTRIBUIÇÃO

O contexto descrito representa o pano de fundo da presente tese de doutorado, a qual tem a intenção de contribuir com o estado da arte, propondo a inclusão dos conceitos pertencentes ao domínio de eventos no espaço de modelagem das aplicações auto-adaptativas. A motivação para essa proposta partiu da percepção de que o domínio de eventos é central no contexto das QSBAs, permeando todo o ciclo de monitoração e adaptação dessas aplicações.

A despeito da relevância do domínio de eventos, as soluções atuais não representam esses elementos nos modelos mantidos em tempo de execução. Como veremos, a mode-

lagem conjunta de conceitos dos domínios de serviços, qualidade, e eventos permite uma representação mais completa das aplicações auto-adaptativas, tornando os eventos cidadãos de primeira classe. As ideias propostas no âmbito desta tese foram materializadas em uma nova plataforma de suporte às aplicações auto-adaptativas, referenciada como *Dynamic Service Oriented Architecture* (DSOA) (SOUZA; SILVA; ROSA, 2017), que está fundamentada nas ideias de sistemas autonômicos e possui como elemento central um laço de controle que engloba mecanismos de injeção de dependências, processamento de eventos complexos, e suporte a modelos em tempo de execução.

Em particular, os modelos utilizados na plataforma DSOA permitem, além da definição de novos tipos de eventos, a especificação de agentes, representando regras de processamento responsáveis pela geração de eventos complexos. A introdução do conceito de eventos e dos agentes de processamento em nível de modelo permite que os desenvolvedores expressem, em alto nível, quais são os eventos relevantes e como estes podem ser processados. Mais ainda, uma vez que estes modelos são levados para o tempo de execução, eles podem ser dinamicamente modificados, promovendo uma adaptação nos próprios mecanismos de monitoração e adaptação.

Outro aspecto importante decorrente da representação dos eventos em nível de modelo é a flexibilização do ponto de vista da definição de métricas de qualidade, as quais são utilizadas como critério de adaptação. Na plataforma DSOA, todas as métricas de qualidade são definidas em função dos eventos que ocorrem em tempo de execução. Uma vez que estes eventos são definidos pelos desenvolvedores e representados através de modelos, novas métricas podem ser definidas e associadas a eles. Mais ainda, uma vez que a plataforma permite definir, nos modelos, a forma como os eventos são processados, o próprio "algoritmo" de computação das métricas pode ser definido pelos desenvolvedores de aplicação. Por fim, como estes modelos são mantidos durante a execução, as métricas podem ser redefinidas e os seus algoritmos de computação dinamicamente modificados.

Além da contribuição conceitual mencionada acima, as seguintes contribuições específicas podem ser elencadas:

1. A extensão de um modelo de componentes baseado em serviços, tornando-o capaz de suportar a composição dinâmica em função de atributos e métricas de qualidade definidos pelas aplicações;

2. A proposição de uma nova plataforma (SOUZA et al., 2011; CAVALCANTI; SOUZA; ROSA, 2013; SOUZA et al., 2014; SOUZA; SILVA; ROSA, 2017) capaz de suportar aplicações auto-adaptativas definidas com base no modelo de componentes DSOA. Além de permitir a adaptação das aplicações em execução, a própria plataforma pode ser dinamicamente reconfigurada, permitindo a troca das diferentes políticas utilizadas, tais como política de seleção de serviços e a política de instanciação de componentes;

3. A proposição de um conjunto de *Domain-Specific Language* - Linguagem Específica de Domínio (DSL) integradas (SOUZA et al., 2015) para suportar a concepção de modelos capazes de representar as diferentes visões identificadas como necessárias para a modelagem de uma QSBA. Dentre esses modelos, destacam-se o modelo de qualidade, o modelo de componentes e serviços, o modelo de eventos e o modelo de monitoração.

1.4 ESTRUTURA DA TESE

A presente tese de doutorado está organizada em três partes. A Parte I visa contextualizar o trabalho desenvolvido, apresentando a sua fundamentação teórica e as pesquisas que compõem o estado da arte.

- Seção 2: discorre acerca dos sistemas adaptativos e auto-adaptativos, apresentando os principais elementos conceituais e as abordagens utilizadas na engenharia desses sistemas.
- Capítulo 3: organiza os elementos que devem constituir uma plataforma genérica de suporte às aplicações auto-adaptativas.
- Capítulo 4: apresenta uma coletânea de trabalhos relacionados, visando fornecer uma análise crítica do estado da arte.

A Parte II compreende a contribuição dessa tese, e está organizada em quatro capítulos:

- Capítulo 5: apresenta uma visão geral da plataforma, ressaltando os objetivos que motivaram a sua proposição.
- Capítulo 6: descreve os modelos utilizados na plataforma DSOA e o meta-modelo que define os elementos conceituais que compõem uma QSBA em execução.
- Capítulo 7: detalha os elementos que compõem a implementação de referência da plataforma DSOA.
- Capítulo 8: apresenta os experimentos que foram realizados para fins de avaliação da solução proposta.

Por fim, a Parte III é composta pelo Capítulo 9 que apresenta as conclusões e os trabalhos futuros que podem ser desenvolvidos visando estender as características atuais da plataforma.

2 APLICAÇÕES ADAPTATIVAS E AUTO-ADAPTATIVAS

> *A diferença entre passado, presente e futuro é apenas uma persistente ilusão.*
>
> Albert Einstein,

Este capítulo tem como objetivo apresentar os principais conceitos relacionados à tese com o intuito de oferecer a fundamentação teórica necessária ao entendimento da mesma e dos avanços que esta representa com relação ao estado da arte. Nesse sentido, o capítulo se inicia destacando a relevância da adaptação no contexto das aplicações atuais e apresentando as técnicas e tecnologias que podem ser utilizadas para viabilizar a adaptação dinâmica, i.e. em tempo de execução. Na sequência, o capítulo discute as consequências de delegar para as próprias aplicações a responsabilidade pela condução do processo de adaptação, dando origem às denominadas aplicações auto-adaptativas, que são centrais para o presente trabalho.

2.1 INTRODUÇÃO

No mundo atual, há uma demanda crescente por aplicações capazes de executar em e interagir com ambientes cada vez mais dinâmicos formados por dispositivos "inteligentes", compondo um cenário referenciado muitas vezes como Internet das Coisas (IoT). A natureza volátil, distribuída e heterogênea desses ambientes requer que as aplicações sejam capazes de ajustar seu comportamento, em tempo de execução, em função de mudanças percebidas na própria aplicação ou em seu ambiente. Aplicações com essa capacidade são referenciadas na literatura como aplicações auto-adaptativas e vêm sendo alvo de importantes pesquisas ao longo das últimas décadas (SALEHIE; TAHVILDARI, 2009; CHENG et al., 2009; LEMOS et al., 2013; MACíAS-ESCRIVá et al., 2013; KRUPITZER et al., 2015).

Apesar dessa demanda, projetar aplicações auto-adaptativas ainda representa um enorme desafio. Para compreendermos as dificuldades envolvidas nesse processo, o primeiro passo consiste em conhecer as abordagens conceituais que podem ser utilizadas para promover a adaptação dinâmica das aplicações. O passo seguinte envolve entender como essas abordagens podem ser realizadas utilizando-se o arcabouço tecnológico disponível atualmente. Por fim, é importante compreender as consequências da decisão de delegar para a própria aplicação a responsabilidade pela condução do processo de adaptação. Esses conceitos são explorados ao longo do capítulo enquanto discutimos as aplicações adaptativas e auto-adaptativas.

2.2 APLICAÇÕES ADAPTATIVAS

Segundo Bruni et al. (2012), uma aplicação adaptativa é caracterizada por possuir (meta-)dados de controle, os quais podem ser modificados, em tempo de execução, para adaptar o comportamento da aplicação. Em princípio, a condução do processo de adaptação pode ser realizada por diferentes agentes, sejam estes externos ou internos à aplicação.

Em um cenário típico, a aplicação adaptativa disponibiliza acesso a seus (meta-)dados de controle através de uma *Application Programming Interface* (API), a qual é utilizada por *scripts* de configuração produzidos e executados por um administrador humano, responsável por identificar a necessidade de adaptação e por realizar as ações necessárias.

Para entendermos em que consistem os (meta-)dados de controle e ações de adaptação, devemos inicialmente compreender as abordagens comumente utilizadas na realização da adaptação: adaptação paramétrica e adaptação composicional (MCKINLEY et al., 2004).

2.2.1 Adaptação Paramétrica e Composicional

A adaptação paramétrica consiste na utilização de parâmetros, representados através de (meta-)dados de controle, que podem ser modificados de forma a promover uma alteração no comportamento de uma aplicação. Uma limitação importante concernente ao uso de adaptação paramétrica, é que a aplicação adaptável já deve ser concebida de forma a ser capaz de interpretar os diferentes valores assumidos pelos parâmetros. Desta forma, a utilização da adaptação paramétrica está normalmente relacionada ao ajuste de comportamentos previamente definidos.

Para que uma aplicação possa incorporar novos comportamentos é necessária a capacidade de realizar uma adaptação composicional, que consiste na recomposição dinâmica de uma aplicação através da adição, remoção, substituição, e/ou reconexão de elementos. No centro dessa abordagem está a visão de uma aplicação como uma composição de unidades elementares de reuso, fortemente coesas e fracamente acopladas. Por um lado, elementos fortemente coesos tratam de aspectos bem identificados e definidos, o que permite que a adaptação tenha um efeito localizado. Por outro, o fraco acoplamento viabiliza a substituição desses elementos em tempo de execução.

O nível de flexibilidade obtido a partir da utilização da abordagem composicional vem fazendo com que esta seja utilizada como alicerce de diversas pesquisas relevantes (OREIZY et al., 1999; GARLAN et al., 2004; FLOCH et al., 2006; KRAMER; MAGEE, 2007; HALLSTEINSEN et al., 2012; HUBER et al., 2017), as quais consideram a adaptação sob uma ótica essencialmente arquitetural.

Para viabilizar a utilização da adaptação composicional, é necessário definir a unidade elementar de reuso e a forma pela qual diferentes unidades podem ser conectadas. Neste contexto particular, dois paradigmas de desenvolvimento distintos, eventualmente vistos como complementares, se destacam: (a) Desenvolvimento Baseado em Componentes e (b)

Computação Orientada a Serviços.

2.2.2 Desenvolvimento Baseado em Componentes

O paradigma de desenvolvimento baseado em componentes (*Component-Based Development* (CBD)) (SZYPERSKI, 1998; HEINEMAN; COUNCILL, 2001) foi fortemente impulsionado pelas noções de separação de interesses (HURSCH; LOPES, 1995) e reuso. A essência desse paradigma está na ideia de que o desenvolvimento de componentes individuais e a montagem de aplicações podem ser realizados separadamente, por perfis distintos, e em lugares diferentes (CERVANTES; HALL, 2004).

Sob a ótica da separação de interesses, a proposição de CBD objetiva viabilizar a separação entre lógica de negócio e lógica não-funcional, transferindo a responsabilidade por esta última, dos componentes de negócio, para os ambientes de execução. De fato, antes mesmo da proposição de CBD, algumas plataformas de *middleware* tradicionais (e.g., CORBA (GROUP, 2006)) já possuíam mecanismos capazes de tratar aspectos como concorrência, transação, e segurança; contudo, a utilização desses mecanismos era normalmente realizada de forma programática, promovendo uma mistura entre código de negócio e aspectos não-funcionais. A experiência com essas plataformas levou os desenvolvedores a uma constatação importante: o código responsável pelos aspectos não-funcionais é, em geral, mais complexo que o código de negócio, sem, contudo, agregar valor do ponto de vista do domínio da aplicação. Essa percepção ressaltou a necessidade de mecanismos que permitissem extrair das aplicações a lógica relativa ao tratamento dos aspectos não-funcionais, simplificando as atividades de desenvolvimento e de manutenção.

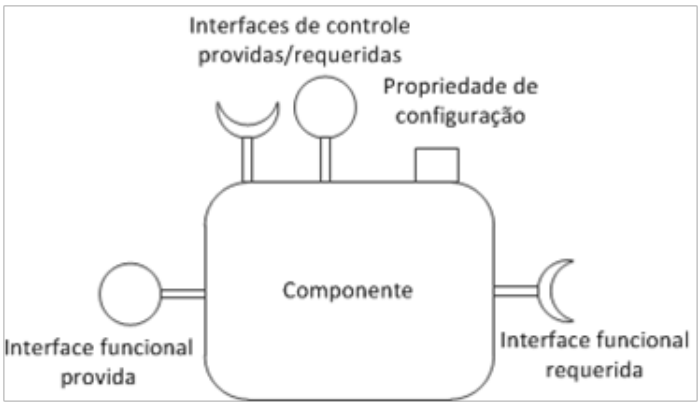
Do ponto de vista do reuso, a proposta de CBD visa permitir a criação de componentes fortemente coesos e, tipicamente, de maior granularidade do que é usual na abordagem de orientação a objetos. Toda interação com os componentes ocorre através de interfaces bem-definidas, enfatizando-se o papel da composição como mecanismo de construção de aplicações, e viabilizando a utilização desses elementos como unidades básicas de reuso (SZYPERSKI, 1998).

2.2.2.1 Componente

A despeito do papel central dos componentes no contexto de CBD, não há uma definição padrão para o conceito. Dentre as definições de componentes referenciadas na literatura, destacamos Szyperski (1998): “um componente de software é uma unidade de composição com interfaces contratualmente especificadas e que possui somente dependências explícitas em seu contexto. Um componente de software pode ser instalado independentemente e está sujeito à composição por parte de terceiros”. Nessa definição destacam-se os conceitos de interface e composição.

As interfaces definem os pontos de acesso de um componente, sendo sempre declaradas de forma explícita. Cada interface possui um tipo, que define o conjunto de operações

Figura 1 – Representação conceitual de um componente



Fonte: Cervantes (2004)

associado à mesma, e uma direção, que define se a interface é provida ou requerida pelo componente. Neste contexto, cada interface requerida representa uma dependência do componente, a qual deve ser conectada à uma interface provida de mesmo tipo (ou de um subtipo) para que possa ser satisfeita. As interfaces podem ser classificadas como: (1) funcionais, utilizadas para representar funcionalidades em nível de negócio; ou (2) de controle, utilizadas para representar as interações entre um componente e seu ambiente de execução, sendo essenciais para as atividades de gerenciamento.

Embora as interfaces definam "quais" funcionalidades um componente provê e requer, elas não especificam "como" o componente realiza tais funcionalidades. De fato, a lógica associada a um componente é definida através da especificação de sua implementação. Essa separação explícita entre interface e implementação favorece o reuso e permite que a implementação possa evoluir sem implicar em impactos externos.

De forma similar à abordagem de orientação a objeto, o paradigma de CBD também enfatiza a separação entre as noções de classe e de instância. Neste contexto, interfaces e implementação fazem, na verdade, parte da definição da classe de um componente. Em tempo de execução, o comportamento de um componente é realizado por suas instâncias, as quais são, em geral, obtidas a partir de fábricas criadas durante o processo de instalação do componente em um ambiente de execução. Para que essas instâncias possam ser individualmente configuradas, a definição de um componente também pode incluir a especificação de um conjunto de propriedades. Diferentes plataformas baseadas em componentes permitem a modificação dos valores dessas propriedades em tempo de execução. Essa modificação dinâmica de propriedades viabiliza a adaptação paramétrica das aplicações. A **Figura 1** ilustra os elementos que tipicamente constituem uma classe de componente.

Por fim, é necessário entender como ocorre a integração entre a lógica funcional realizada pelos componentes e a lógica não-funcional realizada pelos elementos do ambiente de execução. Para que a lógica não-funcional seja transferida dos componentes para um

ambiente de execução, ela deve ser implementada de uma forma genérica, sendo o seu comportamento configurável de acordo com as necessidades individuais dos componentes. Por exemplo, para que a lógica transacional seja implementada por um ambiente de execução, ela deve genérica, permitindo, assim, que cada componente possa informar onde uma transação distinta deve ser iniciada e concluída. Neste contexto, para que um desenvolvedor de componente possa configurar a lógica não-funcional genérica, a definição de um componente deve incluir um conjunto de meta-dados (anotações) (VOLTER; SCHMID; WOLFF, 2002), os quais devem ser empacotados juntamente com a implementação. Durante a instalação do componente, esses meta-dados são interpretados.

2.2.2.2 Composição

Em CBD, cada componente representa explicitamente suas dependências. Essa representação explícita permite que a montagem de aplicações seja realizada fora do código dos componentes, através da utilização de conectores interligando esses elementos. A extração do mecanismo de conexão de dentro do código dos componentes, maximiza seu potencial de reuso e provê uma maior visibilidade à arquitetura das aplicações. De fato, componentes e conectores são elementos fundamentais na representação de uma arquitetura de software (ALLEN; GARLAN, 1994), ressaltando a forte relação existente entre essa área e a abordagem proposta por CBD.

Uma composição é, essencialmente, uma descrição de como os componentes de uma aplicação devem ser interligados, podendo ser realizada de diferentes formas. Uma primeira possibilidade consiste em descrever as composições diretamente em linguagens imperativas (de uso geral ou de *script*). A despeito da viabilidade técnica dessa solução, ela vai de encontro às abordagens que há muito enfatizam as diferenças entre as linguagens utilizadas na programação de unidades elementares e aquelas utilizadas na composição dessas unidades (DEREMER; KRON, 1976; ALLEN; GARLAN, 1994). Com base nessas ideias, é comum a realização de composições através de linguagens declarativas, normalmente sob forma de *Architecture Description Language*s (ADLs).

Diversas plataformas baseadas em componentes permitem a modificação dinâmica das composições (ESCOFFIER; HALL; LALANDA, 2007; BRUNETON, 2009; SEINTURIER et al., 2012; HNETYNKA; PLASIL, 2011), a qual pode ser realizada através da inclusão, remoção e/ou substituição dos componentes. Essa capacidade caracteriza a adaptação composicional, descrita na Seção 2.2.1, sendo fundamental para a construção de plataformas de suporte às aplicações auto-adaptativas.

Por fim, é importante destacar que diferentes plataformas implementam suporte aos conceitos de componente e composição, cada uma estabelecendo suas próprias regras acerca de como os componentes podem ser definidos e compostos. O conjunto dessas regras corresponde ao conceito de *modelo de componentes* (HEINEMAN; COUNCILL, 2001; LAU; WANG, 2007; CRNKOVIC et al., 2011).

2.2.2.3 Modelo de Componentes

Segundo Crnkovic et al. (2011), um modelo de componentes define padrões para (1) as propriedades que os componentes individuais devem satisfazer, e (2) os métodos e mecanismos que podem ser utilizados na composição desses elementos. Uma definição mais precisa é apresentada por Lau e Wang (2007): "um modelo de componentes de software é uma definição da semântica dos componentes, da sintaxe dos componentes, e das regras de composição". Como se pode perceber, os conceitos de componente e composição devem sempre ser interpretados no contexto de um modelo de componente específico.

No cerne de um modelo de componente está um *modelo abstrato* que define os elementos conceituais suportados pelo modelo (ROUVOY; MERLE, 2009) (e.g., componente, interface, e conectores). Para que esses elementos conceituais possam ser utilizados no desenvolvimento de aplicações, é necessário que eles sejam representados em alguma linguagem de programação. Essa representação é materializada no conceito de um *modelo de programação*, o qual estabelece uma API, encapsulando aspectos técnicos necessários para suportar as abstrações que formam o modelo de componentes. Dentre os elementos que compõem essa API estão as interfaces de controle, padronizando, assim, as interações entre os componentes e o ambiente de execução. Em geral, um *modelo de programação* impõe, também, um conjunto de restrições de implementação (VOLTER; SCHMID; WOLFF, 2002) que estabelece os recursos que não devem ser utilizados diretamente pelos desenvolvedores (e.g., manipulação explícita de *threads*). A **Figura 2** apresenta a implementação de um componente com base no modelo Fractal e destaca alguns elementos que compõem o modelo de programação Java correspondente (BRUNETON, 2009).

Em particular, a **Figura 2** apresenta a implementação do componente *RequestDispatcher*, o qual implementa uma interface de negócio e outra interface definida pelo modelo de programação Fractal, promovendo uma mistura entre código de negócio e código associado ao modelo de componentes. Essa abordagem torna a implementação específica de plataforma e dificulta a portabilidade da aplicação para outros modelos de componentes. Consequentemente, um aspecto importante a ser considerado quando adota-se um modelo de componentes é a quantidade de código necessária para a utilização da API imposta pelo modelo em questão e a forma como o seu uso está atrelado ao código de negócio. Em um cenário ideal, o desenvolvedor de aplicações deve ficar centrado no domínio da aplicação e o código necessário para a manipulação dos aspectos técnicos relativos ao modelo de componentes deve ser mínimo.

Embora as definições apresentadas acima compreendam as ideias fundamentais concernentes ao conceito de modelo de componentes, elas não explicitam alguns aspectos relevantes no contexto do presente trabalho. O primeiro diz respeito aos atributos não-funcionais. De fato, todos os modelos de componentes ressaltam a necessidade de permitir que esses elementos expressem, de forma explícita, as funcionalidades por eles providas e/ou requeridas. Contudo, a possibilidade de especificação de atributos não-funcionais é

Figura 2 – Modelo de componentes Fractal

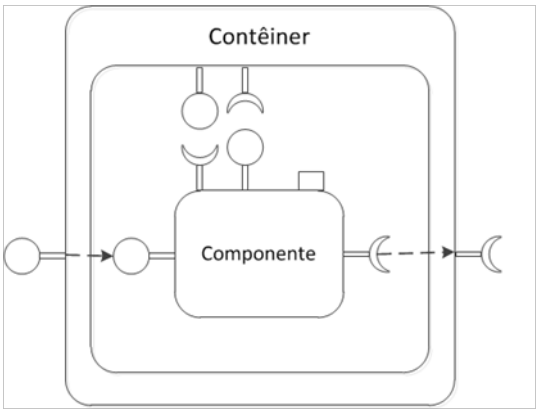
```
/*
    RequestDispacher representa a implementação de um componente
    Fractal. Esse componente implementa uma interface de "negócio"
    (RequestHandler) e uma interface relacionada ao modelo de
    componentes Fractal (BindingController), ressaltando a mistura
    entre código de negócio e código relacionado ao modelo de
    programação correspondente.
*/
public class RequestDispatcher implements RequestHandler,
BindingController {
  private Map handlers = new TreeMap();
  // Métodos associados ao Modelo de Programação Fractal
  public String[] listFc () {
    return (String[])handlers.keySet().toArray(new String[handlers.
    size()]);
  }
  public Object lookupFc (String itfName) {
    if (itfName.startsWith("h")) { return handlers.get(itfName); }
    else return null;
  }
  public void bindFc (String itfName, Object itfValue) {
    if (itfName.startsWith("h")) { handlers.put(itfName, itfValue);
     }
  }
  public void unbindFc (String itfName) {
    if (itfName.startsWith("h")) { handlers.remove(itfName); }
  }
  // Método associado ao domínio de negócio
  public void handleRequest (Request r) throws IOException { ... }
}
```

Fonte: baseada em http://fractal.ow2.org/tutorial/

bem menos comum (CRNKOVIC et al., 2011). Em geral, quando essa especificação é possível, ela ocorre através de interfaces estendidas ou de meta-dados. Nesse contexto, um aspecto importante diz respeito ao conjunto de atributos não-funcionais compreendido por um modelo particular e à capacidade de extensão desse conjunto pelos desenvolvedores de aplicação. Esse é um aspecto central no contexto da presente tese.

Por fim, outro aspecto relevante não destacado está relacionado à capacidade de execução das aplicações construídas com base no modelo de componentes. Como ressaltado por (LAU; WANG, 2007), nem todos os modelos de componentes são executáveis. De fato, para que as aplicações baseadas em um modelo de componentes possam ser executadas, é essencial a disponibilização de um ambiente de execução capaz de entender os conceitos associados a esse modelo.

Figura 3 – Contêiner



Fonte: Cervantes (2004)

2.2.2.4 Ambientes de Execução

Os ambientes de execução são responsáveis por "dar vida" aos componentes. Em geral, para que os componentes possam ser utilizados, eles devem ser empacotados e instalados nesses ambientes. Mais ainda, diversos modelos tratam o conceito de componente de forma similar ao conceito de classes na orientação a objetos, sendo necessária a instanciação para que esses elementos possam ser executados e utilizados para compor aplicações. Neste contexto, é comum que o processo de instanciação seja realizado através do uso de fábricas, as quais, além de permitir um desacoplamento entre os clientes e a implementação do componente em si, viabilizam a utilização de diferentes políticas de instanciação (CERVANTES; HALL, 2004).

Em tempo de execução, as instâncias de componentes devem ser gerenciadas, cabendo essa responsabilidade a um elemento central no CBD, denominado contêiner (CONAN et al., 2001; VOLTER; SCHMID; WOLFF, 2002). Conceitualmente, um contêiner envolve uma instância de um componente, intermediando as interações entre ela e os demais elementos que compõem seu ambiente de execução (vide **Figura 3**). A concepção de um contêiner como um mediador introduz um nível de indireção que é essencial para a implementação da adaptação composicional, e posiciona a orientação a componentes como uma tecnologia importante no contexto das aplicações auto-adaptativas (MCKINLEY et al., 2004).

A capacidade de intermediação também posiciona o contêiner como um elemento central do ponto de vista do gerenciamento dos aspectos não-funcionais (CONAN et al., 2001). De fato, ao interceptar os acessos a um componente, o contêiner tem a oportunidade de acionar serviços técnicos (também referenciados como serviços de suporte), delegando a eles a responsabilidade pelo tratamento desses aspectos. Para que esses serviços técnicos possam ser úteis em diferentes cenários, eles são projetados em termos genéricos, devendo ser configurados para que possam atender às necessidades específicas de cada componente. Neste contexto, cabe ao contêiner a responsabilidade por essa configuração, que deve ser

realizada com base em um conjunto de meta-dados informados pelos desenvolvedores.

Para manter os meta-dados separados da lógica de negócio é comum a utilização da programação orientada a atributos (CONAN et al., 2001; WADA; SUZUKI; OBA, 2005; ROUVOY; MERLE, 2009). Em geral, essa técnica se baseia na utilização de arquivos de configuração (também referenciados como descritores de instalação) ou de anotações no próprio código fonte para associar meta-dados aos componentes correspondentes. Por fim, os meta-dados (empacotados juntamente com os componentes da aplicação) são processados pelo contêiner durante o processo de instalação. É importante destacar que o uso dessa abordagem permite que os requisitos não-funcionais sejam tratados de forma não intrusiva e que a implementação dos componentes seja centrada na lógica de negócio.

Em síntese, o desenvolvimento de um componente compreende, além da definição de sua implementação, a especificação explícita de um conjunto de meta-dados, descrevendo as funcionalidades providas e requeridas pelo componente, as suas propriedades de configuração, e as informações necessárias à configuração dos requisitos não-funcionais. Para que os componentes possam ser executados, os meta-dados e o código dos componentes devem ser empacotados e instalados em um ambiente de execução. Esse ambiente é composto por contêineres, responsáveis pelo gerenciamento de instâncias dos componentes, e por serviços técnicos, que são acionados pelos contêineres para tratar os aspectos não-funcionais.

2.2.3 Computação Orientada a Serviço

Um outro paradigma que pode ser utilizado para montar aplicações a partir de unidades elementares de reuso é a orientação a serviços (PAPAZOGLOU et al., 2007). A introdução desse paradigma teve um forte impacto na perspectiva sob a qual o reuso dos elementos de software era tradicionalmente considerado. No centro dessa nova perspectiva está a noção de "mundo aberto" (BARESI; Di Nitto; GHEZI, 2006; Di Nitto et al., 2008).

Segundo Baresi, Di Nitto e Ghezi (2006), todo o desenvolvimento de software tradicional estava fundamentado na presunção de que as fronteiras entre os sistemas e os ambientes correspondentes eram conhecidas e estáveis. Assim, um levantamento cuidadoso dos requisitos seria capaz de capturar todas as características necessárias e identificar os elementos que compõem os ambientes de execução. Os sistema desenvolvidos com base nessa perspectiva de "mundo fechado" não eram capazes de se adaptar às mudanças que não haviam sido previstas quando do seu projeto original. Como veremos, a proposição de SOC teve um impacto direto nesse cenário, viabilizando a construção de aplicações mais dinâmicas e flexíveis baseadas na composição de elementos auto-contidos e reusáveis, referenciados como serviços.

2.2.3.1 Serviço

Um serviço (CERVANTES; HALL, 2004; PAPAZOGLOU et al., 2007) é uma funcionalidade autônoma, coesa, e reusável, descrita de forma contratual através de um *descritor de serviço*. Esse descritor especifica um serviço através de sua interface funcional, e pode conter, também, informações referentes à semântica, ao comportamento, e ao nível de qualidade.

Em SOC, os serviços podem ser dinamicamente descobertos e invocados em função do estilo arquitetural SOA, o qual especifica as interações entre três papeis distintos: provedor, consumidor e registro de serviços. Os provedores publicam descritores de serviços em um registro, que é utilizado pelos consumidores para buscar serviços com as características necessárias. Ao receber a lista de serviços candidatos, o consumidor deve selecionar aquele que considera mais adequado e ligar-se a ele para que possa realizar requisições.

O conceito de serviço e o estilo arquitetural SOA posicionam a orientação a serviços como um paradigma natural para a construção de plataformas com suporte aos sistemas auto-adaptativos. De fato, por serem fortemente coesos, os serviços podem ser facilmente isolados, servindo para estabelecer fronteiras entre funcionalidades distintas. Além disso, o baixo acoplamento, decorrente da capacidade de descoberta, seleção, e ligação dinâmica, facilita a substituição dos serviços utilizados por uma aplicação, viabilizando a realização da adaptação composicional.

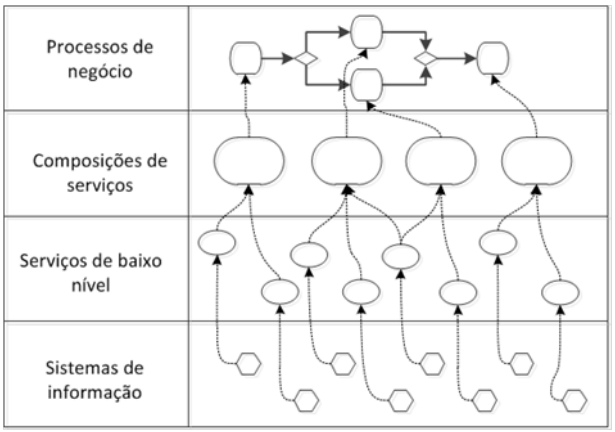
A possibilidade de utilização de serviços disponibilizados por terceiros introduziu um aspecto completamente novo no contexto do reuso. De fato, ao reusar um serviço de software fornecido por um terceiro, uma aplicação não está somente fazendo uso de um componente de negócio desenvolvido e mantido por esse terceiro, mas sim delegando para ele a responsabilidade pela execução e gerenciamento de parte da aplicação, uma vez que ela não tem nenhum controle sobre o ambiente no qual o serviço está em execução (Di Nitto et al., 2008).

Por fim, é importante ressaltar que a essência de SOC, materializada nas definições de serviço e de SOA, não envolve nenhum aspecto tecnológico, de forma que a orientação a serviços pode ser implementada utilizando-se diferentes tecnologias.

2.2.3.2 Composição de Serviços

Uma composição de serviços é, em essência, um programa que realiza a sua funcionalidade através do consumo de um conjunto de serviços. Dada a sua natureza programática, uma composição deve possuir um fluxo de controle que é responsável por realizar a coordenação da invocação dos serviços e a transferência de dados entre eles (CERVANTES; HALL, 2004). A natureza agnóstica inerente ao estilo arquitetural SOA viabiliza a utilização de diferentes linguagens para descrever o fluxo de controle de uma composição, desde linguagens de uso geral, até linguagens próprias para descrição de fluxos de trabalho (i.e.,

Figura 4 – Níveis de composição de serviços



Fonte: baseada em Davis (2009)

workflows).

Em SOC, serviços e composições são tipicamente organizados em níveis (vide **Figura 4**). O nível mais baixo apresentado na figura é composto pelos sistemas de informação, os quais realizam as funcionalidades de negócio disponíveis. Essas funcionalidades são tipicamente expostas por serviços primitivos de baixa granularidade, os quais são referenciados na figura como "serviços de baixo nível". Esses serviços são utilizados em composições que oferecem serviços de maior granularidade. Por fim, os serviços primitivos e as composições podem ser utilizados na montagem de processos de negócio, os quais são composições de mais alto nível, normalmente caracterizadas por execuções longas e estados de espera (DAVIS, 2009).

Independentemente do nível e da linguagem utilizada, uma composição de serviços deve ser descrita em termos de referências para interfaces dos serviços requeridos, as quais somente devem ser resolvidas em tempo de execução, quando os provedores de serviços são dinamicamente descobertos e ligados às respectivas referências. Neste contexto, a composição deve tratar de diversas particularidades próprias da abordagem orientada a serviços, tais como a disponibilidade dinâmica (CERVANTES; HALL, 2004), e a filtragem e seleção dos serviços candidatos utilizados pela composição.

2.2.3.3 Ambiente de Execução

Para que serviços e composições possam ser utilizados, eles devem ser instalados em uma infraestrutura de suporte, referenciada como ambiente de execução. Segundo (PAPAZOGLOU et al., 2007), os ambientes de execução devem possuir mecanismos capazes de suportar: (1) a interação entre provedores e consumidores de serviços, (2) a composição dinâmica de serviços em unidades de maior granularidade, e (3) o gerenciamento dos serviços e composições.

A interação entre provedores e consumidores é viabilizada através do suporte ao estilo

arquitetural SOA. Um elemento central neste contexto é o registro de serviços. Como vimos, um registro funciona como um elemento de indireção que armazena descrições dos serviços disponíveis, podendo ser consultado dinamicamente para descoberta de serviços candidatos. Diferentes modelos de registros compreendem diferentes descrições de serviços e implementam diferentes operações. Visando manter a descrição de um registro sob uma perspectiva conceitual, pode-se dizer que um registro suporta idealmente as seguintes operações: publicação, retirada, atualização, consulta, e notificação.

As operações de publicação, retirada, e atualização compõem a perspectiva do provedor de serviços. A operação de publicação é utilizada para cadastrar uma descrição de um serviço fornecido, associando a essa descrição uma referência para o serviço correspondente. A operação de retirada é utilizada com o intuito de remover uma descrição de serviço, inviabilizando a descoberta dinâmica do mesmo. Por fim, a operação de atualização visa permitir que a descrição do serviço oferecido seja modificada dinamicamente.

Do ponto de vista dos consumidores, as operações relevantes são consulta e notificação. A operação de consulta é utilizada por consumidores para descobrir, sob demanda, serviços com as características necessárias a partir de uma descrição do serviço requerido, a qual normalmente está associada à sua interface. Essa descrição também pode envolver outras propriedades, tais como informações referentes aos provedores de serviços ou ao nível de qualidade requerido. A partir da descrição informada por um consumidor, o registro realiza uma filtragem, retornando para o cliente uma lista contendo os serviços candidatos, cabendo ao consumidor a responsabilidade pela seleção do serviço e pela ligação dinâmica. A operação de notificação visa oferecer maior suporte ao dinamismo, permitindo que um consumidor se registre para receber notificações referentes à publicação, retirada, e atualização de descrições de serviço.

O suporte à composição dinâmica de serviço é normalmente realizado por motores de execução e contêineres embutidos nos ambientes. Os motores de execução são responsáveis por realizar composições comportamentais descritas por fluxos de trabalho, realizando processos de negócio. Uma vez que a presente tese não trata desses processos, as composições comportamentais não serão mais exploradas. Por outro lado, as composições estruturais são suportadas por contêineres que buscam e selecionam serviços requeridos e os integram às composições através de mecanismos de injeção de dependências. Com o intuito de guiar a atuação dos contêineres, as características dos serviços requeridos são tipicamente descritas através de uma ADL. A utilização de contêineres em plataformas orientadas a serviços é discutida na Seção 2.2.4.

Sob a perspectiva do gerenciamento, os ambientes de execução devem compreender mecanismos para permitir desde a instalação e configuração de serviços primitivos e compostos, até a monitoração e eventuais ajustes nesses serviços. Mais ainda, esses ambientes devem idealmente fornecer mecanismos que permitam o gerenciamento do próprio ambiente e dos serviços que ele oferece. Por fim, de forma similar aos ambientes de exe-

cução no mundo de componentes, diversos serviços técnicos responsáveis por aspectos não-funcionais também são comumente integrados aos ambientes de execução.

2.2.4 Modelos de Componentes Orientados a Serviços

A necessidade de tratar de aspectos não relacionados ao domínio do negócio motivou a proposição da utilização de conceitos próprios do mundo de componentes nas soluções baseadas em serviços, dando origem aos modelos de componentes orientados a serviços (CERVANTES; HALL, 2004; ESCOFFIER; HALL; LALANDA, 2007; WALLS, 2009). O núcleo dessa abordagem consiste em fazer com que cada composição declare explicitamente as características dos serviços que esta provê e requer, cabendo a um contêiner a responsabilidade pela seleção dinâmica dos serviços utilizados pela composição, e pela publicação dos serviços providos por ela.

Neste contexto, cabe aos desenvolvedores a responsabilidade por anotar as características dos serviços providos e requeridos, o que é tipicamente realizado junto da descrição arquitetural. As composições projetadas com base nessa abordagem são tipicamente referenciadas como composições estruturais (CERVANTES; HALL, 2004; ESCOFFIER; HALL; LALANDA, 2007), enquanto que aquelas especificadas através de linguagens de fluxo são referenciadas como composições comportamentais, uma vez que seu comportamento é exposto através do fluxo de controle que define a composição.

Dentre os modelos de componentes orientados a serviços, um dos que recebeu bastante atenção da indústria de *software* foi *Service Component Architecture* (SCA) (BEISIEGEL et al., 2007). SCA é definido através de um conjunto de especificações que descrevem um modelo de composição estrutural para a criação de aplicações orientadas a serviços independentes de plataforma. As especificações que definem SCA foram implementadas tanto em plataformas comerciais, como o IBM Websphere [1], quanto em soluções de código aberto, como o Apache Tuscany [2].

Na visão de SCA, as aplicações são formadas por componentes, os quais interagem entre si através do fornecimento e consumo de serviços. SCA não impõe uma linguagem específica para o desenvolvimento desses componentes nem um protocolo específico para a conexão entre eles, trazendo grande flexibilidade para o desenvolvimento de aplicações baseadas em serviços. A proposta também facilita o tratamento de requisitos não-funcionais através da definição dos conceitos de *intents* e *policy sets*.

Contudo, SCA deixou uma importante lacuna do ponto de vista das capacidades de configuração e gerenciamento das aplicações. De fato, embora as especificações estabeleçam como uma aplicação pode ser desenvolvida, configurada, e instalada, ela não incorpora capacidades reflexivas, não sendo capaz de suportar a reconfiguração dinâmica das aplicações.

[1] https://www-03.ibm.com/software/products/pt/appserv-was
[2] http://tuscany.apache.org/

Visando preencher essa importante lacuna, Frascati (SEINTURIER et al., 2009; SEINTURIER et al., 2012) estendeu SCA incorporando o recurso de reflexão de forma sistemática, estabelecendo um modelo de componentes utilizado tanto para implementar as aplicações de negócio quanto a própria plataforma. Um aspecto importante relacionado à implementação de Frascati é que essa plataforma foi realizada com base no modelo de componentes Fractal (BRUNETON, 2009). Embora esse modelo de componentes seja bastante completo, e com possibilidades de customização dos recursos expostos via reflexão, a utilização do modelo de programação de Fractal requer que o programador tenha ciência do modelo de componentes, e desenvolva seu código de acordo com esse modelo. Desta forma, ao adotar o modelo de programação de Fractal/Frascati, o desenvolvedor é levado a misturar código do modelo de componentes com código da aplicação de negócio.

Outra plataforma que permitiu combinar com sucesso os conceitos de componentes e serviços foi OSGi (The OSGi Alliance, 2007). No centro dessa proposta está um conjunto de especificações que definem uma plataforma de serviços baseada em Java. Nessa plataforma, os serviços são empacotados em unidades instaláveis, denominadas *bundles*, que são arquivos *jar* contendo código e recursos. Esses *bundles* podem fornecer e consumir serviços, os quais são publicados no registro da plataforma. A especificação define, ainda, um conjunto de mecanismos que permitem a instalação, ativação, desativação, e remoção de *bundles*.

O registro de OSGi, além de possuir facilidades para consultas de serviços a partir de filtros *Lightweight Directory Access Protocol* (LDAP), possui a capacidade de notificar eventuais consumidores quando da disponibilidade dos serviços por eles requeridos. Essa é uma capacidade bastante interessante do ponto de vista do suporte ao dinamismo. Mais ainda, a plataforma é bastante leve, tendo sido projetada com um foco especial em dispositivos de baixa capacidade de processamento. Por fim, a especificação da própria plataforma prevê a comunicação assíncrona através de um barramento interno de eventos. Em virtude dessas características, a plataforma OSGi vem sendo largamente utilizada como plataforma de base para a construção de software de infraestrutura, como servidores de aplicação, além de ser a base da plataforma Eclipse de desenvolvimento.

Dentre os aspectos negativos relacionados ao uso direto da plataforma OSGi, o principal aspecto identificado diz respeito à complexidade de gerenciamento do dinamismo. De fato, embora o registro tenha a capacidade de notificar as aplicações acerca das ocorrências relacionadas aos serviços, o tratamento correto do dinamismo requer um grande cuidado por parte dos desenvolvedores. Em particular, a percepção desse problema motivou a introdução de várias soluções baseadas em contêineres na plataforma OSGi, destacando-se dentre elas, a plataforma iPojo.

2.2.4.1 Plataforma iPojo

A plataforma iPojo (ESCOFFIER; HALL; LALANDA, 2007; ESCOFFIER; BOURRET; LALANDA, 2013) faz parte do projeto Apache Felix [3] e foi proposta com o intuito de facilitar o desenvolvimento de aplicações baseadas em serviço, mascarando a disponibilidade dinâmica desses elementos. A ideia geral é retirar das aplicações a lógica responsável pela descoberta, seleção, e monitoração dos serviços requeridos, delegando essa responsabilidade para contêineres embutidos na plataforma.

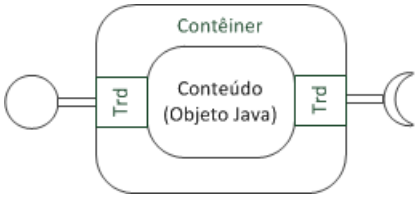
Em iPojo, um serviço é especificado através da definição de uma interface Java que determina as suas operações. Nessa plataforma, os serviços são fornecidos e consumidos por componentes, os quais são implementados através da definição de classes Java "puras", seguindo um modelo de programação normalmente referenciado como *Plain Old Java Objects* (POJO). Neste contexto, para que um componente forneça um serviço, a classe que define seu comportamento deve implementar a interface correspondente. Por outro lado, para que um componente consuma um serviço, sua implementação deve declarar um atributo cujo tipo é definido pela interface do serviço.

Para que um componente possa fornecer e consumir serviços, ele deve ser instalado em um ambiente de execução. Durante o processo de instalação, a plataforma cria uma instância de uma fábrica de componentes, a qual é configurada de acordo com um conjunto de meta-dados definidos através de anotações feitas na classe que implementa o componente e/ou de arquivos de configuração. Uma vez configurada, a fábrica é incluída no registro de serviços e vinculada ao tipo de componente, podendo ser solicitada para criar instâncias. Quando uma instância do componente é criada, ela é envolvida por um contêiner, o qual é responsável por injetar os serviços requeridos e publicar os serviços providos no registro de serviços da plataforma. Cada instância é independente das demais, consumindo e provendo seus próprios serviços.

Para tratar dos diferentes aspectos não-funcionais, a plataforma iPojo introduziu o conceito de tratador (do inglês *handler*). Um tratador é um tipo especial de componente, o qual pode fornecer e requerer serviços, de forma similar aos componentes de aplicação. Em geral, os tratadores são responsáveis por gerir os aspectos não-funcionais, como persistência, transação, etc. Para tanto, os tratadores participam das decisões acerca do estado de uma instância de componente, assim como são informados acerca dos acessos à mesma. Cabe ao contêiner identificar os tratadores necessários de acordo com o tipo de componente, verificar se esses tratadores estão disponíveis na plataforma, criar instâncias dos tratadores, e vinculá-las à instância do componente. Para identificar os tratadores necessários, o contêiner utiliza meta-dados representados, como vimos, nas anotações ou em arquivos de configuração. Em iPojo, não há um conjunto fechado de tratadores pré-estabelecidos. De fato, extensões à plataforma podem ser realizadas através da definição

[3] http://felix.apache.org/

Figura 5 – Instância de componente iPojo



Fonte: baseada em Escoffier, Hall e Lalanda (2007)

de novos tratadores, os quais podem ser associados aos tipos de componente, sendo integrados na atividade de gerenciamento das instâncias desses tipos.

É importante observar que a fábrica cria uma instância de componente, que não é simplesmente uma instância da classe de negócio correspondente ao tipo de componente (vide **Figura 5**). De fato, uma instância de componente é formada por diferentes elementos, incluindo: (1) objetos de negócio criados sob demanda, com base na definição da classe correspondente; (2) contêiner responsável por mediar o acesso ao objeto de negócio; (3) tratadores, responsáveis por gerenciar um aspecto, normalmente não-funcional, relacionado à instância do componente.

Um aspecto de particular interesse na plataforma iPojo diz respeito à disponibilidade dinâmica dos serviços, a qual tem um impacto direto no ciclo de vida dos componentes. Visando minimizar as possibilidades de erro decorrentes da tentativa de uso de serviços quando esses não estão mais disponíveis, o contêiner se registra para ouvir notificações enviadas pelo registro, sendo capaz de detectar que uma dependência não pode mais ser satisfeita logo que um serviço requerido não está mais disponível. Nesse contexto, diz-se que uma dependência de serviços está inválida. Para evitar potencias erros, esse contêiner torna inválida a instância de componente que depende do serviço. Por fim, uma vez que uma instância inválida não pode fornecer serviços, os serviços que seriam normalmente providos por ela devem ser removidos do registro. Assim como os demais aspectos não-funcionais, o suporte ao dinamismo é realizado através de tratadores, em particular, pelos tratadores de dependências e de provimento de serviços, os quais são nativamente incorporados na plataforma.

Em iPojo, uma dependência de serviço é caracterizada por um conjunto de propriedades especificadas através de anotações, estabelecendo, por exemplo, a interface do serviço requerido, a opcionalidade da dependência, o filtro de seleção a ser utilizado na busca de serviço candidato, etc. Essas propriedades são utilizadas para configurar o tratador de dependências de forma que ele possa descobrir e selecionar os serviços que devem ser utilizados por uma instância de componente. Com base nessas informações, o tratador consulta o registro e seleciona os serviços a serem injetados nas instâncias. Por *default*, as dependências são consideradas dinâmicas, podendo passar por ciclos de validação e invalidação conforme haja ou não serviços capazes de atender aos critérios especificados.

A despeito da diversidade de propriedades que podem ser utilizadas na configuração das dependências, o seu gerenciamento é sempre realizado pelo tratador de forma automática, não sendo permitida nenhuma gestão externa. Em suma, não é possível determinar qual o serviço deve ser utilizado, apenas especificar, dentro das configurações possíveis, as características desejadas para o serviço a ser selecionado.

O fornecimento de serviços também é realizado por um tratador que pode ser configurado através de propriedades indicando as interfaces funcionais dos serviços providos, as propriedades que devem ser inseridas no registro, e a política de criação de instâncias da classe que implementa o componente. Nesse contexto, a classe correspondente ao componente deve implementar as interfaces dos serviços providos. O tratador responsável pelos serviços providos também participa do tratamento do dinamismo. Como descrito previamente, se uma instância se torna inválida, ela não pode fornecer serviços, cabendo a esse tratador a remoção dos serviços fornecidos pela instância do registro.

2.2.5 Reflexão

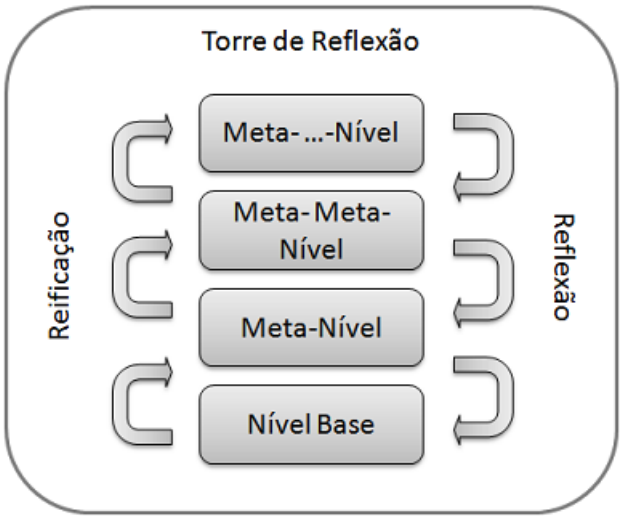
Para que as aplicações possam ser dinamicamente adaptadas, é necessário que elas sejam projetadas para isso. Um passo importante nesse sentido corresponde à concepção de uma aplicação como uma composição de unidades que possam ser dinamicamente substituídas. Embora as seções anteriores tenham apresentado diferentes paradigmas que podem ser utilizados nesse sentido, elas não apresentaram como, efetivamente, as reconfigurações dinâmicas podem ser realizadas. Nesse contexto, um mecanismo essencial consiste no uso do padrão arquitetural reflexão (SMITH, 1982; BUSCHMANN et al., 1996).

Em sua essência, a computação reflexiva permite que os sistemas computacionais manipulem representações deles mesmos, de forma similar às manipulações que estes realizam nas representações do seu domínio de negócio. Estas representações próprias, referenciadas como meta-representações, devem compreender aspectos relativos ao seu estado e/ou comportamento. Um sistema computacional é dito reflexivo se existe uma conexão causal entre o próprio sistema em execução e a sua meta-representação, de forma que alterações nesta representação promovam modificações equivalentes no sistema em execução e vice-versa (MAES, 1987).

O processo de exposição das características e comportamentos de um sistema através da criação das meta-representações é denominado reificação. O processo inverso, que corresponde à adaptação do sistema de forma a refletir alterações efetuadas na sua meta-representação é denominado reflexão. Ainda em termos de terminologia, a utilização de uma meta-representação para fins de observação e análise é referenciada como introspecção, enquanto que a intersecção corresponde à utilização da meta-representação para promover modificações no estado e/ou comportamento da aplicação em execução.

Como observado em Ferber (1989), a concepção de sistemas reflexivos no paradigma de orientação a objetos pode ser realizada de forma natural utilizando o conceito de objetos

Figura 6 – Torre de Reflexão



Fonte: baseada em (COSTA, 2001)

tanto na construção da lógica de negócio, onde os objetos devem representar elementos do domínio da aplicação, quanto na construção da meta-representação, onde objetos (nesse contexto referenciados como meta-objetos) são utilizados para representar os objetos que compõem a aplicação. Nesse sentido, as computações realizadas pelos meta-objetos, referenciadas como meta-computações, tem a finalidade de observar, analisar e, eventualmente, modificar os objetos representados, modificando, assim, a própria aplicação. Uma evolução natural desse conceito consiste na utilização de componentes como unidades elementares de composição, tanto no nível base quanto no nível meta (HNETYNKA; PLASIL, 2011).

A estruturação dos sistemas em termos de objetos e meta-objetos (ou componentes e meta-componentes) leva à proposição de uma arquitetura organizada em níveis. Neste contexto, o nível mais baixo da hierarquia, chamado de nível base, é composto pelo conjunto de objetos (ou componentes) responsáveis por realizar as funcionalidades relacionadas ao domínio de negócio da aplicação. O nível acima do nível base é composto pelos meta-objetos (ou meta-componentes), os quais integram a meta-representação, sendo responsáveis por prover a capacidade reflexiva do sistema. Esta organização em níveis pode ser naturalmente estendida, uma vez que meta-meta-objetos podem ser utilizados para representar os próprios meta-objetos, levando à constituição de torres de reflexão, onde cada nível define uma representação causalmente conectada com o nível abaixo (vide **Figura 6**). É importante destacar que a estruturação da arquitetura em camadas permite uma separação natural de preocupações, sendo o nível base responsável pela realização dos requisitos de negócio, e os níveis meta responsáveis por realizar as características não-funcionais.

O acesso aos meta-objetos pode ocorrer de forma implícita ou explícita. O mecanismo

de acesso implícito permite que invocações no nível base sejam reificadas de forma que o controle da sua execução seja transferido para o nível meta sem que o nível base tenha ciência desse fato. Este tipo de acesso é normalmente implementado através da utilização de interceptadores.

A disponibilização de um mecanismo de acesso explícito é responsável por possibilitar a meta-programação, ou seja, é através dele que são expostas as interfaces dos meta-objetos (referenciadas como meta-interfaces). Estas interfaces são normalmente organizadas em meta-espaços que, em conjunto, compõem o que se chama de protocolo de meta-objetos, ou seja, uma API para acesso às funcionalidades do nível meta, que pode ser utilizada para fins de introspecção e/ou intersecção.

Nos sistemas adaptativos, a manipulação das meta-representações pode ser realizada por seres humanos através de ferramentas/*scripts* de administração, viabilizando a reconfiguração da aplicação sem a necessidade de interromper inteiramente a sua execução. No contexto dos sistemas auto-adaptativos, os próprios sistemas conduzem o seu processo de adaptação. Como veremos na seção seguinte, o projeto desses sistemas é complexo, uma vez que envolve, além da lógica de negócio em si, a lógica responsável pela condução da adaptação.

Considerando-se a relevância dos mecanismos de reflexão, eles são normalmente embutidos em diferentes níveis, desde as linguagens de programação até os modelos de componentes (e.g., Fractal (BRUNETON, 2009) e SOFA2 (HNETYNKA; PLASIL, 2011)). No contexto dos modelos de componentes, o recurso da reflexão é responsável por expor as propriedades de configuração dos componentes através de elementos do nível meta, e por permitir que essas representações sejam manipuladas, promovendo alterações das propriedades dos componentes em execução no nível base. Desta forma, o uso de reflexão viabiliza a realização da adaptação paramétrica.

Do ponto de vista da adaptação composicional, os mecanismos de reflexão embutidos nos modelos de componentes são responsáveis por permitir que uma aplicação adicione, remova, ou substitua dinamicamente componentes. Em suma, as capacidades reflexivas embutidas nos modelos de componentes fazem com que esses modelos tenham papel de destaque nas pesquisas relacionadas ao desenvolvimento das aplicações adaptativas.

2.3 APLICAÇÕES AUTO-ADAPTATIVAS

Segundo Bruni et al. (2012), uma aplicação *adaptativa* deve disponibilizar uma coleção de meta-dados de controle, os quais podem ser modificados em tempo de execução, de forma a promover alterações no seu comportamento. Quando a própria aplicação utiliza esses meta-dados para dirigir seu processo de adaptação, ela é dita *auto-adaptativa*. No contexto dos sistemas reflexivos, esses meta-dados compõem a meta-representação da aplicação, a qual é exposta através de meta-objetos (ou meta-componentes).

A despeito da disponibilização de uma meta-representação causalmente conectada com uma aplicação representar uma característica necessária para viabilizar a concepção das aplicações auto-adaptativas, ela não é, em geral, suficiente. De fato, a concepção de aplicações auto-adaptativas requer uma característica essencial: a capacidade de tomar decisões em tempo de execução. Para que uma aplicação possa tomar tais decisões e, consequentemente, se adaptar, é necessário que ela mantenha conhecimento acerca dela própria (i.e., *self-awareness* (HINCHEY; STERRITT, 2006)) e do ambiente no qual ela está inserida (i.e., *context-awareness* (PARASHAR; HARIRI, 2005)). Dessa forma, a construção de uma aplicação auto-adaptativa requer além de uma meta-representação da própria aplicação, uma meta-representação dos elementos que integram seu ambiente de execução.

Uma vez que essas meta-representações são responsáveis por concentrar as informações necessárias para a condução do processo de adaptação, é fundamental que essas reflitam o estado atual da aplicação e do seu ambiente. Assim, uma aplicação auto-adaptativa deve ser capaz de monitorar esses elementos ao longo de sua execução e de utilizar os dados coletados para atualizar as meta-representações.

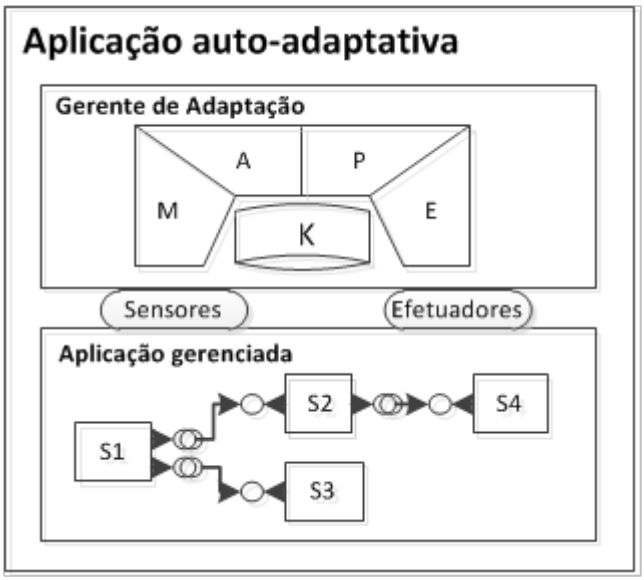
A partir das informações mantidas nas meta-representações, uma aplicação auto-adaptativa deve ser capaz de: identificar as circunstâncias que requerem sua adaptação, planejar a adaptação visando a satisfação dos seus requisitos, e realizar a adaptação em tempo de execução sem a necessidade de intervenção humana. O conjunto de atividades relacionadas à manutenção dessas meta-representações e à sua utilização na realização da adaptação de uma aplicação corresponde à implementação de um laço fechado de controle que é considerado elemento essencial na engenharia das aplicações auto-adaptativas (BRUN et al., 2009).

2.3.1 Laço de Controle

Os laços de controle implementados nos sistemas auto-adaptativos correspondem à ideia de retroalimentação, a qual consiste na utilização de medidas extraídas das saídas dos sistemas (e.g., tempos de resposta e taxas de utilização de recursos) para a composição de entradas de controle. Estas entradas são utilizadas por um controlador responsável por modificar o comportamento do sistema com o intuito de atingir objetivos previamente estabelecidos. Uma aplicação auto-adaptativa projetada com base nessas ideias pode ser concebida como sendo composta por dois módulos: o gerente de adaptação e o núcleo adaptável da aplicação. De forma geral, o gerente de adaptação compreende atividades de supervisão e de adaptação propriamente dita, permitindo que o núcleo da aplicação se concentre unicamente na lógica de negócio (BENNACEUR et al., 2014).

Com base nessas ideias, diferentes laços de controle foram propostos na literatura, destacando-se dentre eles, o laço que integra a arquitetura de referência para aplicações autonômicas proposta pela IBM, denominada *Autonomic Computing Reference Architecture (ACRA)* (IBM, 2005). É importante destacar que, no contexto desse trabalho, o

Figura 7 – Aplicação auto-adaptativa baseada em MAPE-K



Fonte: o autor

conceito de aplicação autonômica é similar ao de aplicação auto-adaptativa.

2.3.2 Autonomic Computing Reference Architecture (ACRA)

A arquitetura ACRA foi proposta com o intuito de definir os elementos conceituais que tipicamente integram as aplicações autonômicas. Segundo essa arquitetura, uma aplicação autonômica é composta por dois módulos: o gerente autonômico (ou de adaptação) e a aplicação gerenciada (adaptável). A interação entre o gerente autonômico e a aplicação gerenciada é realizada através de pontos de controle, que podem ser de dois tipos: sensores e efetuadores.

Os sensores são elementos centrais para a atividade de supervisão, sendo responsáveis por prover informações sobre o estado atual da aplicação gerenciada e sobre o seu ambiente de execução. Por outro lado, os efetuadores são essenciais para a atividade de adaptação e representam as ações que podem ser utilizadas para modificar o comportamento da aplicação.

O núcleo do gerente autonômico consiste na implementação de um laço de controle, referenciado como MAPE-K (IBM, 2005), acrônimo utilizado para representar as quatro atividades que integram o laço (*Monitor, Analyse, Plan, Execute*), juntamente com a base de conhecimento (*Knowledge*) responsável por manter os dados concernentes ao estado da aplicação e do ambiente. A **Figura 7** representa os elementos que compõem uma aplicação auto-adaptativa, destacando o laço de controle MAPE-K, que compõe o núcleo do gerente de adaptação.

No contexto do laço MAPE-K, a monitoração tem como finalidade a construção/atualização de uma meta-representação compreendendo a aplicação gerenciada e o seu

ambiente de execução. Esta meta-representação pode ser mais ou menos sofisticada, dependendo da complexidade da aplicação e do nível de gerenciamento desejado. Durante a monitoração, o gerente autonômico coleta informações a partir do conjunto de sensores providos pela aplicação gerenciada e por seu ambiente. Normalmente, a informação coletada diretamente dos sensores é de baixo nível e deve ser filtrada, transformada, e agregada de forma a viabilizar a construção de representações de mais alto-nível.

A atividade de análise utiliza a representação construída e mantida pela monitoração para identificar eventuais problemas. Quando um problema é identificado, a análise deve indicar qual o estado esperado para a aplicação diante do seu contexto atual. O planejamento deve receber da análise a indicação do estado esperado, sendo seu objetivo determinar uma sequência de ações corretivas, as quais devem ser executadas para conduzir a aplicação até esse estado. As ações passíveis de uso no planejamento dependem dos efetuadores disponibilizados pela aplicação gerenciada. O laço de controle se encerra com a atividade de execução que é responsável pela realização do plano estabelecido através do uso dos efetuadores disponíveis.

Por fim, um aspecto fundamental na implementação de uma aplicação com base no laço de controle MAPE-K diz respeito ao projeto da base de conhecimentos. Nesse contexto, uma abordagem promissora consiste em construir essa base a partir de um conjunto de modelos mantidos em tempo de execução.

2.3.3 Modelos em Tempo de Execução

Vimos na seção anterior que uma solução comumente utilizada para desenvolver aplicações auto-adaptativas consiste na utilização de gerentes de adaptação que implementam um laço de controle baseado em uma base de conhecimento, a qual corresponde, em essência, a uma meta-representação causalmente conectada com a aplicação adaptável. Nesse contexto, as ações de adaptação são realizadas sobre essa meta-representação e refletidas na aplicação através da conexão existente entre elas.

Como discutido em Vogel e Giese (2010), uma prática comum consiste em utilizar como meta-representações modelos reflexivos bastante próximos da implementação, fornecendo aos gerentes de adaptação uma visão de baixo nível e tornando o seu desenvolvimento mais complexo. Mais ainda, os gerentes de adaptação desenvolvidos com base nessa abordagem estão fortemente ligados à plataforma de execução subjacente, tendo sua reusabilidade comprometida.

Com o intuito de permitir o desenvolvimento de soluções de mais alto nível para suprir as necessidades de adaptação, e impulsionadas pelos avanços das pesquisas na área de *Model-Driven Engineering* (MDE) (FRANCE; RUMPE, 2007), diferentes equipes de pesquisa propuseram a ideia de utilização de modelos mais abstratos em tempo de execução, nascendo assim os *models@runtime* (BLAIR; BENCOMO; FRANCE, 2009).

Segundo Blair et al. Blair, Bencomo e France (2009), um modelo em tempo de execução é uma representação causalmente conectada com um sistema que enfatiza a estrutura, comportamento, ou objetivos desse sistema, projetada a partir de uma perspectiva orientada ao espaço do problema.

Dada sua natureza reflexiva, decorrente da conexão causal com o sistema, os modelos em tempo de execução podem ser utilizados para fins de monitoração, análise, e adaptação, podendo compor a base de conhecimento utilizada pelos gerentes de adaptação. De fato, a utilização de modelos em tempo de execução como mecanismo de abstração entre o gerente de adaptação e uma aplicação adaptável possui dois importantes benefícios. De um lado, há uma potencial redução da complexidade do gerente, uma vez que este manipula conceitos de mais alto-nível, e, idealmente, independentes de plataforma. De outro, há uma maior possibilidade de reuso, pois os gerentes não ficam vinculados à plataforma de execução específica.

Atualmente essa modalidade de sistemas reflexivos tem uma grande relevância para o desenvolvimento de aplicações auto-adaptativas, sendo frequente a proposição de soluções baseadas em conjuntos de modelos mantidos em tempo de execução (MORIN; BARAIS, 2008; VOGEL; SEIBEL; GIESE, 2011; HUBER et al., 2014a).

2.4 APLICAÇÕES BASEADAS EM SERVIÇOS CIENTES DE QUALIDADE

No contexto deste trabalho, uma categoria particularmente interessante de aplicações auto-adaptativas é formada pelas QSBA (*Quality-aware Service Based Aplications*). As QSBAs são composições de serviço inerentemente dinâmicas, nas quais o nível de qualidade dos serviços é utilizado como gatilho para o processo de adaptação.

Em geral, o suporte a essas aplicações envolve o desenvolvimento de uma infraestrutura SOA compreendendo mecanismos que permitam a monitoração dos serviços disponíveis e a adaptação das aplicações em execução. A elaboração desses mecanismos requer uma representação clara das características de qualidade dos serviços.

2.4.1 Qualidade de Serviço

O conceito de qualidade de serviço compreende o conjunto de características não-funcionais, as quais são utilizadas para descrever *como* um serviço é realizado, ao invés de *o quê* efetivamente ele faz (CHUNG; LEITE, 2009). Essas características são consideradas importantes fatores distintivos entre serviços que oferecem uma mesma funcionalidade. De fato, embora haja uma grande ênfase nos aspectos funcionais, "a funcionalidade não é útil ou utilizável sem as necessárias características não-funcionais" (CHUNG; LEITE, 2009).

De acordo com Kritikos e Plexousakis (2009), "a qualidade de serviço corresponde ao conjunto de características não-funcionais que podem impactar a capacidade do serviço de satisfazer as necessidades estabelecidas ou implícitas". Embora a descrição apresentada

forneça uma visão conceitual de qualidade e permita distinguir aspectos funcionais e não-funcionais, ela possui um certo grau de subjetividade, não fornecendo os elementos necessários para descrever o nível de qualidade dos serviços de forma precisa. Visando preencher essa lacuna e estabelecer uma representação adequada que pudesse, ao mesmo tempo, padronizar e estruturar as características de qualidade, foram propostos os *modelos de qualidade*.

2.4.2 Modelos de Qualidade

Segundo Benbernou et al. (2010), um modelo de qualidade (*Quality Model* (QM)) é uma taxonomia ou categorização de atributos de qualidade, onde cada atributo representa um aspecto particular associado à qualidade de um item. Esses atributos podem ser: mensuráveis ou não-mensuráveis. Enquanto os atributos não-mensuráveis representam, em geral, informações estáticas e de natureza qualitativa, os atributos mensuráveis são inerentemente dinâmicos, sendo avaliados através de métricas de qualidade, as quais representam medidas quantitativas que indicam em que grau um item possui um determinado atributo de qualidade (BURNSTEIN, 2010). Essa natureza dinâmica torna os atributos mensuráveis centrais para plataformas de suporte às QSBAs, sendo normalmente utilizados como gatilhos para o processo de adaptação. Assim, a despeito da relevância dos demais atributos, o presente trabalho se concentra unicamente nos atributos de qualidade mensuráveis (e.g., tempo de resposta), os quais devem ser monitorados pelas infraestruturas SOA e utilizados para guiar a adaptação das aplicações.

De acordo com ISO/IEC (2010), os modelos de qualidade desempenham diferentes papeis, sendo essenciais para: especificar requisitos, estabelecer medições, e realizar avaliações de qualidade. Para que possam ser utilizados em todo o seu potencial no contexto das infraestruturas SOA, os modelos de qualidade devem conter informações suficientes tanto para atender às necessidades dos provedores e consumidores de serviço, quanto àquelas da infraestrutura SOA subjacente.

Do ponto de vista dos provedores e consumidores de serviço, o principal requisito relacionado aos modelos de qualidade é que estes forneçam uma representação clara e precisa dos conceitos de qualidade, definindo uma taxonomia que compreenda os atributos e as métricas de qualidade necessários, os quais são normalmente organizados em categorias, tais como desempenho e segurança.

Embora o conceito de modelo de qualidade possua um papel central no contexto de SOA, não há um modelo padrão capaz de estabelecer um consenso na comunidade, provocando uma proliferação de modelos com características distintas. Esses modelos diferem entre si em diversos aspectos, incluindo: tamanho, estrutura, terminologia, conjunto de atributos, e assim por diante (ORIOL; MARCO; FRANCH, 2014).

Em geral, essas diferenças decorrem da percepção de que aplicações distintas podem necessitar de diferentes atributos e métricas de qualidade. Consequentemente, uma re-

presentação única e universal dos atributos de qualidade dificilmente será suficiente (LIU; NGU; ZENG, 2004; SENTILLES, 2012). Em um cenário ideal, uma aplicação deve ser capaz de definir, de forma independente, seu modelo de qualidade customizado, estabelecendo uma taxonomia própria. Mais ainda, esse modelo deve ser extensível, viabilizando a definição dinâmica de novos atributos e métricas.

Para que uma infraestrutura SOA seja capaz de suportar o nível de customização e flexibilidade descritos, ela não deve ser projetada com base em um modelo de qualidade específico e pré-definido. Idealmente, as infraestruturas SOA devem ser concebidas em torno de uma representação de qualidade mais geral e extensível, que pode ser realizada sob a forma de um meta-modelo, representando os elementos conceituais relacionados à descrição de qualidade (e.g., *Categoria, Atributo, e Métrica*) (KRITIKOS et al., 2013).

Essa abordagem estabelece um mecanismo padrão para expressar modelos de qualidade, os quais podem ser dinamicamente interpretados, de forma a definir novos atributos e métricas. No centro dessa proposta está o padrão *Type Object* (YODER; JOHNSON, 2002), o qual permite a definição de novos tipos em tempo de execução através da criação de instâncias de uma classe genérica.

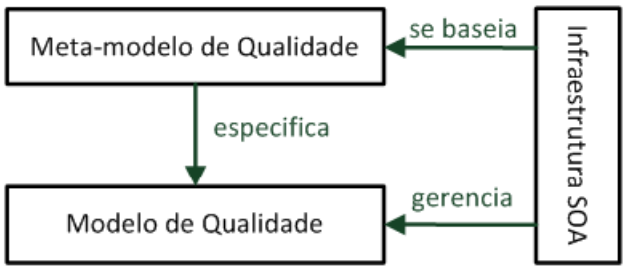
Uma vez que os conceitos representados no meta-modelo são embutidos nas infraestruturas SOA, estas são capazes de compreender os modelos de qualidade customizados, os quais são compostos por instâncias dos elementos definidos no meta-modelo. Esses modelos de qualidade podem ser mantidos em tempo de execução, sendo naturalmente extensíveis. As infraestruturas SOA concebidas a partir dessa abordagem possuem um elevado grau de flexibilidade do ponto de vista da descrição das características de qualidade. A **Figura 8** (baseada em Benbernou et al. (2010)) representa a relação entre o meta-modelo, os documentos de qualidade, e a infraestrutura SOA.

É importante observar que a possibilidade de concepção de modelos de qualidade customizados favorece a separação de papéis nas equipes de desenvolvimento, introduzindo a figura do especialista em qualidade. Este novo papel é responsável por utilizar ferramentas de apoio ao desenvolvimento, incluindo editores (gráficos ou textuais), para conceber modelos de qualidade contendo os atributos e métricas necessários. Por fim, os modelos de qualidade concebidos podem ser compartilhados por um conjunto de provedores e consumidores de serviços para que eles possam ser utilizados para compor aplicações.

Embora um modelo de qualidade seja essencial, uma vez que define o vocabulário de atributos e métricas de qualidade, ele não é o único documento relevante para a caracterização do nível de qualidade dos serviços. De fato, as características de qualidade dos serviços providos e requeridos são especificadas através de descritores de serviços baseados em qualidade (*Quality-based Service Description*s (QSDs)) (KRITIKOS et al., 2013). Esses descritores, classificados como *ofertas* ou *requisições*, são compostos, essencialmente, por um conjunto de restrições associadas às métricas definidas nos modelos de qualidade.

Idealmente, uma QSD descrevendo um serviço requerido (i.e., uma requisição) deve

Figura 8 – Relação entre meta-modelo de qualidade, modelo de qualidade, e infraestrutura SOA



Fonte: baseada em Kritikos et al. (2013)

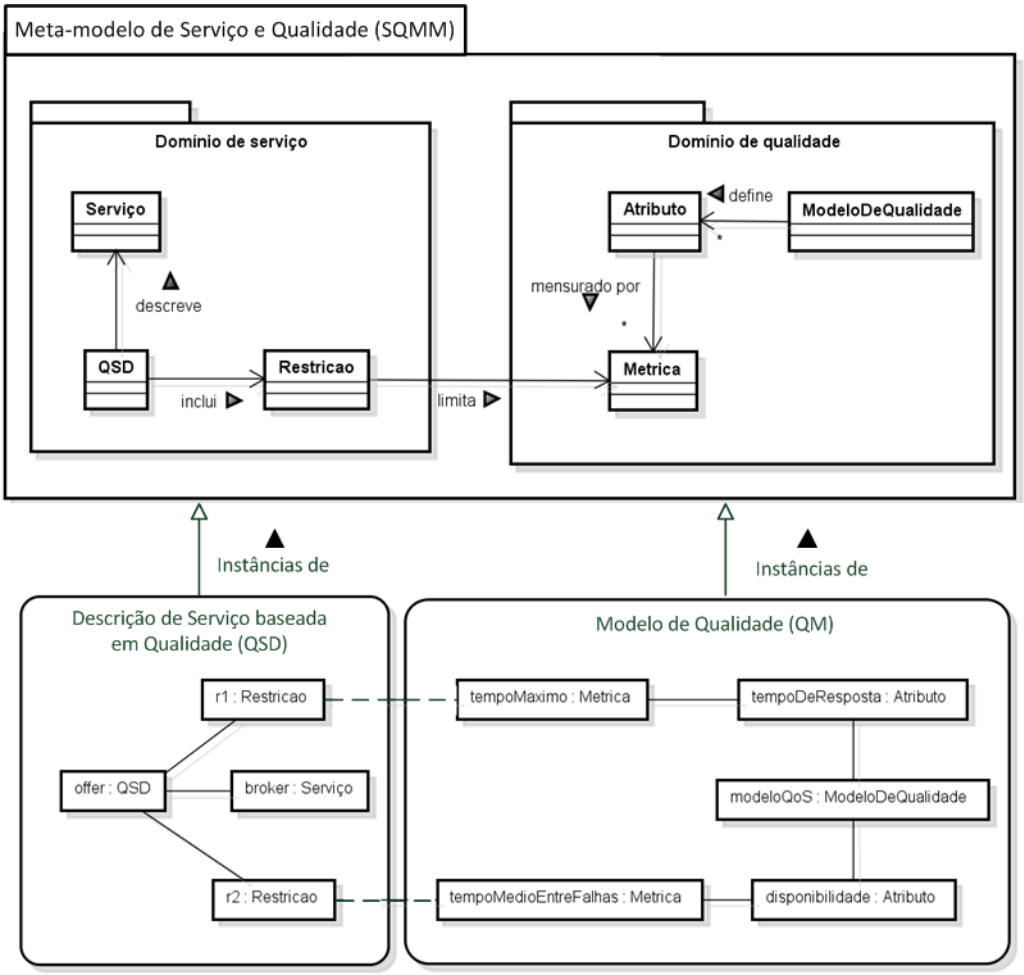
permitir que um consumidor associe, a cada restrição, uma preferência e/ou prioridade. Esse tipo de informação é essencial quando existem diferentes serviços candidatos oferecendo funcionalidade equivalente, cada um apresentando suas próprias características de qualidade. Neste contexto, a especificação das preferências e prioridades permite que esses serviços sejam classificados em função do que é efetivamente mais relevante para um dado consumidor de serviço.

Visando definir, de forma integrada, os elementos conceituais utilizados na descrição da qualidade dos serviços, Kritikos et al. (2013) propõe que os conceitos relacionados ao domínio de serviços também sejam representados em nível de meta-modelo, dando origem ao conceito de meta-modelo de serviço e qualidade (*Service Quality Meta-Model* (SQMM)). Esse meta-modelo incorpora, além dos elementos que compõem a taxonomia de qualidade, conceitos como: *Serviço, Descritor de Serviço, e Restrição*. A **Figura 9** apresenta as relações entre um meta-modelo de serviço e qualidade, uma descrição de serviço baseada em qualidade, e um modelo de qualidade.

Apesar dos elementos apresentados até o momento serem suficientes para descrever os serviços do ponto de vista dos provedores e consumidores de serviço, eles não são suficientes para permitir o projeto de uma infraestrutura SOA capaz de suportar as QSBAs. Do ponto de vista dessas infraestruturas, o requisito fundamental é que a descrição de qualidade possa ser facilmente processável e que possua informações para a viabilizar o gerenciamento das QSBAs ao longo do seu ciclo de vida, compreendendo: (1) publicação de serviços, (2) descoberta e seleção, (3) utilização e monitoração, e (4) adaptação (vide **Figura 10**).

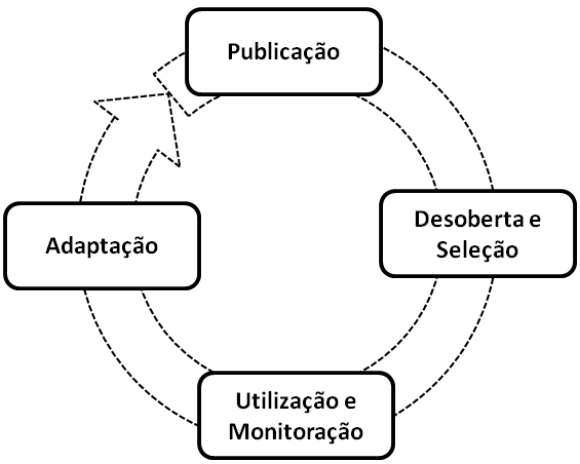
Em geral, as informações contidas nas QSDs e nos modelos de qualidade são suficientes para permitir a publicação, descoberta, e seleção de serviços, mas não possuem elementos para permitir uma monitoração customizada das características de qualidade. Desta forma, os provedores e consumidores de serviços não são capazes de informar à infraestrutura como ela deve monitorar os atributos de qualidade relevantes.

Figura 9 – Relação entre SQMM, QSD, e QM



Fonte: o autor

Figura 10 – Ciclo de vida de uma QSBA



Fonte: baseada em (KRITIKOS et al., 2013)

Como discutido em Kritikos et al. (2013), Oriol, Marco e Franch (2014), poucos modelos possuem uma descrição acerca de como as métricas de qualidade devem ser computadas. Mais ainda, mesmo nas propostas que incorporam alguma descrição das métricas, esta descrição é normalmente realizada em torno de uma referência para um recurso externo (e.g., um agente de monitoração) capaz de devolver o valor corrente da métrica, ficando os desenvolvedores de aplicação/serviço à margem desse processo. A ausência de informação acerca de como as métricas devem ser dinamicamente monitoradas compromete a utilidade desses modelos de qualidade, trazendo reflexos diretos no processo de adaptação, uma vez que este é realizado com base nos dados coletados na monitoração.

De fato, uma solução comumente encontrada na literatura, é embutir nas próprias plataformas as métricas relevantes. Neste contexto, os provedores e consumidores de serviços não são capazes de determinar como as métricas devem ser computadas, devendo aceitar os algoritmos de computação de métricas integrados na plataforma. Como vimos, a utilização de um universo de métricas pré-definidas (através da adoção de um modelo de qualidade específico) possui um impacto direto na extensibilidade e flexibilidade das plataformas SOA.

Em suma, em uma infraestrutura SOA ideal, os desenvolvedores devem ser capazes não só de descrever os serviços do ponto de vista de qualidade, mas também de especificar o conjunto de métricas de qualidade que deve ser utilizado para avaliar esses serviços. Mais ainda, é importante que esses desenvolvedores possam definir como essas métricas podem ser efetivamente computadas, estabelecendo, através de uma notação de alto nível de abstração e independente de plataforma tecnológica, os "algoritmos de computação das métricas". Por fim, o universo de métricas reconhecido pela plataforma deve ser dinamicamente extensível, viabilizando a definição de novas métricas e algoritmos de computação em tempo de execução.

2.5 CONSIDERAÇÕES FINAIS

Como vimos nesse capítulo, as aplicações auto-adaptativas são capazes de modificar o seu comportamento em tempo execução. Em geral, existem duas abordagens que podem ser utilizadas para promover adaptações em tempo de execução. A abordagem paramétrica consiste na modificação dinâmica de parâmetros interpretados por uma aplicação, possibilitando uma mudança em seu comportamento. A flexibilidade decorrente da abordagem paramétrica é limitada, uma vez que o seu uso direto permite apenas a seleção de comportamentos previstos em tempo de desenvolvimento.

Uma solução mais flexível do ponto de vista da adaptação consiste na utilização da abordagem composicional, que viabiliza a substituição dos elementos que constituem a aplicação em execução. Para que essa abordagem possa ser utilizada, é necessário que uma aplicação possa ser concebida como uma composição de unidades elementares fortemente coesas e fracamente acopladas. Nesse cenário, o desenvolvimento de uma aplicação con-

siste em duas etapas: (1) concepção das unidades elementares de reuso, e (2) composição (montagem) da aplicação a partir das unidades previamente disponíveis. Nesse contexto, dois paradigmas, eventualmente vistos como complementares, podem ser utilizados: desenvolvimento baseado em componentes e computação orientada a serviços.

Embora a orientação a componentes e a serviços viabilizem a adaptação em tempo de execução, esses paradigmas não são suficientes para a construção de aplicações auto-adaptativas. De fato, uma aplicação auto-adaptativa deve ser capaz de conduzir seu próprio processo de adaptação, devendo para tanto ser capaz de supervisionar e adaptar seu comportamento em função de modificações em seu contexto de execução. Em suma, a natureza auto-adaptativa dessas aplicações impõe a necessidade de embutir nas aplicações uma lógica de adaptação, a qual é normalmente realizada através da implementação de um laço de controle.

Visando separar as lógicas de negócio e de adaptação, a construção de aplicações auto-adaptativas normalmente se baseia na utilização de gerentes de adaptação externos, os quais implementam a lógica de adaptação sob a forma de um laço de controle, realizando o gerenciamento da porção adaptável da aplicação, a qual está centrada no domínio de negócio. Para que esses gerentes possam tomar decisões, eles devem utilizar uma base de conhecimento formada por meta-representações das aplicações. Nesse contexto, o uso das técnicas de reflexão viabiliza a conexão causal entre essas meta-representações e a aplicação em execução, permitindo que as representações sejam utilizadas como interfaces de gerenciamento.

Para simplificar o projeto dos gerentes e ampliar a sua possibilidade de reuso, as meta-representações que integram a base de conhecimento são normalmente constituídas de modelos de alto nível mantidos em tempo de execução, referenciados frequentemente como *models@runtime*.

Dentre as aplicações auto-adaptativas, as QSBAs são de particular interesse para esta tese. Essas aplicações são composições dinâmicas, nas quais o processo de adaptação é guiado pela qualidade dos serviços. Para que essas aplicações possam ser gerenciadas em tempo de execução, é necessária uma definição clara das características de qualidade, envolvendo desde a definição de uma taxonomia (através da concepção de um modelo de qualidade extensível) até a ampliação dos descritores de serviços, os quais devem contemplar tanto características funcionais quanto não-funcionais. Mais ainda, é necessário que as descrições de qualidade contenham informações suficientes para subsidiar a infraestrutura subjacente na condução da monitoração e adaptação das aplicações.

Visando contextualizar os conceitos apresentados, o capítulo seguinte discute como eles podem ser organizados na proposição de uma plataforma genérica de suporte às aplicações auto-adaptativas.

3 PLATAFORMA GENÉRICA DE SUPORTE ÀS APLICAÇÕES AUTO-ADAPTATIVAS

> *It is not what you say, but how you say it.*
>
> Archibald Putt,

A plataforma DSOA foi concebida com o propósito de apoiar o desenvolvimento e execução de aplicações auto-adaptativas baseadas em serviço e cientes de qualidade, referidas no contexto desse trabalho como QSBAs. Essas aplicações são inerentemente dinâmicas, sendo capazes de se reconfigurar em função de variações no nível de qualidade dos serviços providos e requeridos. Desenvolver plataformas capazes de suportar aplicações auto-adaptativas é, por natureza, um problema complexo, envolvendo desde a concepção de um ambiente de execução até a construção de ferramentas de apoio ao desenvolvimento.

Para contextualizar as dificuldades envolvidas na concepção de uma plataforma dessa natureza, o presente capítulo organiza o arcabouço conceitual apresentado anteriormente com o intuito de definir uma plataforma genérica de suporte às aplicações auto-adaptativas.

Visando estruturar a apresentação dos elementos que compõem essa plataforma, dividimos o capítulo em três partes. Primeiramente, apresentamos os elementos que normalmente integram um ambiente de execução capaz de suportar a auto-adaptação, discorrendo acerca do papel de cada um e de como eles podem ser efetivamente integrados.

Na sequência, apresentamos os recursos que devem ser fornecidos por um ambiente de desenvolvimento, focando no papel dos modelos e nos benefícios que podem ser obtidos a partir da utilização de uma abordagem de desenvolvimento dirigida por esses modelos.

Por fim, discutimos o papel dos ambientes de meta-modelagem, os quais são utilizados para definir as linguagens embutidas nos ambientes de desenvolvimento. A partir da definição dessas linguagens, é possível a geração de um conjunto de ferramentas integradas nesses ambientes, incluindo editores e transformadores de modelos, e geradores de código.

3.1 AMBIENTE DE EXECUÇÃO

A combinação dos conceitos de componentes e serviços com as ideias de modelos em tempo de execução formam a essência de uma abordagem promissora para a construção de aplicações auto-adaptativas. De fato, enquanto a utilização de componentes e serviços fornece as bases para a construção de aplicações a partir de unidades elementares de reuso fortemente coesas e fracamente acopladas, a utilização de modelos em tempo de execução disponibiliza interfaces reflexivas de alto nível, tipicamente expostas por modelos arquiteturais. Assim, as ações de reconfiguração (como a substituição dos elementos que compõem uma aplicação) são efetivadas nos modelos e propagadas para a aplicação em execução através da conexão causal existente entre eles.

Nas aplicações auto-adaptativas, a responsabilidade pelo processo de adaptação é da própria aplicação, sendo normalmente materializada através da introdução de um gerente de adaptação, que utiliza as informações contidas nos modelos para realizar um laço de controle que combina as atividades de supervisão e adaptação (BENNACEUR et al., 2014). Assim, a natureza auto-adaptativa de uma aplicação advém da visão desta como uma composição da aplicação gerenciada, situada no nível base da arquitetura reflexiva, com o gerente de adaptação, que compõe, juntamente com os modelos, o nível meta. Uma vez que os modelos concentram os dados utilizados pelos gerentes de adaptação e definem as interfaces de gerenciamento da aplicação, eles ocupam um papel de destaque na concepção dos ambientes de execução.

3.1.1 Modelos em Tempo de Execução

Como vimos, um modelo em tempo de execução é uma representação causalmente conectada de um sistema que enfatiza sua estrutura, comportamento, ou objetivos, projetada a partir de uma perspectiva orientada ao espaço do problema (BLAIR; BENCOMO; FRANCE, 2009). Segundo France France e Rumpe (2007), a ideia por trás da utilização de modelos em tempo de execução é fazer com que esses modelos sejam utilizados por agentes de mudança como interfaces para adaptar, reparar, estender, ou retroalimentar um sistema em execução. Nessa visão, os modelos não são somente artefatos primários de desenvolvimento, eles representam também o mecanismo básico através do qual desenvolvedores e sistemas podem entender, configurar, e modificar o comportamento de um sistema em execução.

Segundo Vogel Vogel e Giese (2012), grande parte das pesquisas na área de modelos em tempo de execução está concentrada na produção e utilização de modelos arquiteturais causalmente conectados com uma aplicação em execução. De forma geral, esses modelos são constituídos por instâncias de conceitos que compõem um *modelo abstrato de componentes*, tipicamente representando a aplicação a partir de seus componentes, interfaces, e conectores. Uma característica importante desses modelos é a capacidade de evoluir ao longo do tempo. Por um lado, essa capacidade é necessária para garantir que os modelos descrevam, de forma fidedigna, o estado atual de uma aplicação em execução (função descritiva). Por outro lado, um modelo em tempo de execução pode ser modificado para indicar o estado desejado da aplicação, ou seja, o estado para o qual essa aplicação deve evoluir em função da conexão causal existente entre ela e o respectivo modelo (função prescritiva). Assim, um modelo arquitetural mantido em tempo de execução deve ser dinâmico e deve possuir elementos capazes de suportar tanto a sua função descritiva, quanto prescritiva (LEHMANN et al., 2011).

Uma vez que os modelos arquiteturais são utilizados para fins de reconfiguração dinâmica das aplicações, é necessário associar comportamento a esses modelos, o que pode ser feito de duas formas. A primeira consiste na definição de operações de reconfiguração

como parte integrante dos elementos do modelo mantido em tempo de execução. Essa abordagem se baseia na definição de "modelos ricos", nos quais a definição dos elementos conceituais compreende tanto a sua estrutura quanto o seu comportamento. Uma segunda possibilidade consiste na visão do modelo como uma estrutura de dados, sendo o comportamento definido externamente ao modelo.

É importante destacar que, embora grande parte das pesquisas sobre modelos em tempo de execução esteja centrada em modelos arquiteturais causalmente conectados, a utilização de outros modelos sem conexão causal é frequente (GARLAN et al., 2004; MORIN; BARAIS, 2008; VOGEL; GIESE, 2012; HUBER et al., 2017). Nesse contexto, Vogel e Giese (2012) classificou os modelos utilizados em tempo de execução em duas categorias: modelos reflexivos e modelos de adaptação.

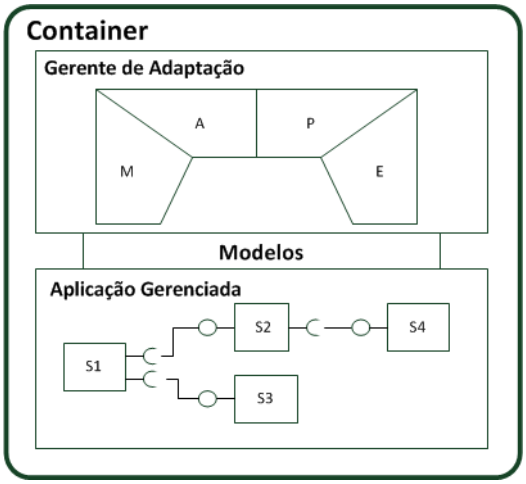
Os modelos reflexivos são modelos causalmente conectados, utilizados para fornecer ao gerente de adaptação conhecimento acerca da própria aplicação adaptável (i.e., *self-awareness* (HINCHEY; STERRITT, 2006)) e do ambiente no qual ela está inserida (i.e., *context-awareness* (PARASHAR; HARIRI, 2005)). Embora essenciais para a reconfiguração dinâmica, os modelos reflexivos não são suficientes para a construção de uma plataforma de apoio às aplicações auto-adaptativas verdadeiramente flexível.

De fato, embora esses modelos possam ser utilizados para promover a configuração e reconfiguração de uma aplicação adaptável, eles não possibilitam a configuração do próprio gerente de adaptação, uma vez que não possuem informação acerca de como ele deve realizar as atividades de supervisão e adaptação, que são centrais para a implementação do laço de controle. Cabe aos modelos de adaptação preencherem essa lacuna, representando, por exemplo, o espaço de configuração da aplicação, as restrições aplicáveis, e as regras que dirigem os processos de monitoração e adaptação. Desta forma, a seleção do conjunto de modelos utilizados para guiar o processo de adaptação é decisiva para determinar o nível de flexibilidade das diferentes plataformas.

Outro aspecto importante relacionado ao uso de modelos em tempo de execução diz respeito ao nível de abstração dos conceitos representados nesses modelos. Como vimos anteriormente, a utilização de modelos de alto nível como interface entre o gerente de adaptação e a aplicação adaptativa permite uma potencial redução na complexidade no código do próprio gerente e facilita o seu reuso em diferentes plataformas tecnológicas. Contudo, é importante observar que a capacidade de reuso de um gerente de adaptação não é somente definida pela utilização de modelos de alto nível.

De fato, para que essa reutilização seja possível, é necessário que o gerente de adaptação seja concebido como um elemento externo à própria aplicação. Considerando-se que a adaptação pode ser vista como um aspecto ortogonal aos requisitos funcionais das aplicações, é comum que, nas plataformas baseadas em componentes, os gerentes sejam embutidos em contêineres, os quais são naturalmente responsáveis por fornecer suporte aos requisitos não-funcionais e por gerenciar as aplicações em execução. A essência dessa

Figura 11 – Contêiner, gerente de adaptação, modelos, e aplicação



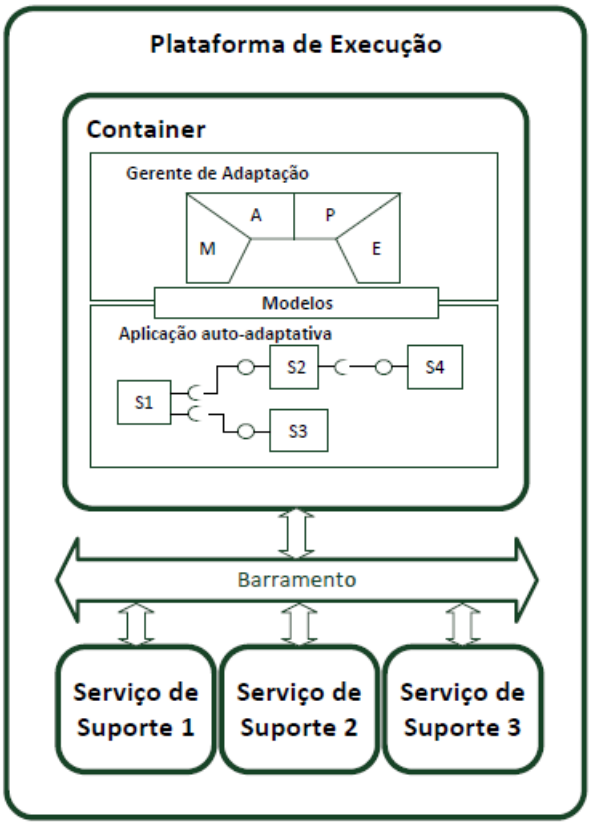
Fonte: o autor

abordagem está representada na **Figura 11**.

3.1.2 Contêineres e Serviços Técnicos

Embora os contêineres concentrem o gerenciamento dos requisitos não-funcionais, eles não atuam isoladamente. De fato, os diversos aspectos técnicos relacionados à implementação desses requisitos são tipicamente realizados através da utilização de um conjunto de serviços técnicos (e.g., serviços de nomes e diretórios, monitores de transação, serviços de controle de acesso, e assim por diante), os quais também integram um ambiente de execução (vide **Figura 12**). Em geral, a utilização desses serviços (também referenciados como serviços de suporte) é mediada pelos contêineres, os quais são configurados através de meta-dados informados pelos desenvolvedores e embutidos na aplicação através da utilização de técnicas de programação orientada a atributos (CONAN et al., 2001). A conexão entre um contêiner e os serviços técnicos pode ser realizada através da introdução de um barramento de serviços, que funciona como espinha dorsal de uma plataforma genérica de suporte às aplicações auto-adaptativas.

Figura 12 – Plataforma de execução



Fonte: o autor

3.2 AMBIENTE DE DESENVOLVIMENTO

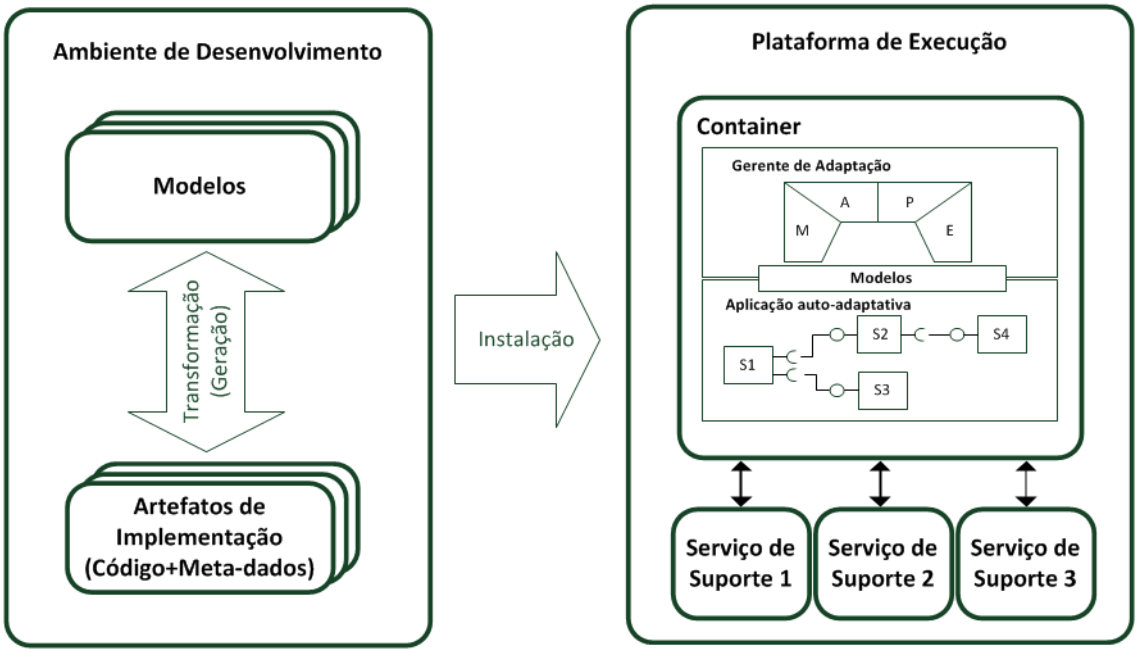
Como vimos, a responsabilidade pela condução do processo de adaptação nas aplicações auto-adaptativas é do gerente de adaptação, o qual realiza reconfigurações dinâmicas através do uso de modelos mantidos em tempo de execução. Nesse contexto, uma importante questão diz respeito ao processo de criação e povoamento desses modelos.

3.2.1 Modelos de Desenvolvimento e Modelos de Execução

A criação de modelos de execução pressupõe a disponibilidade de um conjunto de informações, as quais são necessárias para povoar esses modelos. Parte dessas informações são materializadas sob a forma de meta-dados advindos do tempo de desenvolvimento, os quais são embutidos no código das aplicações ou em arquivos de configuração. Cada plataforma possui seus próprios mecanismos de configuração e utiliza diferentes formas de descrever os meta-dados correspondentes. Neste contexto, os artefatos utilizados para descrever esses meta-dados são, em essência, modelos de desenvolvimento específicos de plataforma.

A geração manual desses modelos específicos implica na necessidade dos desenvolvedores conhecerem em detalhes a plataforma de execução. Mais ainda, as aplicações

Figura 13 – Ambiente de desenvolvimento



Fonte: o autor

desenvolvidas ficam fortemente vinculadas aos aspectos tecnológicos da plataforma subjacente, gerando a necessidade de um novo ciclo de desenvolvimento no caso de mudanças nessas plataformas. De fato, a necessidade de produzir artefatos específicos de plataforma leva os desenvolvedores a concentrarem a atenção no domínio da solução, desviando o foco do domínio do problema.

Uma abordagem interessante para amenizar esses problemas e reduzir a complexidade envolvida na atividade de desenvolvimento, consiste na utilização das técnicas de MDE. Em essência, essas técnicas visam reduzir a distância conceitual entre os domínios do problema e da solução (FRANCE; RUMPE, 2007) através da utilização de modelos abstratos de alto-nível e independentes de plataforma como entidades de primeira classe no processo de desenvolvimento (BÉZIVIN, 2006). Tais modelos não incorporam aspectos relativos ao domínio da solução computacional, sendo agnósticos com relação à plataforma tecnológica subjacente. Ao longo do processo de desenvolvimento, esses modelos abstratos são transformados em modelos específicos de plataforma através do uso de ferramentas de apoio, as quais devem ser parte integrante dos ambientes de desenvolvimento.

Por fim, em tempo de execução, os modelos específicos de plataforma são interpretados, dando origem aos modelos de execução que compõem a base de conhecimento utilizada pelos gerentes de adaptação. A abordagem descrita é resumida na **Figura 13**.

3.2.2 Modelos e Linguagens de Modelagem

Segundo Favre (2004), um modelo é uma representação abstrata de um sistema, criada com uma finalidade específica. Desta forma, um sistema pode e deve ser representado simultaneamente por múltiplos modelos, cada um representando uma visão/perspectiva particular. Neste contexto, os modelos assumem um papel importante, levando a uma separação natural de interesses, onde os atores projetam diversos modelos, provendo diferentes visões de um mesmo sistema.

De fato, adoção de múltiplos modelos em tempo de execução é comum nas plataformas de suporte às aplicações auto-adaptativas. Estes modelos são povoados a partir de modelos de desenvolvimento concebidos através do uso de linguagens (de modelagem) específicas de domínio, i.e., DSLs. Em geral, essas linguagens devem ser formalmente definidas, viabilizando a construção automatizada de ferramentas como editores com recursos de validação, transformadores de modelos, e geradores de código, as quais são integradas em ambientes de desenvolvimento e auxiliam a produção de modelos "corretos por definição". A definição de uma DSL envolve diferentes aspectos, os quais estão relacionados à sintaxe e à semântica da linguagem.

Do ponto de vista sintático, um primeiro aspecto a ser considerado diz respeito à definição da notação (gráfica ou textual) utilizada pelos usuários para a edição dos modelos. Essa notação corresponde à sintaxe concreta da linguagem. Outro aspecto fundamental na concepção de uma linguagem consiste na especificação da sua *sintaxe abstrata*. Neste contexto, a palavra "abstrata" indica que essa sintaxe é utilizada para representar "o quê" a linguagem expressa, sem determinar "como" essa informação deve ser representada. Em outras palavras, a sintaxe abstrata de uma linguagem é responsável por representar os elementos conceituais que integram a linguagem e as possíveis relações existentes entre eles. Em geral, esses elementos são organizados sob forma de uma árvore, referenciada como *árvore sintática*. Para que essa estrutura possa ser povoada, é necessária a especificação de um *mapeamento sintático*, definindo a relação entre os elementos que integram as sintaxes concreta e abstrata. Uma consequência importante da distinção entre esses aspectos sintáticos é que a estrutura dos modelos pode ser definida de forma independente da notação utilizada na sua edição.

Além dos aspectos sintáticos, a especificação de uma linguagem requer a definição da sua semântica, ou seja, a atribuição de significado aos elementos conceituais que compõem a sintaxe abstrata. Uma forma de conceber essa atribuição é definir um *mapeamento semântico*, que associa esses elementos conceituais aos elementos de um domínio semântico conhecido. A relação entre os elementos sintáticos e semânticos descritos é representada na **Figura 14**.

Adotando uma perspectiva mais pragmática, Voelter et al. (2013) considera a semântica sob dois aspectos: semântica estática e semântica de execução. A semântica estática é estabelecida através de restrições e de regras impostas pelo sistema de tipos, as quais

Figura 14 – Sintaxe e semântica

Fonte: baseada em (OVERBEEK, 2006)

devem ser respeitadas pelos modelos para que esses possam ser considerados válidos. Por sua vez, a semântica de execução está associada ao significado do modelo quando em execução, sendo embutida nos motores responsáveis pelo processamento desses modelos. Esse processamento pode ser implementado através de interpretação ou de tradução (compilação).

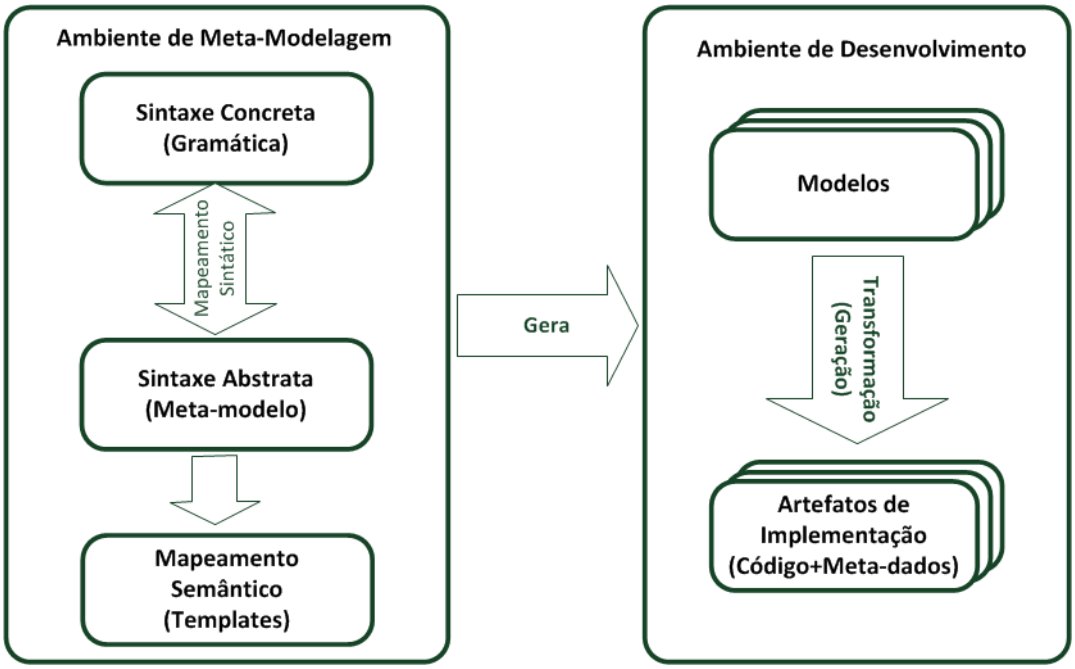
Em síntese, percebe-se que um aspecto relevante no contexto do desenvolvimento de aplicações baseadas em modelos em tempo de execução diz respeito à concepção das DSLs utilizadas pelos desenvolvedores para dar origem aos modelos de desenvolvimento, os quais são utilizados para povoar os modelos de execução. Para que essas DSLs possam ser efetivamente utilizadas, é importante a disponibilização de um conjunto de ferramentas de apoio integradas em um ambiente de desenvolvimento. Criar ambientes com esses recursos é uma tarefa complexa, sendo importante a utilização de ambientes próprios para a definição de linguagens e geração de ferramentas.

3.3 AMBIENTE DE META-MODELAGEM

Atualmente, diferentes ambientes de desenvolvimento de linguagens (JÉZÉQUEL; BARAIS; FLEUREY, 2011; COMBEMALE et al., 2014; BETTINI, 2016) se apresentam como opções maduras para a concepção e formalização de DSLs e para geração automatizada de ambientes de desenvolvimento como os descritos na seção anterior. Uma parte significativa desses ambientes se baseiam no conceito de meta-modelo (FAVRE, 2004) para definir os tipos de elementos que podem ser representados através dessas linguagens e os tipos de relacionamento que podem existir entre eles. Neste contexto, um *meta-modelo* pode ser descrito como um *modelo prescritivo* utilizado para definir uma linguagem de modelagem (GASEVIC; KAVIANI; HATALA, 2007).

De uma forma mais precisa, diz-se que um meta-modelo define a sintaxe abstrata e a semântica estática de uma linguagem (incluindo as restrições associadas ao sistema de tipos e à cardinalidade dos relacionamentos). Uma vez que os meta-modelos definem os tipos de elementos e relacionamentos válidos no contexto do uso de uma linguagem, eles podem ser utilizados para validar modelos, verificando se esses estão "em conformidade" com o meta-modelo correspondente. De fato, existe uma relação de instância entre um modelo válido e um meta-modelo, devendo o primeiro respeitar as restrições impostas pela semântica estática definida no segundo. Considerando-se o papel dos meta-modelos

Figura 15 – Ambiente de meta-modelagem e de desenvolvimento



Fonte: o autor

na definição de uma linguagem de modelagem, os ambientes de desenvolvimento dessas linguagens são muitas vezes referenciados como *ambientes de meta-modelagem*.

Embora essenciais, os meta-modelos não definem todos os aspectos de uma linguagem. De fato, para que uma linguagem possa ser efetivamente utilizada, é necessária a definição da sua sintaxe concreta e do mapeamento entre esta sintaxe e a sintaxe abstrata definida pelo meta-modelo. No caso de DSLs textuais, como as concebidas no âmbito do presente trabalho, é comum a utilização de gramáticas especificadas através de variações de Backus–Naur Form (BNF) para esse propósito. Mais ainda, alguns ambientes (e.g., xText (BETTINI, 2016)) são capazes de utilizar essas gramáticas para gerar os próprios meta-modelos, automatizando o mapeamento sintático.

Segundo Voelter et al. (2013), uma das formas de definir a semântica de uma linguagem é através da definição de um motor de execução responsável pela interpretação ou transformação do modelo (compilação). Neste contexto, os ambientes de meta-modelagem costumam oferecer recursos para a definição de *templates* capazes de realizar transformações automáticas de modelo para modelo ou de modelo para texto (normalmente código). Esse recurso é bastante explorado no contexto de MDE para transformar modelos independentes de plataforma em modelos específicos de plataforma, eventualmente incluindo arquivos de configuração e esqueletos de código. A relação entre os ambientes de meta-modelagem e de desenvolvimento é apresentada na **Figura 15**.

3.3.1 Níveis de Meta-Modelagem

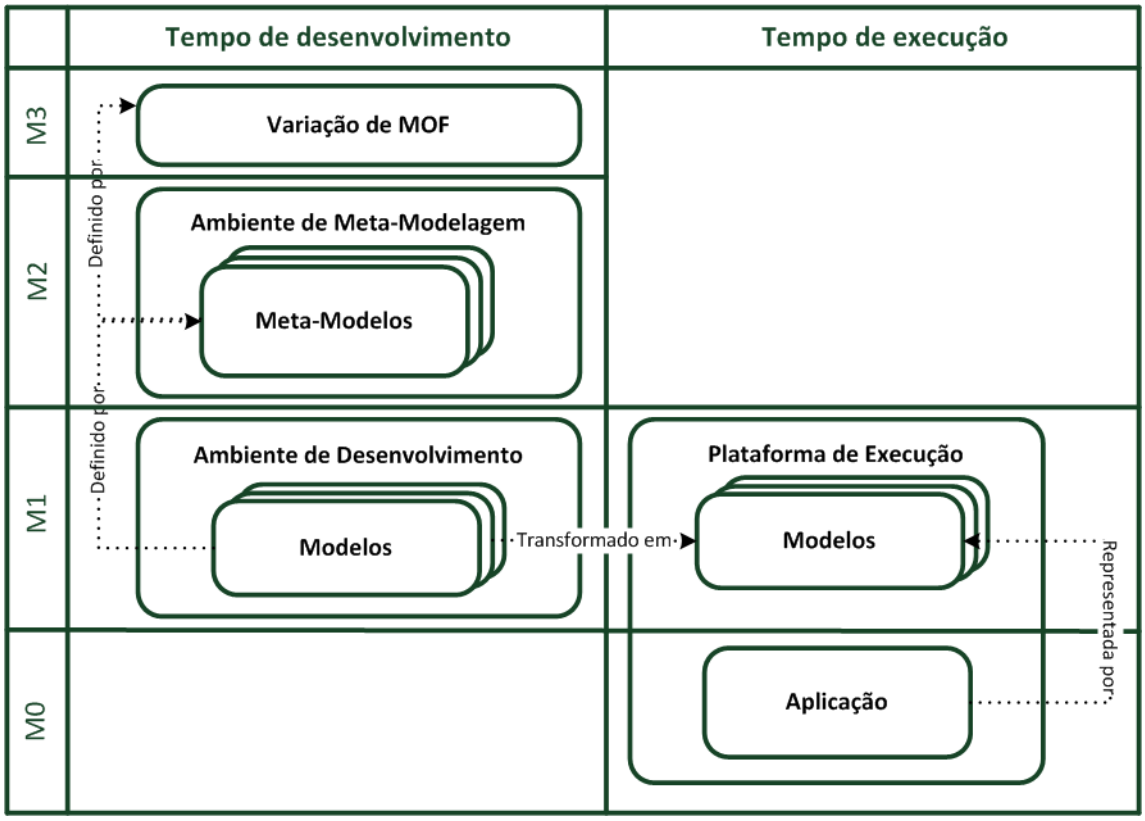
Diante da estrutura genérica proposta para representar as plataformas de suporte às aplicações auto-adaptativas, uma questão importante diz respeito ao posicionamento dos elementos dessas plataformas em relação aos níveis tradicionais de meta-modelagem. As abordagens de meta-modelagem costumam ser organizadas em torno de quatro níveis, rotulados como M0, M1, M2, e M3.

No nível M3, encontram-se as linguagens utilizadas nos ambientes de meta-modelagem para a definição de meta-modelos. É comum esses ambientes estarem fundamentados em variações de *Meta-Object Facility* (MOF) (OMG, 2015) (e.g., *Eclipse Modeling Framework* (EMF) (STEINBERG et al., 2009)). Os meta-modelos desenvolvidos nos ambientes de meta-modelagem compõem o nível M2. Esses meta-modelos definem a sintaxe abstrata e parte da semântica estática das DSLs. Os modelos concebidos nos ambientes de desenvolvimento com base nas DSLs são instâncias dos meta-modelos definidos em M2 e integram o nível M1 da arquitetura de meta-modelagem. O nível M0 é composto pelos objetos conceituais do mundo real representados através dos modelos em M1.

No processo de instalação de uma aplicação, os modelos concebidos em tempo de desenvolvimento são interpretados pela plataforma de execução. Uma vez que esses modelos podem ser distintos daqueles utilizados em tempo de execução, o processo de interpretação pode compreender uma transformação de modelos. Neste cenário, ambos os modelos (desenvolvimento e execução) integram o nível M1 da arquitetura de meta-modelagem. Por fim, o nível M0 é composto pela aplicação "viva" na plataforma de execução, a qual é representada pelos modelos em tempo de execução.

É importante destacar, que a utilização de meta-modelos em tempo de execução não é mandatória. Embora a utilização desses meta-modelos permita a adoção de ferramentas automáticas de transformação de modelos, ela introduz dependências entre o ambiente de execução e as APIs utilizadas para manipulação dos conceitos definidos no meta-modelo. Como discutido em Fouquet et al. (2014), essas dependências podem representar um alto custo em termos de tamanho da plataforma e, consequentemente, impor a necessidade de mais recursos nas plataformas de execução. Neste contexto, uma abordagem alternativa consiste em embutir nas plataformas (em particular, nos ambientes de execução) um conjunto de classes que representam os tipos de elementos conceituais que podem ser utilizados nos modelo de nível M1. Uma característica particularmente interessante dessa abordagem é viabilizar a construção de modelos ricos, incluindo dados e comportamentos, sendo esses últimos realizados através dos métodos definidos nas classes. A **Figura 16** posiciona os elementos de uma plataforma de suporte às aplicações auto-adaptativas com relação aos níveis de meta-modelagem.

Figura 16 – Níveis de Meta-Modelagem



Fonte: o autor

3.4 CONSIDERAÇÕES FINAIS

Nas aplicações auto-adaptativas, os gerentes de adaptação são responsáveis por conduzir o processo de reconfiguração. Uma abordagem promissora para a construção desses gerentes consiste em fazer com que eles desempenhem suas atividades com o auxílio de modelos em tempo de execução, os quais, além de armazenar os dados necessários para subsidiar a tomada de decisão, funcionam como interfaces de gerenciamento. Idealmente, esses modelos devem ser de alto nível e independentes de plataforma, possibilitando a construção de soluções mais simples e portáveis.

Para que modelos possam ser utilizados em tempo de execução, eles devem ser inicialmente povoados a partir de dados advindos do tempo de desenvolvimento. Em geral, esses dados são especificados sob a forma de modelos de desenvolvimento independentes de plataforma. Uma vez concebidos, esses modelos de desenvolvimento passam por processos de transformação, visando a obtenção de modelos específicos de plataforma, os quais podem ser posteriormente interpretados para povoar os modelos em tempo de execução.

A concepção de modelos de desenvolvimento requer a utilização de linguagens de modelagem. Neste contexto, ambientes de desenvolvimento capazes de “entender” essas linguagens e apoiar o desenvolvedor na concepção de modelos “corretos” são importantes. Em geral, esses ambientes costumam integrar ferramentas tais como: editores, validadores,

transformadores de modelos e geradores de código.

A despeito da sua utilidade, a construção manual de ambientes de desenvolvimento é uma atividade bastante complexa. Em um cenário ideal, as ferramentas que integram esses ambientes são geradas automaticamente a partir da especificação das linguagens de modelagem suportadas por eles. Atualmente, essa automatização é possível a partir da utilização de ambientes de meta-modelagem, os quais viabilizam a concepção de DSLs através da definição de meta-modelos e de gramáticas.

Uma vez que os principais conceitos relacionados à adaptação dinâmica foram apresentados e organizados em uma plataforma genérica de suporte às aplicações auto-adaptativas, é possível discutir como eles podem ser utilizados na proposição de uma plataforma de suporte às QSBAs, na qual o gatilho para o processo de adaptação está centrado na qualidade de serviço. Antes disso, porém, é necessário identificar as principais lacunas no estado da arte, as quais motivaram a proposição da plataforma DSOA. Neste contexto, o capítulo seguinte avalia o estado da arte através de um estudo amplo das soluções atualmente disponíveis.

4 TRABALHOS RELACIONADOS

> *Wise men learn by other men's mistakes.*
>
> H. G. Wells,

A concepção de aplicações auto-adaptativas baseadas em atributos de qualidade é um tema complexo e vem sendo alvo de diversas pesquisas. Essa complexidade está relacionada a duas características principais. De um lado, está a própria natureza das aplicações auto-adaptativas, as quais devem se adaptar sem interferência direta de seres humanos. De outro lado, está a natureza do aspecto de qualidade, que é ortogonal à funcionalidade das aplicações.

Visando delimitar o universo de trabalhos relacionados e organizar a sua apresentação, este capítulo utilizou as duas características mencionadas como critério de classificação. Como ponto de partida, apresentamos um conjunto de plataformas de suporte às aplicações auto-adaptativas, destacando suas principais características. Na sequência, apresentamos alguns meta-modelos de qualidade utilizados no contexto de serviços. Em particular, a intenção é identificar se os conceitos representados nesses meta-modelos são suficientes para a proposição de uma plataforma de suporte às aplicações auto-adaptativas baseadas em serviço e cientes de qualidade.

4.1 PLATAFORMAS DE SUPORTE ÀS APLICAÇÕES AUTO-ADAPTATIVAS

O desenvolvimento de sistemas auto-adaptativos é um tópico de pesquisa bastante amplo e complexo, que normalmente envolve diferentes áreas de conhecimento (CHENG et al., 2009; LEMOS et al., 2013). Essa amplitude representa um grande desafio para a seleção dos trabalhos que devem ser avaliados para identificar lacunas no estado da arte. Dentre as várias pesquisas analisadas, um ponto comum frequentemente encontrado é a utilização de modelos arquiteturais (OREIZY et al., 1999; GARLAN et al., 2004; KRAMER; MAGEE, 2007; VOGEL; GIESE, 2010).

Neste contexto, o universo dos trabalhos avaliados foi definido com base na adoção de modelos na condução do processo de adaptação. Na avaliação de cada trabalho, procuraremos identificar: (1) quais os gatilhos para o processo de adaptação, (2) como e onde a adaptação é descrita, (3) se é possível a definição de novas características de qualidade, e (4) se os desenvolvedores podem especificar como as métricas de qualidade podem ser computadas.

4.1.1 Rainbow

Rainbow (GARLAN et al., 2004; GARLAN; SCHMERL; CHENG, 2009) é uma plataforma de suporte às aplicações auto-adaptativas na qual as decisões relativas ao processo de adap-

tação são tomadas com base em modelos arquiteturais mantidos em tempo de execução, os quais representam os componentes e conectores das aplicações, e as restrições arquiteturais aplicáveis.

A plataforma Rainbow utiliza *probes* para coletar informação acerca das aplicações em execução, e *gauges* para agregar e abstrair as informações coletadas. A infraestrutura de monitoração baseada em *probes* e *gauges* possui diversas características importantes. A proposição desses dois elementos leva a uma separação natural entre a monitoração e a interpretação, permitindo diferentes interpretações para um mesmo conjunto de observações. Uma vez que os *gauges* são desacoplados dos *probes* a partir de um mecanismo de eventos, eles podem ser realizados em ambientes de execução separados, não sobrecarregando o ambiente no qual o sistema está efetivamente em execução. Mais ainda, a informação extraída de diferentes *gauges* pode ser combinada para derivar novas informações.

As informações coletadas e processadas são utilizadas para atualizar os modelos arquiteturais e de ambiente que suportam o mecanismo de adaptação. Atualizados os modelos, as restrições arquiteturais são verificadas para detectar uma eventual necessidade de adaptação. Caso uma adaptação seja necessária, uma estratégia de adaptação escrita em uma linguagem própria, denominada Stitch (CHENG; GARLAN, 2012), é selecionada com base em uma função de utilidade e acionada. Essa estratégia realiza a adaptação no sistema alvo utilizando um conjunto de efetuadores.

A despeito dos pontos positivos apresentados acima, a solução proposta também possui limitações. Em primeiro lugar, é importante mencionar que os *gauges* precisam ser implementados para processar as observações recebidas. Nesse contexto, a plataforma apenas fornece um conjunto de interfaces a serem utilizadas pelos desenvolvedores na implementação desses elementos. O mesmo acontece com relação aos *gauges consumers*, os quais devem ser implementados para receber a informação gerada pelos *gauges* e atualizar os modelos.

Em suma, a adaptação é disparada em função de violações às restrições estabelecidas nos modelos arquiteturais e realizada através de *scripts* escritos em uma linguagem imperativa própria. A plataforma suporta novas características de qualidade através da introdução de *probes* e *gauges*, os quais devem ser definidos de forma programática. Esse mesmo mecanismo também define como as métricas devem ser computadas.

Na plataforma DSOA, os agentes de processamento são responsáveis pela interpretação dos dados obtidos a partir dos sensores. Estes agentes são descritos utilizando-se modelos declarativos escritos em uma linguagem própria. Mais ainda, os agentes podem ser reconfigurados dinamicamente a partir da atualização dos modelos mantidos em execução. Assim, a plataforma DSOA permite que os desenvolvedores possam trabalhar em um nível de abstração mais elevado. A proposição de uma linguagem declarativa própria para a especificação de agentes permite também uma maior especialização dos perfis de

desenvolvedores.

4.1.2 MADAM

MADAM (*Mobility and ADaptation-enAbling Middleware*) (FLOCH et al., 2006) é uma plataforma de *middleware* que suporta a adaptação das aplicações através do uso de modelos arquiteturais, que descrevem as aplicações como composições formadas por componentes conectados através de portas. Além da estrutura da aplicação, esses modelos descrevem as possíveis variações e funções de utilidade, as quais permitem selecionar a variante que deve ser utilizada.

Em tempo de execução, MADAM utiliza dois tipos de modelos arquiteturais em níveis ontológicos distintos. No nível de tipos, uma aplicação é representada em termos dos tipos de componentes utilizados, os quais representam os pontos de variação. Os modelos de tipo são analisados juntamente com as variantes (que representam as possíveis implementações de componentes) para montar a arquitetura do sistema que deve ser utilizada em execução, a qual é representada através de um modelo arquitetural em nível de instância.

As análises são baseadas em técnicas de planejamento, sendo realizadas sempre que a aplicação é instalada e quando há mudança de contexto. Durante o planejamento, a plataforma avalia as variantes calculando a função de utilidade de cada uma no atual contexto. A arquitetura atual, representada pelo modelo em nível de instância, também é avaliada. Caso a função de utilidade de uma variante seja melhor que a do sistema em execução, o processo de reconfiguração é disparado. Esse processo envolve trazer os componentes para um estado seguro, substituir os componentes e recuperar o estado.

Em síntese, as adaptações são disparadas a partir da identificação de mudanças no contexto. Neste ponto, a lógica de adaptação (pré-definida na plataforma) computa funções de utilidades associadas aos componentes que podem ser utilizados na aplicação para identificar uma eventual necessidade de adaptação. Não há possibilidade de definição de novas características de qualidade. Assim, embora a plataforma possua algumas semelhanças com a plataforma DSOA, uma vez que utiliza modelos arquiteturais de tipos e de instâncias para conduzir a adaptação, a plataforma MADAM não permite a definição de modelos customizados de qualidade, nem mecanismos de monitoração configuráveis de forma declarativa. Observa-se também que a plataforma não permite que as aplicações utilizem novos componentes que não sejam de conhecimento prévio da aplicação, limitando severamente a capacidade de reconfiguração dinâmica.

4.1.3 MUSIC

A plataforma MUSIC (HALLSTEINSEN et al., 2012) é a sucessora de MADAM e foi proposta visando contemplar uma visão de mundo aberto (BARESI; Di Nitto; GHEZI, 2006; Di Nitto et al., 2008). Nesse contexto, a plataforma assumiu como alvo os ambientes computacionais caracterizados pela heterogeneidade e pela necessidade de descoberta e ligação

dinâmicas. Na plataforma MUSIC, os componentes das aplicações podem ser substituídos por serviços, os quais são selecionados com base em características não-funcionais. A plataforma conta com diferentes mecanismos de reconfiguração, incluindo a adaptação paramétrica e composicional, através da substituição de componentes e serviços.

As características autonômicas da plataforma advém da utilização do padrão arquitetural MAPE-K, sendo a base de conhecimentos composta por um modelo arquitetural. Na plataforma MUSIC, os modelos arquiteturais descrevem uma aplicação em termos abstratos, como uma composição de funcionalidades expressas através de um conjunto de tipos de papéis, os quais podem ser realizados por componentes ou serviços.

As portas possuem propriedades e uma função de predição, a qual especifica como as métricas de qualidade podem ser computadas e quais recursos são necessários. As funções de predição são definidas como expressões sobre as propriedades do contexto e dos componentes utilizados para realizar os papéis. Por fim, o processo de adaptação é dirigido por uma função de utilidade, a qual é utilizada para avaliar as diferentes configurações.

Do ponto de vista de modelagem, a plataforma MUSIC propõe um perfil UML que define estereótipos e rótulos, os quais são utilizados para especificar a arquitetura das aplicações, os pontos de variação, as propriedades de componentes e serviços, e as dependências de contexto.

A despeito da grande flexibilidade dos mecanismos de adaptação disponibilizados pela plataforma, uma importante limitação diz respeito à impossibilidade de definição dinâmica de novas métricas de qualidade, de forma que o universo de métricas utilizadas é previamente estabelecido. Como mencionamos, a plataforma DSOA permite a definição de métricas de qualidade de forma declarativa e dinâmica. Mais ainda, os desenvolvedores podem especificar como essas métricas devem ser coletadas a partir da declaração de agentes responsáveis pelo processamento de eventos primitivos. Para definir métricas e agentes, a plataforma DSOA oferece linguagens declarativas próprias, suportadas por um ambiente de desenvolvimento.

4.1.4 SAFRAN

SAFRAN (DAVID et al., 2009; DAVID; LEDOUX, 2006) é uma extensão do modelo de componentes Fractal proposta com o intuito de fornecer uma plataforma dinâmica que suporta aplicações auto-adaptativas baseadas em contexto. SAFRAN propõe que a adaptação seja tratada através do conceito de aspecto, promovendo uma separação natural entre código de negócio e código de adaptação. Em particular, a solução se baseia no conceito de "orientação a aspectos baseada em eventos", a qual utiliza eventos como *pointcuts* que indicam quando uma política de adaptação (*advice*) deve ser disparada.

Em SAFRAN, os componentes de uma aplicação são gerenciados por um gerente de adaptação, o qual é configurado através da definição de uma política de adaptação escrita em *FScript*, uma linguagem procedural proposta para manipular componentes Fractal.

FScript suporta a navegação nos elementos de uma aplicação e permite a realização das operações de reconfiguração normalmente oferecidas pelo modelo Fractal, incluindo a adição e remoção de instâncias de componentes, e a substituição desses elementos nas aplicações em execução. Por fim, os eventos que disparam a adaptação podem ser eventos internos (como a invocação de métodos do componente) ou externos (como a alocação de memória).

De forma similar à plataforma DSOA, SAFRAN se baseia em um modelo arquitetural para realizar a adaptação, sendo esse processo disparado por eventos. Do ponto de vista das regras de adaptação, SAFRAN possui um modelo mais elaborado do que aquele implementado na plataforma DSOA, definindo uma linguagem própria para a especificação de políticas de adaptação.

Contudo, SAFRAN não prevê como os eventos primitivos podem ser processados, não sendo possível a definição de eventos de mais alto nível. Mais ainda, não há a representação explícita do conceito de qualidade, ficando o desenvolvedor limitado a especificar a adaptação através de *scripts* de baixo nível disparados por eventos primitivos da plataforma.

4.1.5 Capucine

Capucine (PARRA et al., 2011; PARRA; BLANC; DUCHIEN, 2009) é uma plataforma baseada em *linhas de produto de software dinâmicas* para a construção de aplicações auto-adaptativas baseadas em serviço e sensíveis ao contexto. A plataforma propõe que uma composição seja concebida como um modelo de *features*, o qual é utilizado para orientar a montagem da aplicação. Durante o processo de montagem, cada *feature* é mapeada para um "modelo de aspecto", que é "costurado" no modelo de *features* representando a aplicação em tempo de desenvolvimento ou de execução. Para que os modelos possam ser especificados, a plataforma define um conjunto de meta-modelos.

Capucine tem algumas similaridades com a plataforma DSOA. Em primeiro lugar, ambas as propostas são baseadas em modelos de componentes orientados a serviço, o que significa que as aplicações são modeladas utilizando-se os mesmos elementos conceituais (e.g. componentes, serviços, e interfaces). Desta forma, o meta-modelo de aplicação utilizado em Capucine apresenta conceitos similares aos utilizados no meta-modelo da plataforma DSOA. Em segundo lugar, ambas as plataformas oferecem apoio ao desenvolvimento das aplicações através da definição de um conjunto de linguagens de modelagem. Por fim, as duas plataformas fornecem uma infraestrutura de execução capaz de realizar a adaptação dinâmica.

A despeito das similaridades, há importantes diferenças. Enquanto os modelos da plataforma DSOA são mantidos em tempo de execução e utilizados para dirigir o processo de adaptação, os modelos de Capucine são utilizados para gerar *scripts* de reconfiguração. Uma outra diferença fundamental diz respeito ao foco dos trabalhos. A plataforma

Capucine aborda as aplicações sensíveis ao contexto, não oferecendo facilidades para a definição de novas características de qualidade. Por outro lado, a plataforma DSOA foi proposta com o objetivo de suportar as aplicações auto-adaptativas baseadas em serviço e cientes de qualidade, de forma que o nível de qualidade de serviço possui um papel central.

4.1.6 SASSY

SASSY (*Self-Architecting Software Systems*) (MALEK et al., 2011; ESFAHANI et al., 2009) é uma plataforma que visa suportar a adaptação dinâmica de sistemas orientados a serviços em função de requisitos não-funcionais. Esses sistemas são representados em tempo de execução através de modelos arquiteturais responsáveis por dirigir o processo de adaptação, o qual pode ser disparado em função de modificações nos requisitos ou de variações na qualidade de serviço. Quando esse processo é disparado, as adaptações são realizadas através de modificações no modelo correspondente visando, assim, garantir a satisfação dos requisitos não-funcionais.

Em SASSY, um especialista no domínio especifica um sistema através de uma linguagem gráfica de modelagem orientada a atividades, referenciada como *Service Activity Schemas* (SAS) (ESFAHANI et al., 2009). A linguagem SAS é definida através de um metamodelo que pode ser dividido em três partes. A primeira delas representa o comportamento do sistema através de um grafo contendo atividades, serviços e fluxos de dados entre eles. A segunda permite a especificação dos serviços e de suas interfaces definidas em função dos dados de entrada e de saída. Por fim, a terceira é utilizada para a definição das métricas de qualidade. Os modelos gerados em SAS representam os serviços que compõem o sistema e as interações entre eles. Os requisitos não-funcionais são representados através de funções utilidade, as quais se baseiam nas métricas de qualidade e são avaliadas ao longo da execução para identificar a necessidade de adaptação.

As possíveis variações na arquitetura correspondem a padrões arquiteturais, os quais são associados a modelos analíticos parametrizados representando a influência do uso desses padrões nas métricas de qualidade. Para guiar o processo de adaptação, um arquiteto deve especificar padrões de adaptação, os quais são responsáveis por efetivar as modificações arquiteturais necessárias ao longo da execução.

A partir da especificação da aplicação e dos padrões arquiteturais, a plataforma deriva a arquitetura do sistema em termos dos tipos de serviço necessários, e um modelo de coordenação que regula a interação entre os serviços realizando o fluxo modelado na linguagem visual. Por fim, os tipos de serviço e as informações de qualidade são utilizados para a descoberta dinâmica dos serviços através da busca em um registro.

Durante a execução, uma aplicação é gerenciada por um gerente de adaptação que implementa o laço MAPE-K. Para tomar as decisões acerca da necessidade da adaptação, esse gerente utiliza as funções de utilidade especificadas. Quando da necessidade de

adaptação, os padrões de adaptação são aplicados de forma a gerar, dinamicamente, uma nova arquitetura.

De forma similar à plataforma DSOA, SASSY visa apoiar o desenvolvimento e execução de aplicações auto-adaptativas baseadas em serviço e cientes de qualidade. A despeito das similaridades conceituais entre as propostas, uma diferença central diz respeito às métricas de qualidade suportadas. De fato, embora os autores mencionem que o conjunto de métricas da plataforma SASSY não é limitado, não identificamos no meta-modelo proposto nenhuma forma de indicar como as métricas de qualidade podem ser efetivamente computadas, não sendo possível definir a semântica de novas métricas. Essa dificuldade é tratada na plataforma DSOA através da possibilidade de utilização de modelos para definir dinamicamente agentes de processamento, os quais são responsáveis por computar os valores das métricas a partir de dados obtidos de eventos primitivos lançados pelos sensores da plataforma.

4.1.7 EUREMA

Vogel et al. (2010), Vogel e Giese (2010) propõe que a adaptação de aplicações em execução seja conduzida a partir de modelos arquiteturais mantidos em tempo de execução. Para tratar a diferença conceitual entre os elementos dos domínios do problema e da solução, os autores propõem o uso de transformações de modelos. A ideia central é que os gerentes de adaptação utilizem modelos mais abstratos, os quais não devem transparecer detalhes de implementação. Em particular, os autores defendem a necessidade de utilização de diferentes modelos, cada um representando uma perspectiva sob a qual a aplicação em execução pode ser avaliada.

Na proposta, os modelos abstratos são obtidos a partir de modelos de baixo nível e dependentes de plataforma, referenciados como modelos-fonte, os quais possuem uma conexão causal com a aplicação em execução. Nesse contexto, o ambiente de execução é responsável por garantir a sincronia entre os modelos abstratos e o modelo fonte, fazendo com que os gerentes tenham acesso às informações atualizadas. Para implementar as ideias propostas, os autores desenvolveram um motor de transformação baseado em EMF (STEINBERG et al., 2009) e *Triple Graph Grammar* (TGG).

Outra contribuição importante foi a identificação da necessidade de utilização do conceito de *megamodelos* em tempo de execução (VOGEL; SEIBEL; GIESE, 2011; VOGEL; SEIBEL; GIESE, 2010). Um *megamodelo* é basicamente um modelo que representa outros modelos (e meta-modelos) e as relações entre eles. Nesse contexto, propõe-se que os *megamodelos* sejam utilizados para representar as relações entre os diferentes modelos que compõem uma base de conhecimento. Para estruturar suas ideias, os autores identificaram e classificaram os tipos de modelos que são normalmente utilizados na implementação de sistemas auto-adaptativos, independentemente de possuírem ou não uma conexão causal com o sistema em execução.

Além de classificar os modelos, a pesquisa identificou as relações que tipicamente existem entre eles, as quais são vistas como atividades que utilizam modelos como entrada e produzem modelos como saída. Neste contexto, um *megamodelo* é exatamente uma representação dessas atividades e dos modelos requeridos e produzidos por elas. Considerando-se a relevância das informações contidas em um *megamodelo*, os autores defendem que eles devem ser mantidos em tempo de execução e utilizados para "programar" gerentes de adaptação. Para representar os *megamodelos*, é proposta uma linguagem de modelagem, referenciada como *ExecUtable RuntimE MegAmodels* (EUREMA) (VOGEL; GIESE, 2013).

As pesquisas mencionadas tiveram uma grande influência na plataforma DSOA por indicar a necessidade de diferentes tipos de modelos e as possíveis relações entre eles. Contudo, não é objetivo de EUREMA suportar a adaptação baseada em métricas de qualidade. Desta forma, aspectos como a definição dinâmica de métricas de qualidade e de algoritmos de computação de métricas não fazem parte de seu escopo.

4.1.8 Descartes

De forma similar à plataforma DSOA, um grupo de pesquisa da Alemanha propôs uma plataforma composta por um ambiente de desenvolvimento e um ambiente correspondente de execução visando suportar a adaptação dinâmica de aplicações baseadas em linguagens de modelagem e modelos arquiteturais mantidos em execução (HUBER et al., 2014b; BROSIG; HUBER; KOUNEV, 2014). Na plataforma proposta, as aplicações também são vistas como composições estruturais de serviços baseadas em atributos de qualidade.

Um diferencial deste trabalho é a capacidade de realização de adaptações com base em predições realizadas em tempo de execução, as quais são realizadas através do uso de modelos estocásticos (redes de filas).

Para viabilizar a adaptação, a plataforma conta, além do modelo arquitetural, com modelos de apoio representando as características de qualidade e as estratégias e táticas que podem ser utilizadas pelo gerente na condução do processo de adaptação automática.

A despeito da grande relevância do trabalho, que ocupa um papel de destaque no estado da arte, uma limitação importante diz respeito à definição das métricas de qualidade. A partir dos elementos propostos, a plataforma não é capaz de permitir que as aplicações indiquem como as métricas de qualidade podem ser monitoradas em execução. De fato, a plataforma assume que o ambiente de execução dispõe de sensores, os quais são responsáveis por produzir, diretamente, os valores para as métricas de qualidade. A plataforma DSOA, por outro lado, entende que os dados extraídos dos eventos primitivos são de baixo nível, e devem ser processados para que informações referentes às métricas de qualidade possam ser efetivamente computadas.

4.1.9 iObserve

A plataforma iObserve (HEINRICH et al., 2017; HEINRICH, 2016) propõe uma solução integrada para apoiar o desenvolvimento e execução de aplicações auto-adaptativas. Nesta plataforma, o processo de adaptação se baseia em um modelo arquitetural construído com base no modelo de componentes Palladio (*Palladio Component Model* (PCM)) (REUSSNER et al., 2016), o qual é utilizado para fins de predição de desempenho. Em tempo de execução, o modelo arquitetural em Palladio é alimentado a partir de dados coletados com a ferramenta Kieker.

Para viabilizar essa solução, o primeiro passo é definir como uma aplicação deve ser instrumentada de forma que os dados necessários para alimentar o modelo possam ser coletados em tempo de execução. Para esse fim, a plataforma introduziu uma linguagem de instrumentação baseada em aspectos, referenciada como *Instrumentation Aspect Language* (IAL), a qual faz referência aos elementos do modelo arquitetural. Para estruturar os dados coletados, a plataforma propôs uma outra linguagem, referenciada como *Instrumentation Record Language* (IRL), a qual permite a declaração de estruturas de dados de forma independente de plataforma. Em suma, modelos construídos com base em IAL e IRL são utilizados para configurar o mecanismo de monitoração, que coleta dados que alimentam o modelo arquitetural utilizado para predição. Por fim, os resultados da predição são utilizados pelos gerentes de adaptação para reconfigurar dinamicamente a aplicação em execução.

Embora a plataforma iObserve permita que os desenvolvedores especifiquem os dados que devem ser monitorados para alimentar o modelo arquitetural utilizado para predição de desempenho, essa plataforma não aborda, de forma explícita, a diferença de nível de abstração entre os dados coletados pelo mecanismo de monitoração e aqueles necessários para povoar um modelo arquitetural de alto nível. Como mencionado, essa diferença é abordada na plataforma DSOA através da definição de agentes de processamento, os quais são especificados de forma declarativa. Mais ainda, a plataforma DSOA trabalha com modelos arquiteturais ricos, os quais, além de expor o estado atual da aplicação em execução, oferecem uma interface que pode ser utilizada pelo gerente para conduzir a adaptação.

4.1.10 iCasa

Lalanda e Escoffier (2017) propõe uma plataforma interessante para suportar aplicações auto-adaptativas em ambientes pervasivos, denominada iCasa. De forma similar à plataforma DSOA, iCasa aproveitou as características de iPojo para construir um suporte para aplicações auto-adaptativas baseadas em serviços descobertos e conectados dinamicamente. O processo de adaptação é conduzido a partir de um modelo genérico baseado no conceito de recursos, o qual pode ser facilmente estendido. A plataforma proposta é flexível

e suporta uma visão de mundo aberto, segundo a qual novos elementos de uma aplicação podem surgir dinamicamente. Contudo, a plataforma não propõe uma representação para as características de qualidade, nem aborda como essas características poderiam ser utilizadas para conduzir o processo de adaptação.

4.2 META-MODELOS DE QUALIDADE

Como vimos na Seção 2.4.2, uma infraestrutura SOA flexível deve ser capaz de suportar diferentes modelos de qualidade, os quais devem ser extensíveis, permitindo a definição de novos atributos e métricas. Mais ainda, esses modelos de qualidade e as QSDs, utilizadas para caracterizar a qualidade dos serviços providos e requeridos por uma QSBA, devem incorporar informações suficientes para apoiar a infraestrutura, permitindo que ela gerencie todas as etapas que compõem o ciclo de vida das aplicações. Como vimos, uma solução para viabilizar a concepção dessas infraestruturas é basear a sua implementação em conceitos definidos através de meta-modelos, compreendendo os domínios de serviço e qualidade, os quais são referenciados como SQMMs.

Neste sentido, uma abordagem para avaliar o estado da arte no que diz respeito aos meta-modelos de qualidade deve compreender duas perspectivas distintas. De um lado, propõe-se avaliar os principais meta-modelos de serviços e qualidade atualmente disponíveis, visando verificar se os modelos concebidos em conformidade com esses meta-modelos contém informação suficiente para o gerenciamento do ciclo de vida das aplicações. De outro, propõe-se avaliar as infraestruturas de suporte às QSBAs, com um foco especial na riqueza das informações utilizadas para caracterizar a qualidade dos serviços suportados por essas infraestruturas.

4.2.1 Web Service Level Agreement (WSLA)

A proposta de *Web Service Level Agreement* (WSLA) (KELLER; LUDWIG, 2003) compreende tanto a linguagem para descrição de qualidade quanto a arquitetura de tempo de execução capaz de suportá-la. Do ponto de vista da linguagem, WSLA oferece a possibilidade de descrição detalhada e customizada das características de qualidade de serviço, as quais são centrais para viabilizar o estabelecimento de SLAs entre consumidores e fornecedores de serviço. Neste contexto específico, a proposta de WSLA vai além do que é suportado na plataforma DSOA, uma vez que esta última não tem a pretensão de suportar o conceito de SLA e as atividades de gerenciamento correspondentes.

Em WSLA, a qualidade de um serviço é descrita através da definição de um conjunto de parâmetros de SLA, cada um representando uma propriedade associada à qualidade do serviço (e.g., disponibilidade, *throughput*, tempo de resposta). Na linguagem definida por WSLA, um parâmetro é definido através de um nome, um tipo, uma unidade e uma métrica a partir da qual o seu valor pode ser obtido. Por sua vez, o valor de uma métrica

depende da natureza da mesma. As métricas compostas são computadas a partir da aplicação de uma função (e.g., soma, divisão, média, valor máximo, valor mínimo, e assim por diante) sobre os valores obtidos de outras métricas. Um exemplo de métrica composta é o tempo de resposta, o qual pode ser computado a partir da soma da latência com o tempo de execução. Por outro lado, as métricas de recurso tem seu valor especificado através de diretivas de medição, as quais indicam como o valor correspondente pode ser diretamente recuperado a partir de um recurso gerenciado, e.g., através da indicação de uma URI de um serviço de gerenciamento. Como exemplos de métricas de recurso, pode-se citar o tempo total de operação de um serviço e o número de invocações ao mesmo.

Por fim, os parâmetros de SLA são utilizados no estabelecimento de *Service Level Objectives* (SLOs), os quais expressam o comprometimento de manter um dado nível de qualidade de serviço dentro de um período estabelecido. Um SLO é definido a partir dos seguintes elementos:

- *Obliged*: define a parte responsável por manter a qualidade definida;
- *ValidityPeriods*: define quais períodos são compreendidos pela garantia;
- *Expression*: uma expressão lógica que define o que é efetivamente garantido. Uma expressão é definida através dos conectores lógicos (and, or, not), os quais conectam predicados e expressões. Por sua vez um predicado é composto por um operador de comparação (maior que, menor que, igual, etc.), um parâmetro e um valor limite. Cada predicado resulta em um valor lógico, verdadeiro ou falso.

WSLA é uma das propostas mais antigas no contexto de qualidade de serviço para serviços Web, apresentando um elevado grau de extensibilidade e flexibilidade. A despeito da sua grande relevância, alguns pontos importantes não são contemplados na proposta. Do ponto de vista da descoberta e seleção de serviços, a proposta não prevê mecanismos para especificar políticas de seleção de serviços ou prioridades associadas às características não funcionais, limitando suas possibilidades de uso nesses cenários. Do ponto de vista da monitoração, a proposta se baseia na utilização de métricas obtidas a partir de recursos externos referenciados através de URIs, de forma que o desenvolvedor não é capaz de especificar como essas métricas devem ser computadas.

É importante destacarmos que, no contexto de WSLA, essas limitações são “naturais”, uma vez que a proposta se concentra no estabelecimento e gerenciamento de contratos, sem uma preocupação maior com a descoberta, seleção, e monitoração de serviços, que compõem a parte central do presente trabalho de doutorado. Considerando-se essa diferença de escopo, os trabalhos centrados no gerenciamento de contratos não são analisados na sequência deste documento.

4.2.2 Quality-Value-Dependency-Priority (QVDP)

A proposta do modelo *Quality-Value-Dependency-Priority* (QVDP) (JURETA; HERSSENS; FAULKNER, 2009) foi resultado de uma ampla pesquisa nos modelos então utilizados para representar qualidade de serviço no contexto de serviços web. A ideia central da proposta era incorporar no modelo elementos necessários para representar as necessidades não só de consumidores e provedores, mas também dos chamados montadores de serviços (*service composers*), os quais são responsáveis pela descoberta e seleção dinâmica de serviços utilizados em uma composição.

Segundo Jureta, Herssens e Faulkner (2009), modelos de qualidade expressivos devem ser capazes de permitir que consumidores especifiquem a qualidade esperada dos serviços, que provedores anunciem a qualidade de serviço com a qual eles se comprometem, e que montadores tenham informação suficiente para diferenciar serviços quando mais de um candidato está disponível para seleção.

QVDP não define atributos de qualidade específicos (e.g., disponibilidade, confiabilidade ou tempo de resposta), mas incorpora elementos que representam os conceitos relevantes para a descrição da qualidade, tais como métricas, valores e unidades. Nesse contexto, a ideia é que QVDP seja utilizado para que projetistas possam definir seus próprios modelos de qualidade, incluindo os atributos e métricas de qualidade relevantes para um determinado sistema. Dentre os diferenciais da proposta de QVDP, destacam-se: a possibilidade de representação do conceito de prioridade entre atributos de qualidade e a representação explícita das relações de dependência entre atributos distintos, ou seja, da correlação entre eles.

Em QVDP, um modelo de qualidade é composto por quatro submodelos: submodelo das características (Q), submodelo de valores (V), submodelo de dependências (D), e submodelo de prioridades (P). O submodelo Q define os conceitos de dimensão de qualidade, que representa uma propriedade não-funcional mensurável de um sistema e de característica de qualidade, que representa um agrupamento de dimensões. O submodelo V é utilizado na definição dos valores associados às dimensões de qualidade, os quais são especificados na definição de requisições ou ofertas de serviço. O submodelo D fornece uma representação explícita das dependências entre as dimensões, permitindo a indicação de correlações positivas e negativas entre as diferentes dimensões. Por fim, o submodelo D visa estabelecer as prioridades entre dimensões distintas, sendo fundamental para suportar a seleção de serviços quando mais de um serviço candidado está disponível. Embora os conceitos representados no modelo QVDP sejam bastante similares aos representados nos meta-modelos de qualidade da plataforma DSOA, existem importantes diferenças entre essas propostas.

Primeiramente, QVDP não deixa claro como os valores das medidas utilizadas na computação da métrica podem ser obtidas. Há somente comentários gerais falando sobre a possibilidade de existência de uma plataforma de monitoração subjacente que seria

responsável pela coleta dos dados, sem detalhes acerca da sua realização.

Em segundo lugar, embora o modelo permita uma descrição textual de como as métricas poderiam ser computadas, a proposta não deixa claro como os dados obtidos deste modelo poderiam ser automaticamente processados para a computação de métricas. Essa característica inviabiliza a utilização deste modelo para a definição e redefinição dinâmica de novas métricas de qualidade, que corresponde à questão fundamental tratada pela plataforma DSOA.

Em terceiro lugar, a proposta não visa a reutilização de modelos de qualidade e valores de forma independente de prioridade, ou seja, aplicações diferentes não são capazes de reutilizar definições de características e dimensões de qualidade, nem definir novas prioridades. Na plataforma DSOA, as informações acerca de prioridade são separadas do modelo de qualidade em si, permitindo que diferentes aplicações estabeleçam diferentes preferências.

Por fim, diferentemente da plataforma DSOA, QVDP não busca indicar como uma infraestrutura de SOA dinâmica (estendida) poderia ser construída em torno do modelo de qualidade proposto.

4.2.3 QoS-enabled WSDL (Q-WSDL)

Este trabalho apresenta uma extensão de WSDL, referenciada como QoS-enabled WSDL (Q-WSDL) (D'AMBROGIO, 2006; BOCCIARELLI; D'AMBROGIO, 2011), a qual visa permitir a especificação de requisitos não-funcionais associados aos serviços web. Para tanto, o trabalho apresenta um meta-modelo de serviços baseado nos conceitos que integram WSDL. Este modelo é estendido com elementos que modelam aspectos de qualidade, os quais foram concebidos com base em conceitos estabelecidos em dois perfis UML propostos pela OMG: (1) *UML Profile for Quality of Service and Fault Tolerance* e (2) *UML Profile Schedulability, Performance and Time*.

Assim como nos perfis citados, a qualidade de um serviço em Q-WSDL é descrita através de suas características de qualidade (*QoS Characteristics*), as quais são quantificadas através de um conjunto de dimensões (*QoS Dimensions*). Cada dimensão é descrita através de um valor, uma unidade de medida, uma fonte, um operador estatístico (e.g., média, variância, desvio padrão, e assim por diante), e uma relação de ordem. A despeito da relevância do trabalho no sentido de incorporar os elementos de qualidade identificados pela OMG no universo dos serviços Web, importantes aspectos limitam as possibilidades de uso de Q-WSDL no contexto das QSBAs.

Uma lacuna importante no meta-modelo proposto é a ausência de informação sobre o mecanismo utilizado na computação das métricas de qualidade associadas às características descritas. Embora o meta-modelo apresente o conceito de dimensões de qualidade que podem ser vistas como métricas de qualidade, não há nesse modelo referência a mecanismos de computação dessas métricas. Basicamente, o que o meta-modelo representa é o

conceito de fonte, o qual, segundo a proposta, especifica como o valor é obtido. Contudo, dado o universo de valores possíveis para este atributo (*required, assumed, predicted, measured, undefined*) observa-se que não é possível a especificação de como o valor da métrica pode ser calculado.

Outra limitação diz respeito à ausência de suporte à especificação de preferências e prioridades por parte do consumidor de serviço. Este tipo de informação é essencial quando diferentes serviços candidatos oferecem uma mesma funcionalidade, contudo com características de qualidade diferentes. Assim, a informação sobre as prioridades pode ser utilizada para ordenar os candidatos, permitindo a seleção daquele que é considerado o melhor serviço. Assim, a despeito da relevância do trabalho, o meta-modelo proposto não é suficiente para suportar o desenvolvimento e a execução das QSBAs.

4.3 CONSIDERAÇÕES FINAIS

O presente capítulo apresentou diversos trabalhos compreendendo as aplicações auto-adaptativas e as características de qualidade. O objetivo da análise desses trabalhos foi permitir a identificação de eventuais lacunas no estado da arte, as quais poderiam indicar caminhos a serem seguidos no contexto desta tese.

As propostas avaliadas representaram importantes avanços e influenciaram, de forma definitiva, a plataforma DSOA. Contudo, elas possuem importantes limitações do ponto de vista do suporte às aplicações auto-adaptativas baseadas em serviço e cientes de qualidade.

Em particular, a maioria das propostas não abordou a possibilidade de definição dinâmica de novas métricas de qualidade e nenhuma delas discutiu a necessidade de disponibilizar mecanismos que permitam que os desenvolvedores definam, em alto nível, como essas métricas podem ser efetivamente computadas.

Em outras palavras, as propostas não consideram como mecanismos de monitoração flexíveis podem ser concebidos e configurados a partir de informações representadas em alto nível pelas equipes de desenvolvimento. Como veremos, a forma de computar as métricas de qualidade tem um impacto direto nos valores mensurados, os quais funcionam como gatilhos para o processo de adaptação. Assim, a forma de computar as métricas influencia diretamente o processo de adaptação.

Buscando-se observar eventuais soluções em nível de modelo, avaliamos importantes meta-modelos utilizados no contexto de qualidade de serviços. A despeito dos pontos fortes de cada um, discutidos ao longo do capítulo, observamos que esses meta-modelos não possuem elementos capazes de permitir que as métricas representadas possam ser automaticamente interpretadas pelas plataformas subjacentes.

Neste contexto, o capítulo seguinte introduz a plataforma DSOA, a qual propõe que essa lacuna seja preenchida através da incorporação dos conceitos do domínio de eventos no espaço de modelagem das aplicações.

5 ARQUITETURA DA PLATAFORMA DSOA

> *A relação entre o pensamento e a palavra é um processo vivo; o pensamento nasce através das palavras. Uma palavra vazia de pensamento é uma coisa morta, e um pensamento despido de palavras permanece uma sombra.*
>
> Lev Semenovich Vygotsky,

Este capítulo introduz a plataforma DSOA e posiciona seus elementos em relação à plataforma genérica apresentada anteriormente. Como veremos, a compreensão da plataforma DSOA requer uma perspectiva multi-dimensional, envolvendo conceitos pertencentes aos domínios de serviço, qualidade, e evento. É a visão integrada desses conceitos em uma plataforma de suporte às aplicações auto-adaptativas que representa o principal diferencial do presente trabalho.

5.1 VISÃO GERAL

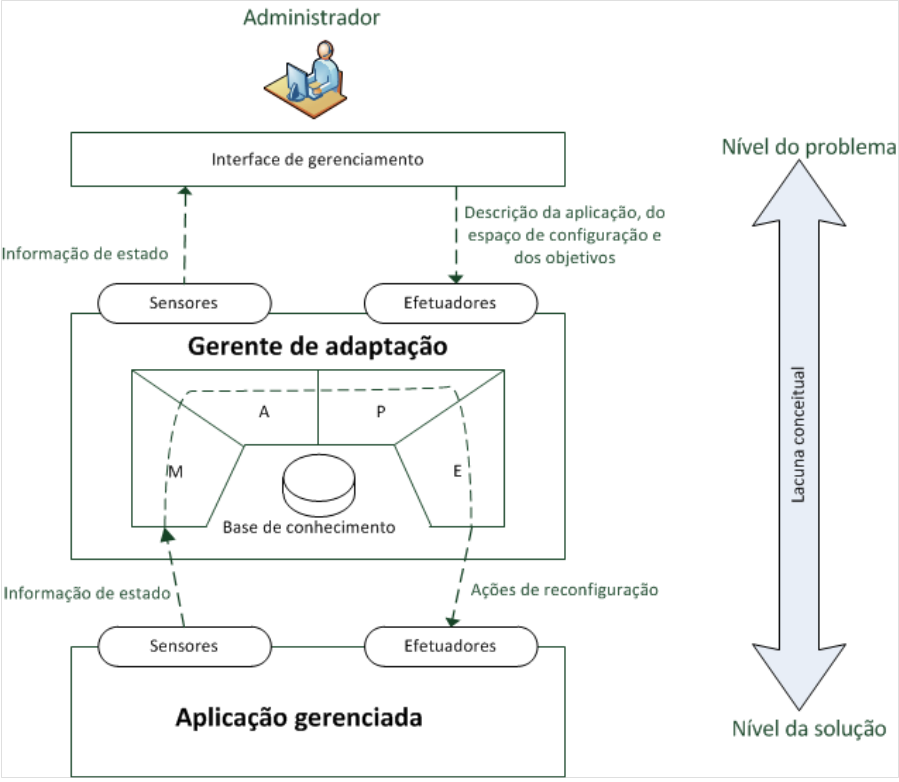
A plataforma DSOA foi proposta com o intuito de prover suporte para o desenvolvimento e execução de QSBAs. A essência dessas aplicações é a sua natureza auto-adaptativa relacionada ao nível de qualidade de serviço.

Como discutido anteriormente, a concepção de aplicações auto-adaptativas requer a elaboração de uma lógica de adaptação, sendo esta frequentemente introduzida através da implementação de um gerente de adaptação (SALEHIE; TAHVILDARI, 2009) responsável por realizar um laço de controle compreendendo as atividades de supervisão e adaptação (BENNACEUR et al., 2014). Uma visão mais detalhada dessas atividades pode ser realizada com base no laço de controle MAPE-K. Neste contexto, enquanto a atividade de supervisão compreende a monitoração e a análise, a atividade de adaptação compreende o planejamento e a execução.

Para conduzir o processo de adaptação, os gerentes manipulam meta-representações causalmente conectadas com a aplicação em execução, de forma que mudanças nessas representações sejam refletidas na aplicação e vice-versa. Tais meta-representações são normalmente materializadas na forma de modelos de execução, os quais são organizados em uma base de conhecimento alimentada a partir de informações fornecidas por desenvolvedores e administradores, e coletadas em tempo de execução, a partir de uma coleção de sensores.

No contexto das QSBAs, os modelos que compõem uma base de conhecimentos devem fornecer ao gerente de adaptação diversas informações acerca da aplicação em execução, tais como: (1) os serviços que a aplicação fornece e/ou requer, (2) os atributos de qualidade que são relevantes do ponto de vista da aplicação, (3) as restrições impostas aos

Figura 17 – Visão conceitual de uma aplicação auto-adaptativa



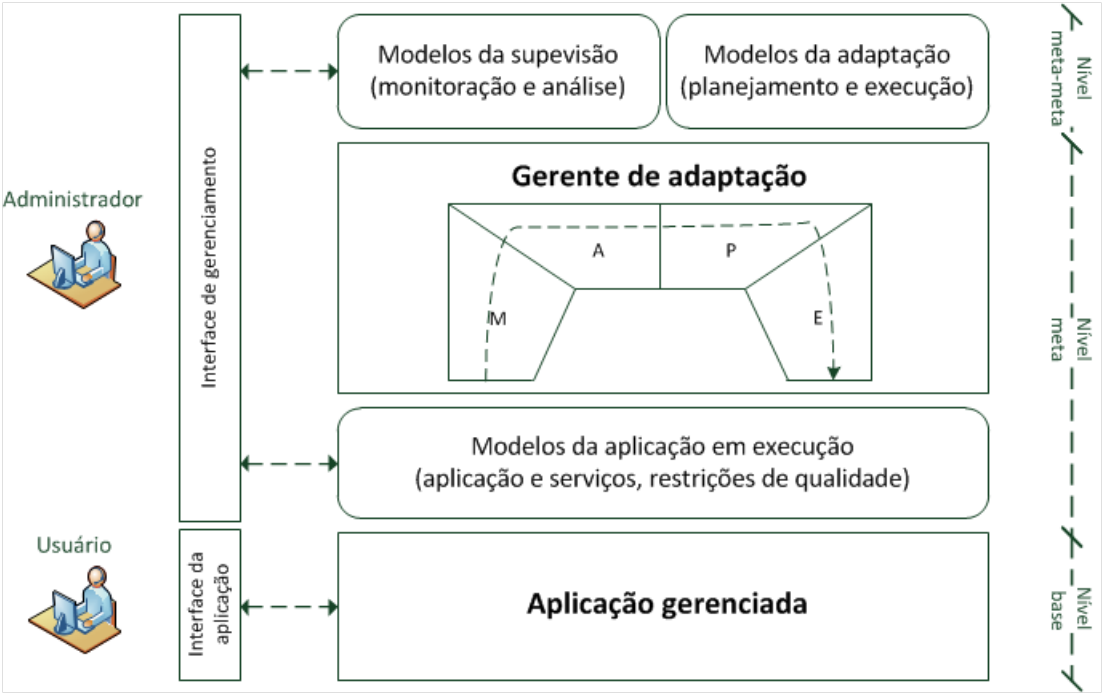
serviços em termos do nível de qualidade esperado, e (4) os serviços similares que estão correntemente disponíveis. Essas informações são essenciais e formam a base de diversas pesquisas relacionadas à adaptação dinâmica de aplicações (ROMERO et al., 2010; HALLSTEINSEN et al., 2012; HUBER et al., 2017). A **Figura 17** representa os elementos conceituais descritos.

A despeito da relevância das informações mencionadas, elas não são suficientes para possibilitar a configuração (nem a reconfiguração dinâmica) do próprio gerente de adaptação, uma vez que não descrevem como esse gerente deve realizar as atividades de supervisão e de adaptação. De fato, para que o próprio gerente de adaptação possa ser adaptado, ele deve expor uma (meta-)representação de seus processos internos, de forma que alterações nessa representação modifiquem a forma como esse gerente realiza as atividades de monitoração, análise, planejamento, e execução. O conjunto composto pela aplicação gerenciada, pelo gerente de adaptação, e pelas meta-representações dessa aplicação e do próprio gerente compõe uma arquitetura reflexiva com três níveis, representada na **Figura 18**.

5.1.1 Arquitetura Reflexiva

Nas aplicações auto-adaptativas baseadas em modelos em tempo de execução, os modelos que descrevem as aplicações ocupam, juntamente com o gerente de adaptação, o plano meta de uma arquitetura reflexiva, enquanto que a própria aplicação, incluindo os modelos

Figura 18 – Arquitetura reflexiva



Fonte: o autor

que esta utiliza para representar o domínio de negócio, compõe o plano base [(ANDERSSON et al., 2009)][(WEYNS; MALEK; ANDERSSON, 2012)].

Como vimos, para que o próprio gerente de adaptação possa ser adaptado, ele deve expor suas funcionalidades através de modelos causalmente conectados. Em particular, para que possamos modificar a forma como um gerente de adaptação realiza a supervisão, é necessário expor seus mecanismos de monitoração e análise através de uma representação causalmente conectada, a qual deve compor o nível meta-meta da arquitetura reflexiva.

De maneira similar, para modificarmos a forma como esse gerente realiza a atividade de adaptação, devemos representar os mecanismos responsáveis pelo planejamento e execução através de modelos, e integrar essas representações no nível meta-meta da arquitetura reflexiva. A Figura 18 representa como os elementos descritos podem ser organizados em torno de uma arquitetura reflexiva de três níveis.

5.1.2 Motivação

Tendo em vista o papel central que os modelos desempenham no contexto das arquiteturas reflexivas e, consequentemente, na condução do processo de adaptação, é fundamental que eles sejam capazes de representar, de forma adequada, tanto as aplicações adaptáveis, quanto o gerente de adaptação. Essa visão é natural, uma vez que uma aplicação auto-adaptativa é, na verdade, composta por esses dois elementos. Em última análise, é o conjunto de modelos reflexivos, juntamente com as funcionalidades neles representadas, que define o nível de flexibilidade das diferentes plataformas.

De forma similar à solução adotada na plataforma DSOA, diversos projetos de pesquisa (e.g., (HUBER et al., 2014a; HALLSTEINSEN et al., 2012)) utilizam modelos arquiteturais para representar aplicações auto-adaptativas como composições de serviços, aos quais estão associadas características de qualidade. Uma vez que os modelos arquiteturais são capazes de representar as aplicações como composições de unidades elementares, e de estabelecer restrições associadas a essas unidades, esses modelos são suficientes nos contextos relacionados à descoberta, seleção, e substituição dinâmica de serviços.

Contudo, poucas soluções atuais oferecem representações dos processos internos que compõem os gerentes de adaptação. Em geral, quando disponíveis, essas representações são centradas nas políticas de adaptação, as quais são normalmente baseadas em regras do tipo Evento-Condição-Ação (ECA) (VOGEL; GIESE, 2010; HUBER et al., 2014a; HUBER et al., 2017) ou na declaração de funções de utilidade e objetivo de alto nível (FLOCH et al., 2006; HALLSTEINSEN et al., 2012).

A plataforma DSOA oferece uma visão complementar a esses trabalhos. De fato, além de propor modelos reflexivos capazes de representar uma aplicação em execução como uma composição de serviços ciente de qualidade, a plataforma DSOA introduziu, em seus modelos, um conjunto de abstrações fundamentais para representar um gerente de adaptação do ponto de vista da atividade de supervisão.

Em essência, essas abstrações são utilizadas para representar, em modelos de alto nível, os atributos e métricas de qualidade relevantes, e os processos utilizados na monitoração e análise desses elementos. Através desses modelos, os desenvolvedores/administradores são capazes de definir novas características de qualidade e de determinar como essas características devem ser dinamicamente monitoradas. Mais ainda, esses modelos são dinâmicos, de forma que novas características de qualidade podem ser dinamicamente incluídas e os processos de monitoração e análise podem ser reconfigurados em tempo de execução.

5.2 PLATAFORMA DSOA

Esta seção apresenta uma visão geral dos elementos que constituem a plataforma DSOA, destacando os modelos de serviço, qualidade, e evento, em virtude da importância que eles possuem no processo de adaptação dinâmica. Uma descrição detalhada dos elementos que constituem a plataforma será apresentada nos capítulos seguintes.

Na concepção da plataforma DSOA, o suporte às QSBAs deve ser integrado, compreendendo, além dos aspectos relacionados à execução, o apoio ao desenvolvimento dessas aplicações. Nesse contexto, apresentaremos uma visão da plataforma com base nos elementos conceituais que compõem uma plataforma genérica de suporte à auto-adaptação (vide Capítulo 3).

5.2.1 Ambiente de Execução

O primeiro passo na concepção da plataforma DSOA consistiu em identificar o conjunto de modelos que devem compor a base de conhecimento utilizada pelo gerente na condução do processo de adaptação. Na plataforma DSOA, a necessidade de adaptação está vinculada ao suporte às QSBAs. Uma vez que essas aplicações são, por definição, composições de serviços cientes de qualidade, é natural que o conjunto de modelos utilizados em tempo de execução compreenda conceitos pertencentes aos domínios de qualidade e serviço.

5.2.1.1 Domínio de Qualidade

No contexto das QSBAs, a qualidade de serviço é um importante fator para diferenciar serviços funcionalmente equivalentes, sendo o principal gatilho para o processo de adaptação dessas aplicações. Neste contexto, uma definição clara dos conceitos pertencentes ao domínio de qualidade é fundamental. Em geral, essa definição se dá através da adoção de um modelo de qualidade pré-definido, o qual descreve e organiza atributos e métricas, compondo uma taxonomia de qualidade.

Como vimos na Seção 2.4.2, essa abordagem possui importantes limitações. A primeira diz respeito à necessidade de identificação prévia do universo de atributos e métricas de qualidade que devem compor o modelo. Uma vez que a área de qualidade é bastante ampla, não é possível a concepção de um modelo único capaz de estabelecer uma lista exaustiva desses elementos. Ademais, a adoção de um modelo pré-definido limita as aplicações, uma vez que estas não são capazes de definir atributos e métricas próprios de seu domínio de negócio.

Uma abordagem mais flexível consiste em projetar ambientes de execução com base em uma visão mais abstrata dos conceitos de qualidade, os quais podem ser representados através de um meta-modelo. Desta forma, um desenvolvedor de aplicação pode utilizar esses conceitos para definir modelos de qualidade customizados, incluindo os atributos e métricas relevantes para o seu contexto de negócio. Visando suportar a adaptação dinâmica, os modelos de qualidade devem ser mantidos em tempo de execução, sendo um componente importante da base de conhecimento. Por fim, as métricas de qualidade definidas nesses modelos podem ser utilizadas para estabelecer o nível de qualidade dos serviços providos e requeridos pelas aplicações.

Fundamentada nessas ideias, a plataforma DSOA representa os conceitos inerentes ao domínio de qualidade em nível de meta-modelo, de forma que esses conceitos possam ser instanciados e utilizados na concepção de modelos de qualidade customizados e extensíveis.

Visando separar a definição dos conceitos de qualidade dos mecanismos utilizados na sua monitoração, os modelos de qualidade adotados pela plataforma não incorporam informação sobre como as métricas definidas nesses modelos devem ser monitoradas em

tempo de execução. Como veremos, essa informação é definida através dos modelos de monitoração que ocupam o nível meta-meta da arquitetura reflexiva e são utilizados para configurar a atividade de supervisão desempenhada pelo gerente de adaptação.

5.2.1.2 Domínio de Serviço

Os modelos propostos pela plataforma DSOA descrevem uma aplicação a partir dos componentes utilizados para implementá-la e das interfaces dos serviços que esses componentes requerem e fornecem, preenchendo uma lacuna deixada pelo padrão arquitetural SOA. De fato, embora SOA desempenhe um importante papel ao oferecer um mecanismo para a exposição de funcionalidades de alto nível através de interfaces bem definidas e do papel de registro de serviços, esse padrão arquitetural não especifica como as aplicações podem ser desenvolvidas nem gerenciadas.

É importante ressaltar que parte dessa lacuna havia sido preenchida com a proposição de SCA (BEISIEGEL et al., 2007) ao definir um modelo de componentes a ser utilizado na estruturação das aplicações baseadas em serviços. À semelhança do que ocorre em outras plataformas baseadas em componentes, as aplicações desenvolvidas em SCA são implementadas em linguagens de programação tradicionais e descritas através da utilização de uma linguagem de montagem. Essa linguagem é, em essência, uma ADL, sendo projetada para permitir a representação dos componentes que formam uma aplicação e das conexões entre eles. Assim, SCA trouxe para o mundo de serviços a noção de arquitetura de software, fomentando a ideia central de CBD de permitir a construção de aplicações a partir de componentes reutilizáveis.

Embora a linguagem de montagem introduzida com a proposição de SCA tenha definido a forma como uma aplicação baseada em serviços pode ser inicialmente configurada, esse padrão não proporcionou solução para a necessidade de reconfiguração dinâmica. De fato, os descritores de aplicação utilizados em SCA podem ser entendidos como modelos de desenvolvimento, os quais não são mantidos em tempo de execução, inviabilizando a utilização dos meta-dados incorporados nesses modelos na condução do processo de adaptação.

Uma evolução importante veio com a proposição de Frascati (SEINTURIER et al., 2009) que introduziu em SCA capacidades reflexivas. No contexto de Frascati, os descritores das aplicações são utilizados para dar origem a um conjunto de objetos que compõem o nível meta de uma arquitetura reflexiva, permitindo o monitoramento e a reconfiguração dinâmica das aplicações. Embora as capacidades reflexivas introduzidas por Frascati sejam essenciais para suportar as QSBAs, elas não são suficientes.

De fato, a natureza auto-adaptativa dessas aplicações requer que elas próprias sejam capazes de conduzir o processo de adaptação, utilizando como guia o nível de qualidade dos serviços providos e requeridos. Assim, para que uma aplicação seja ciente de qualidade, é necessário associar às especificações dos serviços, restrições relativas aos níveis de

qualidade. Essas restrições, estabelecidas com base nas métricas definidas no modelo de qualidade utilizado pela aplicação, são adotadas pelos gerentes de adaptação como critério para a descoberta e seleção de serviços. Como vimos na Seção 2.4.2, os documentos que descrevem os serviços providos e requeridos por uma aplicação, com as respectivas restrições de qualidade, são denominados QSDs.

Com base nos modelos descritos até o presente momento, a construção de uma QSBA envolve: (1) a implementação da lógica de negócio da aplicação, através da utilização de uma linguagem de programação de uso geral; (2) a concepção de um modelo de qualidade, especificando os atributos e métricas relevantes do ponto de vista da aplicação; e (3) a definição de modelos especificando os componentes que formam a aplicação e os serviços por eles providos e requeridos (anotados com restrições de qualidade).

É importante mencionar que modelos de qualidade e serviços semelhantes aos propostos acima são utilizados em diferentes projetos de pesquisa, em especial naqueles com foco na seleção de serviços baseada em atributos de qualidade (*e.g.* (KRITIKOS et al., 2013; HUBER et al., 2014a)).

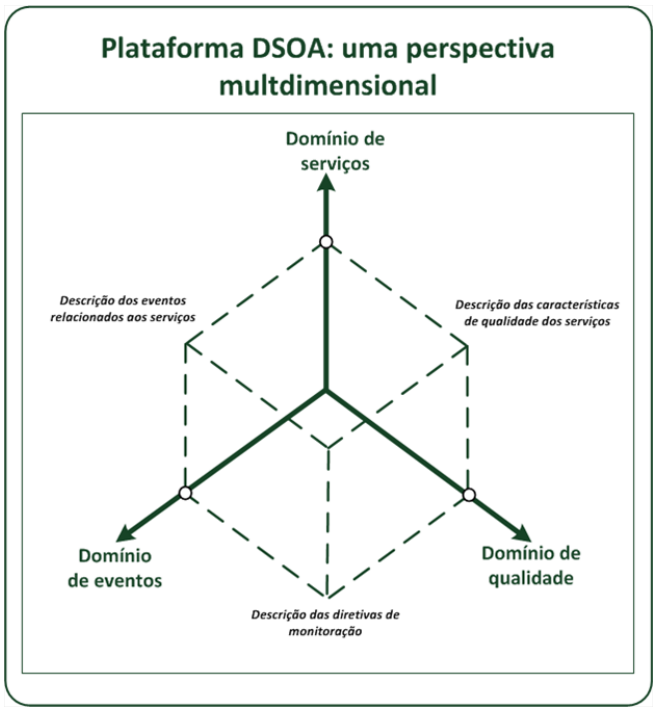
Apesar de sua ampla utilização, modelos representando unicamente serviço e qualidade não são capazes de permitir uma representação dos processos internos realizados por um gerente de adaptação, uma vez que não incorporam nenhuma informação acerca de como os níveis de qualidade podem ser aferidos em tempo de execução. Em suma, a partir desses modelos não é possível determinar como as métricas de qualidade devem ser computadas. Desta forma, embora os referidos modelos possam ser utilizados para caracterizar o nível de qualidade "esperado" dos serviços, eles não são úteis do ponto de vista de monitoração e análise.

Uma vez que os desenvolvedores não podem utilizar esses modelos para configurar os mecanismos de monitoração e análise do ambiente de execução, eles possuem duas alternativas. A primeira consiste em utilizar os mecanismos de monitoração com sua "configuração padrão" e definir o processo de adaptação das aplicações utilizando as métricas de qualidade já embutidas nas plataformas subjacentes, implicando na aceitação da semântica que essa plataforma atribui a essas métricas. Neste contexto, a atividade de supervisão desempenhada pelo gerente de adaptação funciona como uma "caixa preta", não podendo o desenvolvedor interferir nesse processo, que é de inteira responsabilidade da plataforma.

A segunda opção consiste em embutir a lógica de supervisão e adaptação na própria aplicação. Essa abordagem possui dois inconvenientes: não está alinhada com o princípio de separação de interesses, uma vez que promove uma mistura de código de negócio com código de infraestrutura; e promove um alto acoplamento entre a aplicação e a sua plataforma de execução, uma vez que os mecanismos utilizados pelas aplicações para coletar dados relacionados às métricas de qualidade são específicos de plataforma.

Em suma, os conceitos pertencentes aos domínios de qualidade e de serviços não são su-

Figura 19 – Visão multi-dimensional



Fonte: o autor

ficientes para permitir a reconfiguração dinâmica dos ambientes de execução responsáveis por gerenciar as QSBAs. De fato, em uma plataforma flexível, os desenvolvedores devem ser capazes de especificar, em alto nível, não somente as informações necessárias para a configuração e adaptação de suas aplicações, mas também aquelas necessárias à configuração dinâmica dos mecanismos de monitoração e adaptação embutidos nos gerentes de adaptação que compõem os próprios ambientes de execução.

Em um cenário ideal, espera-se que os desenvolvedores possam não somente definir novas métricas de qualidade, mas também determinar como essas devem ser monitoradas e mensuradas. Mais ainda, uma solução flexível deve permitir que o próprio processo de monitoração seja adaptável, devendo ser possível uma modificação dinâmica dos algoritmos de computação dessas métricas.

Visando oferecer toda a flexibilidade mencionada, a plataforma DSOA propõe a inclusão de uma nova dimensão no espaço de modelagem das aplicações: o domínio de eventos (vide **Figura 19**). Como veremos na subseção seguinte, a introdução de modelos representando os conceitos relacionados a esse novo domínio e a interconexão desses conceitos com aqueles advindos dos domínios de qualidade e serviços representam o principal diferencial da plataforma DSOA no que diz respeito à capacidade de suportar as QSBAs.

5.2.1.3 Domínio de Eventos

Como vimos na seção anterior, utilizando apenas os conceitos pertencentes aos domínios de qualidade e serviços, os desenvolvedores não são capazes de fornecer a um ambiente de execução, informações suficientes para que este seja capaz de monitorar as aplicações e computar as métricas de qualidade necessárias. Neste contexto, a proposição de uma solução mais flexível envolve uma compreensão ampla dos mecanismos normalmente utilizados na supervisão das aplicações. Em outras palavras, é necessário entender como os gerentes de adaptação realizam a monitoração e análise das aplicações em execução.

Em geral, essas atividades envolvem a utilização de um conjunto de sensores, os quais são responsáveis por fornecer informações detalhadas acerca das aplicações em execução e do próprio ambiente. Uma abordagem comum na construção de mecanismos de monitoração consiste na utilização de sensores capazes de notificar ocorrências relevantes tão logo essas aconteçam através da utilização de eventos.

Em tempo de execução, os eventos gerados pelos sensores, referenciados como *eventos primitivos*, podem ser analisados através da introdução de agentes de processamento, os quais são capazes de realizar operações de correlação, filtragem, transformação, e/ou agregação, dando origem a eventos de mais alto nível, comumente referenciados como *eventos complexos* ou *derivados* (LUCKHAM, 2001; ETZION; NIBLETT, 2010).

Com base no entendimento desses conceitos, a plataforma DSOA propõe que os tipos de eventos que ocorrem em tempo de execução se tornem visíveis para os desenvolvedores de aplicação. A ideia é que as métricas de qualidade sejam computadas a partir dos eventos gerados pelos sensores que compõem o mecanismo de monitoração. Para tanto, propõe-se a introdução de conceitos do domínio de eventos no espaço de modelagem.

Nesse contexto, é importante observar que o nível de abstração dos dados contidos nos eventos primitivos sinalizados pelos sensores pode não corresponder diretamente ao nível de abstração necessário para computar os valores das métricas de qualidade. Portanto, é necessário estabelecer uma ponte entre esses níveis de abstração através da introdução do conceito de transformação no nível de modelo. Tais transformações realizam a tradução dos eventos primitivos em eventos complexos de mais alto nível, os quais são utilizados para comunicar os valores das métricas de qualidade.

Na plataforma DSOA, as transformações são especificadas através de modelos (concebidos em tempo de desenvolvimento), os quais são utilizados em tempo de execução para gerar e configurar os agentes de processamento de eventos. Ao estabelecer as transformações necessárias para converter eventos primitivos em eventos complexos contendo os valores correspondentes às métricas de qualidade, os agentes de processamento definem a semântica dessas métricas, as quais são organizadas nos modelos de qualidade descritos previamente. Um importante diferencial da plataforma DSOA consiste na manutenção, em tempo de execução, dos modelos utilizados para representar as transformações de eventos, de forma que alterações nesses modelos promovem reconfigurações dinâmicas

dos agentes correspondentes.

Do ponto de vista da arquitetura reflexiva mencionada no início deste capítulo, os modelos representando os eventos e as transformações, assim como os modelos de monitoração, representando os mapeamentos entre os eventos e as métricas de qualidade, compõem a meta-meta-representação do mecanismo de supervisão (monitoração e análise) da plataforma DSOA, a qual pode ser dinamicamente modificada de forma a promover uma reconfiguração dinâmica do próprio gerente de adaptação.

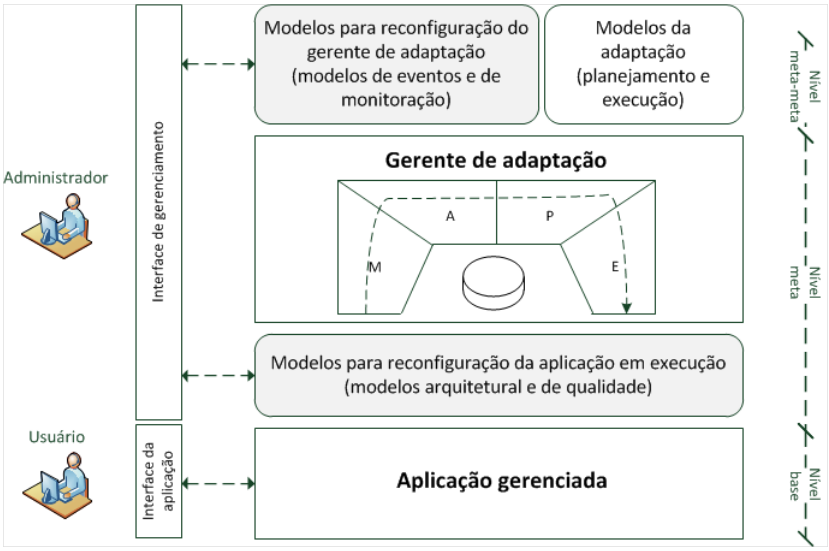
Em suma, a introdução dos conceitos do domínio de eventos e a conexão entre esses conceitos e aqueles oriundos dos domínios de qualidade e serviços habilitam a proposição de um mecanismo de monitoração bastante flexível, sendo esse central para as plataformas de suporte às QSBAs.

A despeito da relevância dos conceitos pertencentes ao domínio de eventos, as plataformas de suporte às aplicações auto-adaptativas atuais não oferecem recursos substanciais para a representação desses conceitos em nível de modelo. Em particular, essas plataformas não permitem a representação de transformações de eventos através de modelos de desenvolvimento mantidos em tempo de execução, limitando a utilização dos modelos para fins de reconfiguração dinâmica dos mecanismos de monitoração. Mais ainda, não há uma associação explícita entre eventos e qualidade, ficando os desenvolvedores limitados à detectar e processar explicitamente eventos primitivos previamente definidos de forma a extrair a informação de qualidade. A necessidade de manipular diretamente tais eventos no código da aplicação tende a promover uma mistura entre código de negócio e de adaptação, podendo comprometer a manutenibilidade do sistema.

Na íntegra, a solução proposta, implementada na plataforma DSOA, permite que os desenvolvedores especifiquem: (1) modelos de qualidade, estabelecendo uma taxonomia que define os atributos e métricas relevantes do ponto de vista de uma aplicação, (2) modelos arquiteturais, representando os componentes que constituem uma aplicação, os serviços que eles fornecem e requerem, e as restrições sobre o nível de qualidade desses serviços, (3) modelos de eventos, definindo os eventos que ocorrem na plataforma e as transformações possíveis entre eles, e (4) modelos de monitoração, responsáveis por associar os eventos compostos às métricas definidas no modelo de qualidade.

Do ponto de vista da arquitetura reflexiva, os modelos arquiteturais e de qualidade compõem, juntamente com o gerente de adaptação, o nível meta, viabilizando a reconfiguração dinâmica das aplicações. Por outro lado, o nível meta-meta dessa arquitetura é ocupado pelos modelos de eventos e de monitoração, os quais reificam o mecanismo de supervisão embutido no gerente de adaptação, e viabilizam a sua adaptação dinâmica. O posicionamento dos modelos propostos na arquitetura reflexiva é apresentado na **Figura 20**.

Figura 20 – Arquitetura reflexiva da plataforma DSOA



Fonte: o autor

5.2.2 Ambiente de Desenvolvimento

Como vimos, o processo de adaptação das QSBAs é conduzido pelo gerente de adaptação através da utilização de modelos mantidos em tempo de execução. Nesse contexto, o papel das equipes de desenvolvimento se concentra na produção de modelos de desenvolvimento utilizados para povoar os modelos de execução. Para que os modelos de desenvolvimento possam ser produzidos, é necessária a definição de um conjunto de linguagens de modelagem, o qual foi estabelecido em função dos domínios representados e dos perfis de desenvolvedores responsáveis por projetar os modelos descritos anteriormente.

Do ponto de vista do domínio de qualidade, a plataforma DSOA propõe o uso de uma linguagem textual simples, referenciada como *DSOA Quality Language* (DSOA-QL). Como veremos, os elementos que compõem a sintaxe abstrata de DSOA-QL são genéricos, permitindo a definição de modelos customizados de qualidade. Essa linguagem é utilizada pelos especialistas em qualidade para criar uma taxonomia que organiza os atributos e métricas de qualidade utilizados por uma QSBA para estabelecer os níveis de qualidade de serviço. DSOA-QL é uma linguagem independente (VOELTER et al., 2013), ou seja, modelos definidos utilizando essa linguagem não fazem referência aos conceitos pertencentes as outras linguagens da plataforma.

As aplicações definidas na plataforma DSOA são especificadas através de modelos arquiteturais, projetados utilizando-se uma ADL própria, referenciada como *DSOA Architecture Language* (DSOA-AL). De forma semelhante a outras ADLs, a linguagem DSOA-AL é uma linguagem incompleta, cuja finalidade é permitir a descrição da aplicação de um ponto de vista estrutural.

Desta forma, os arquitetos utilizam DSOA-AL para definir os componentes que integram uma aplicação e as interfaces dos serviços que esses fornecem e requerem. DSOA-AL

depende da linguagem DSOA-QL para estabelecer os níveis de qualidade desejados. Assim, os serviços especificados nos modelos descritos em DSOA-AL podem ser anotados com restrições, as quais referenciam métricas definidas no modelo de qualidade. Em outras palavras, a linguagem DSOA-AL é utilizada para representar as características de qualidade de uma QSBA.

A plataforma DSOA inclui, também, uma linguagem declarativa utilizada para representar os conceitos do domínio de eventos, chamada *DSOA Event Language* (DSOA-EL). DSOA-EL é uma linguagem independente cujas principais construções são utilizadas para especificar novos tipos de eventos e transformações.

Por fim, a plataforma DSOA possui uma linguagem, chamada *DSOA Monitoring Language* (DSOA-ML), utilizada para a concepção de modelos de monitoração, os quais mapeiam as métricas de qualidade nos eventos complexos gerados pelos agentes de processamento. Dessa forma, DSOA-ML depende diretamente de DSOA-QL e DSOA-EL. Em tempo de execução, esses modelos de monitoração são utilizados na configuração dos gerentes de adaptação.

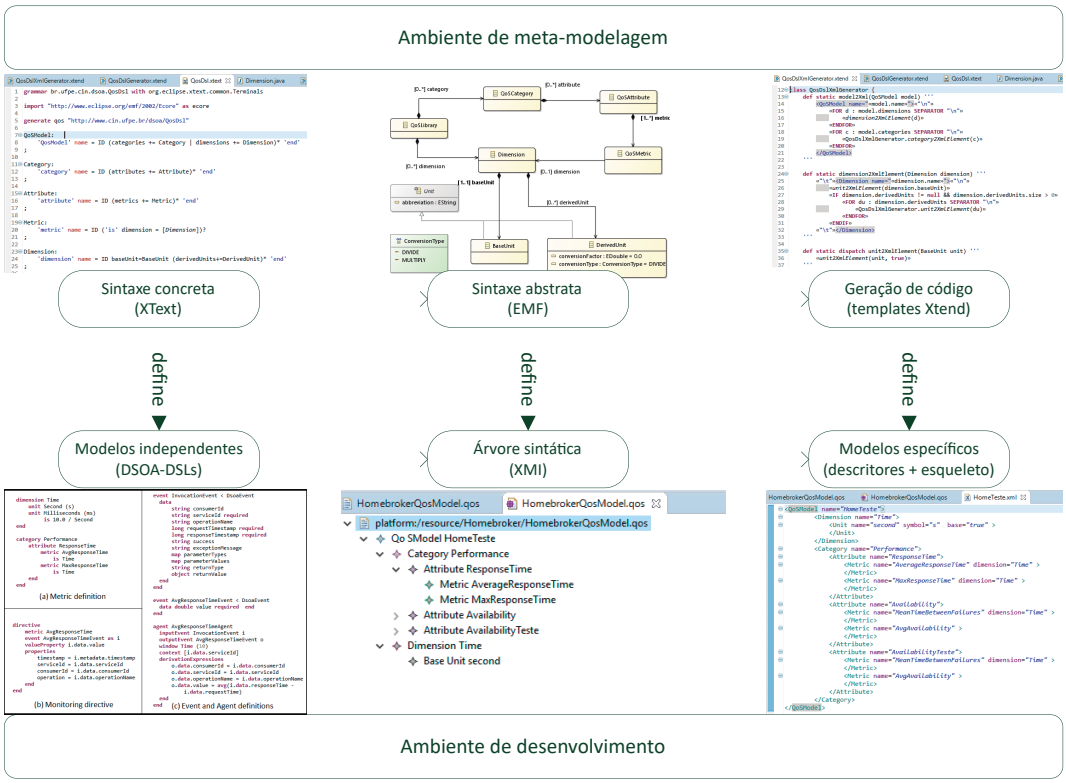
É importante mencionar que o ambiente de desenvolvimento da plataforma possui geradores que utilizam os modelos arquiteturais para gerar esqueletos de código, os quais são complementados manualmente com a descrição do comportamento funcional utilizando-se uma linguagem de uso geral (em particular, Java). Além do gerador de código, o ambiente de desenvolvimento possui um transformador de modelos que utiliza os modelos em DSOA-AL para gerar descritores específicos de plataforma, contendo os meta-dados utilizados na instanciação e configuração dos componentes em tempo de execução.

5.2.3 Ambiente de Meta-Modelagem

Na plataforma DSOA, a definição das linguagens de modelagem é realizada através da utilização de Xtext (BETTINI, 2016). Neste ambiente, uma DSL é definida a partir da especificação de sua sintaxe concreta, materializada sob a forma de regras definidas em uma notação própria similar à *Backus-Naur Form* (BNF). A partir dessas regras, a ferramenta define a sintaxe abstrata e a semântica estática da linguagem através da geração de um meta-modelo em EMF (STEINBERG et al., 2009).

Os modelos de transformação responsáveis pela geração de artefatos específicos de plataforma (esqueletos de código e descritores) são especificados a partir de *templates* em Xtend (BETTINI, 2016), uma linguagem baseada na máquina virtual Java e própria da plataforma Xtext. Por fim, com base nas gramáticas, meta-modelos e *templates*, o Xtext gerou as ferramentas que compõem o ambiente de desenvolvimento. A **Figura 21** ilustra os elementos descritos, a relação entre eles, e os artefatos gerados nos ambientes de desenvolvimento.

Figura 21 – Ambientes de meta-modelagem e de desenvolvimento



Fonte: baseada em Mooij e Hooman (2018)

5.3 CONSIDERAÇÕES FINAIS

A plataforma DSOA foi desenvolvida com o intuito de apoiar o ciclo de vida completo das aplicações auto-adaptativas baseadas em serviço e cientes de qualidade, referenciadas como QSBAs. Nesse sentido, a plataforma oferece desde um conjunto de DSLs utilizadas para a concepção dos modelos de desenvolvimento, até um ambiente de execução baseado nesses modelos.

Considerando-se que as QSBAs são aplicações baseadas em serviço e cientes de qualidade, é natural que essas aplicações sejam representadas, em tempo de execução, por modelos refletindo os conceitos pertencentes a esses dois domínios. Embora esses modelos sejam essenciais para a adaptação dinâmica das aplicações e sejam adotados por diversas plataformas, eles não oferecem informação suficiente para permitir a configuração dinâmica dos mecanismos internos de um gerente de adaptação, limitando o nível de flexibilidade das soluções atualmente disponíveis.

A plataforma DSOA foi proposta com o intuito de preencher essa lacuna conceitual, incorporando modelos representando conceitos do domínio de eventos. Através desses modelos, a plataforma permite a definição declarativa de eventos e de transformações, viabilizando a geração, em tempo de execução, dos agentes responsáveis pelo processamento dos eventos primitivos (produzidos pelos sensores) e pela transformação desses eventos em

eventos complexos de mais alto nível. Uma vez que os agentes de processamento são representados por modelos mantidos em tempo de execução, eles podem ser dinamicamente modificados através da manipulação desses modelos, promovendo uma reconfiguração dinâmica dos mecanismos de monitoração e análise embutidos na plataforma.

Por fim, os eventos complexos produzidos pelos agentes de processamento são mapeados para métricas de qualidade através da utilização de modelos de monitoração. Diz-se nesse contexto, que os agentes de processamento atribuem semântica às métricas de qualidade, uma vez que são as regras incorporadas nesses agentes que definem como essas métricas devem ser computadas.

Em suma, a contribuição central da tese consiste na proposição de uma plataforma de suporte às QSBA com uma arquitetura reflexiva em três camadas, compreendendo modelos mantidos em tempo de execução que representam os domínios de serviço, qualidade, e evento. A representação integrada desses domínios viabiliza a construção de uma plataforma orientada a serviços bastante flexível, capaz de suportar não somente a adaptação das aplicações em execução, mas também a reconfiguração dinâmica dos próprios gerentes de adaptação.

6 MODELOS E META-MODELOS

> *I keep six honest serving men. They taught me all I knew. Their names are What and Why and When and How and Where and Who*
>
> Rudyard Kipling,

No capítulo anterior, motivamos a proposição da plataforma DSOA e introduzimos sua arquitetura reflexiva baseada em três camadas. Nessa arquitetura, os modelos mantidos em execução são peças-chave. Considerando-se essa relevância, é essencial uma definição clara acerca do papel desses modelos e das informações que eles devem conter. De uma forma geral, tais informações devem ser determinadas em função das necessidades de configuração e reconfiguração dinâmica. Nesse contexto, o presente capítulo apresenta uma visão detalhada dos modelos utilizados em tempo de execução pela plataforma. Iniciamos a apresentação discorrendo acerca de como esses modelos de execução são gerados e qual o papel desempenhado por eles no contexto da plataforma DSOA. Na sequência, apresentamos um meta-modelo, descrevendo os elementos que compõem esses modelos.

6.1 MODELOS NA PLATAFORMA DSOA

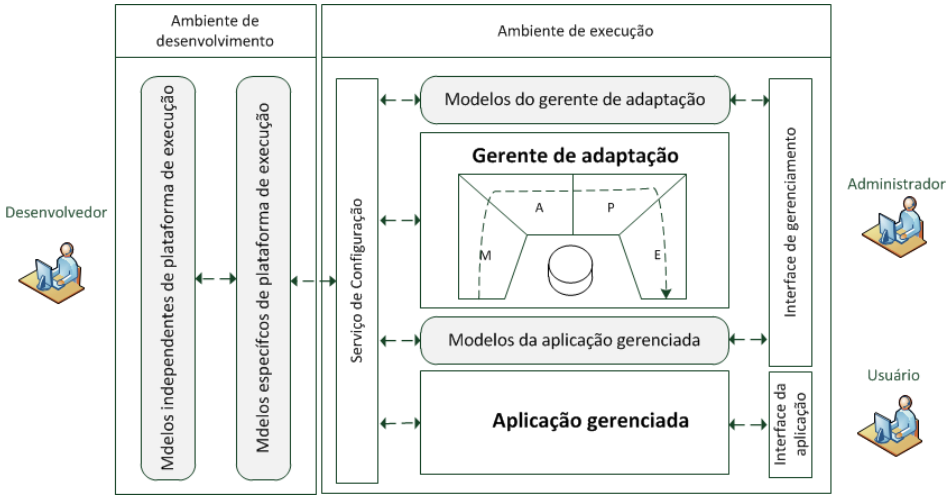
A **Figura 22** posiciona conceitualmente os modelos de desenvolvimento e execução utilizados na plataforma DSOA. Como se pode observar, o desenvolvimento de uma aplicação se inicia com a produção de um conjunto de modelos independentes de plataforma, os quais são concebidos utilizando-se linguagens específicas de domínio suportadas por um ambiente de desenvolvimento próprio. Em seguida, esses modelos são transformados em modelos específicos de plataforma (arquivos de configuração) e em um esqueleto de código fonte, o qual deve ser manualmente completado.

Concluído o desenvolvimento, a aplicação pode ser instalada em um ambiente de execução. Durante o processo de instalação, os modelos específicos de plataforma são interpretados por um serviço de configuração, que utiliza os meta-dados contidos nesses modelos para: (1) instalar, instanciar, e configurar os componentes que formam a aplicação, (2) configurar o gerente de adaptação, e (3) povoar os modelos de execução.

Visando representar uma QSBA, os modelos de execução (e, consequentemente, os de desenvolvimento utilizados para povoá-los) devem conter meta-dados descrevendo: (1) os componentes da aplicação e os serviços que esses fornecem e requerem, (2) os atributos e métricas de qualidade conhecidos, e (3) as restrições que devem ser impostas sobre o nível de qualidade dos serviços.

A despeito das informações mencionadas serem suficientes para descrever uma aplicação, não há entre elas nenhuma informação que possa orientar o gerente de adaptação

Figura 22 – Modelos na plataforma DSOA



Fonte: o autor

acerca de como ele deve conduzir suas atividades. Em particular, embora haja uma descrição acerca das restrições que devem ser observadas com relação ao nível de qualidade dos serviços, essas descrições estão em "alto nível", sendo necessária alguma informação acerca de como as ocorrências de tempo de execução podem ser mapeadas para essas restrições, e o que o gerente de adaptação deve fazer quando essas restrições não forem observadas.

Para fornecer ao gerente as informações necessárias ao processo de adaptação, além de representar os elementos que compõem as aplicações, os modelos de desenvolvimento independentes de plataforma devem possuir informações relacionadas à configuração dos mecanismos de supervisão e adaptação.

Visando suprir essa necessidade, os modelos de desenvolvimento utilizados na plataforma DSOA incluem elementos do domínio de eventos, definindo como ocorrências em tempo de execução podem ser processadas para derivar os valores das métricas de qualidade associadas aos serviços. Mais ainda, a plataforma permite que os desenvolvedores definam como os serviços requeridos pelas aplicações podem ser selecionados e se a aplicação deve ou não ser adaptada quando da violação dos limites estabelecidos para o nível de qualidade.

É importante observar que as informações descritas até esse ponto não representam aspectos dinâmicos, os quais são fundamentais nos modelos de tempo de execução (LEHMANN et al., 2011; BRETON; BÉZIVIN, 2001). Esses aspectos são abordados nas seções seguintes que detalham os meta-modelos dos modelos de tempo de execução da plataforma DSOA.

6.2 META-MODELOS DE TEMPO DE EXECUÇÃO

Segundo Bennaceur et al. Bennaceur et al. (2014), não há atualmente nenhuma forma sistemática que permita a geração automática de meta-modelos de modelos em tempo de execução a partir dos meta-modelos utilizados em tempo de desenvolvimento. Mais ainda, uma vez que um modelo em tempo de execução deve representar fielmente uma aplicação em execução, é necessário o estabelecimento de uma conexão causal entre o modelo e a aplicação. Neste contexto, também não há uma forma sistemática de realizar a manutenção dessa conexão.

A despeito da ausência de sistemática, diferentes trabalhos na área apontam importantes similaridades no que diz respeito à natureza das informações representadas. Segundo Breton Breton e Bézivin (2001), esses meta-modelos devem possuir: (a) uma parte estática, descrevendo a estrutura dos modelos, e (b) uma parte dinâmica, descrevendo as situações nas quais o modelo (e, consequentemente, a aplicação) pode se encontrar e dados históricos.

De forma geral, a decisão acerca dos elementos que devem ser representados na parte dinâmica dos modelos está associada às necessidades de gerenciamento das aplicações. Assim, para que se possa definir o conjunto de elementos representados na parte dinâmica deve-se refletir acerca de como um gerente de adaptação pode fazer uso desses elementos para conduzir atividades que compõem o seu laço de controle.

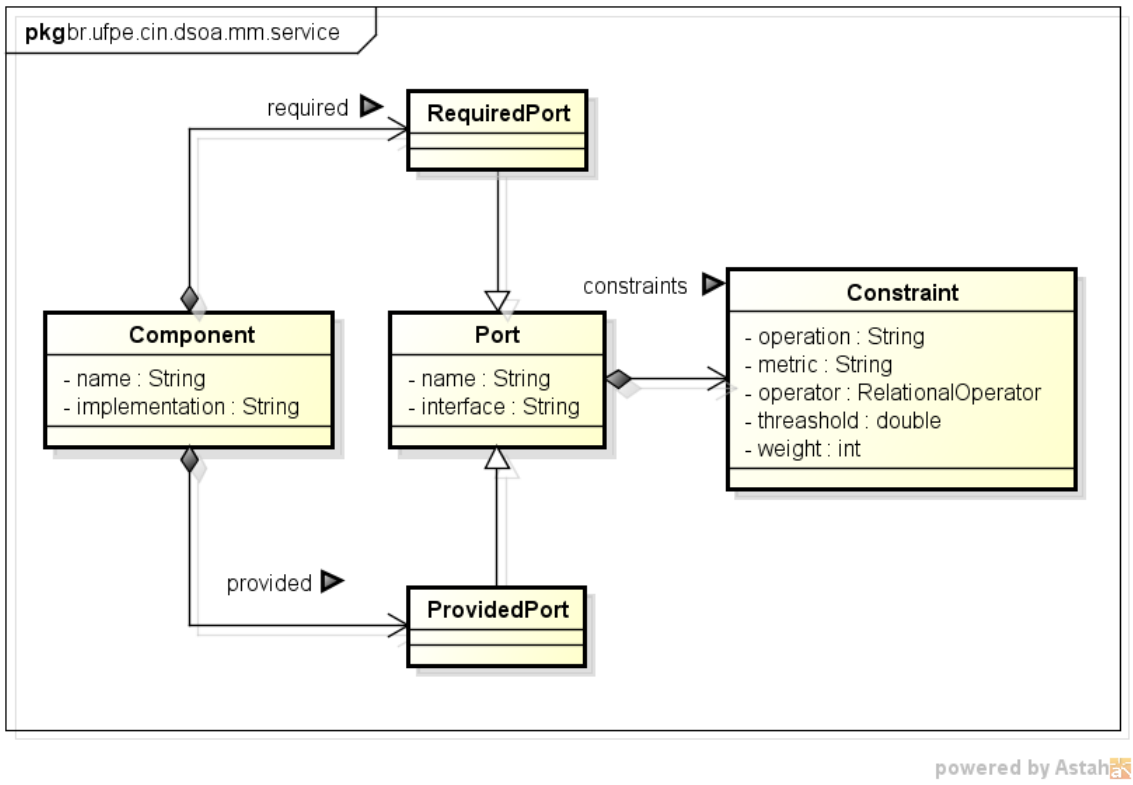
Considerando-se a conexão causal existente entre o modelo e a aplicação em execução, pode-se concluir que os elementos dinâmicos possuem uma dupla finalidade. De um lado, esses elementos possuem uma natureza descritiva, devendo apresentar para o módulo de supervisão do gerente de adaptação, uma visão acerca do estado atual (e, eventualmente, passado) da aplicação. Do ponto de vista do módulo de adaptação, o modelo em tempo de execução deve apresentar elementos de natureza prescritiva, os quais são utilizados para descrever o estado desejado do sistema. Por fim, Lehmann et al. (2011) defende que os meta-modelos que representam os modelos de tempo de execução também representem os elementos que permitem mudanças de estado, as quais podem envolver tanto a parte prescritiva quando a descritiva.

Neste contexto, o restante dessa seção apresenta os elementos que compõem o meta-modelo dos modelos de tempo de execução utilizados pela plataforma DSOA, indicando as partes estáticas e dinâmicas, e destacando sua natureza prescritiva ou descritiva.

6.2.1 Domínio de Serviços

O meta-modelo dos modelos de tempo de execução da plataforma DSOA define o conjunto de elementos que podem ser utilizados para representar uma aplicação em execução. Esses elementos compõem um modelo abstrato de componentes baseado em serviços, representando conceitos bem-estabelecidos, tais como, componentes e portas.

Figura 23 – Componentes e Portas



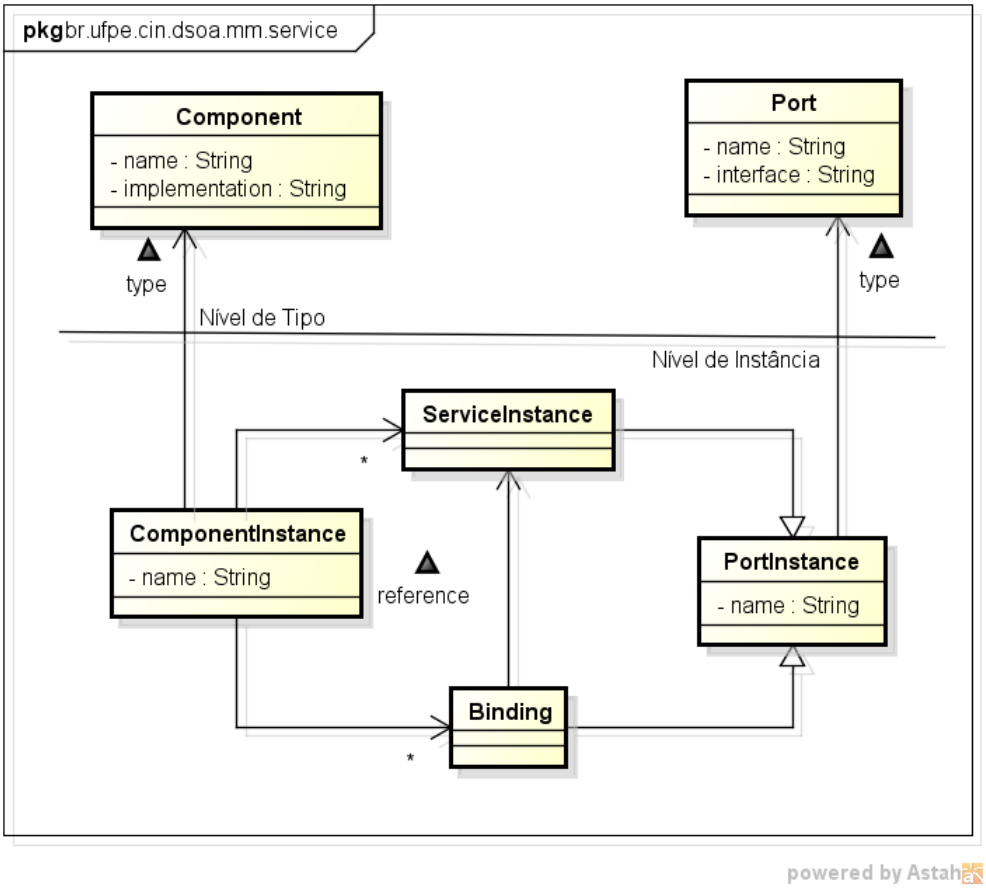
Fonte: o autor

Para desenvolver uma aplicação na plataforma DSOA, o primeiro passo é conceber os *tipos de componentes* que vão formar uma instância dessa aplicação e os serviços que cada um desses tipos fornece e/ou requer. Cada tipo de componente possui uma implementação que determina o seu comportamento, e estabelece como ele implementa os serviços providos e faz uso dos serviços requeridos. Por sua vez, cada serviço possui uma interface que define suas funcionalidades através da especificação de um conjunto de operações, às quais são impostas restrições relativas ao nível de qualidade esperado. A **Figura 23** apresenta como esses conceitos foram representados no meta-modelo da plataforma.

Como podemos ver, um tipo de componente, representado pela meta-classe *Component*, possui um nome e uma implementação (na versão de referência da plataforma, essa implementação indica o nome de uma classe Java). Cada tipo de componente define como suas instâncias podem interagir através da especificação de portas. Do ponto de vista conceitual, uma porta (modelada pela meta-classe *Port*) representa um ponto de comunicação através do qual um tipo de serviço pode ser provido ou requerido. Assim, cada porta deve definir as características funcionais e não-funcionais do tipo de serviço correspondente.

Do ponto de vista funcional, um tipo de serviço é descrito através do nome de uma interface que define suas operações. Por sua vez, a perspectiva não-funcional é descrita através de um conjunto de restrições (*Constraints*) estabelecidas sobre as métricas de qualidade, as quais devem ser observadas pelo gerente de adaptação quando do acesso às

Figura 24 – Instâncias de componentes e de portas



Fonte: o autor

operações desse tipo de serviço. Opcionalmente, uma restrição pode indicar um peso, o qual pode ser utilizado no processo de seleção dos serviços que devem ser consumidos.

É importante observar que os conceitos descritos até o presente ponto representam a parte estática dos modelos de tempo de execução. Por meio deles, não é possível representar nenhuma informação acerca do estado das instâncias dos componentes e serviços que formam uma aplicação. De fato, enquanto os conceitos representados objetivam descrever "tipos" de componentes e serviços, uma aplicação em execução é composta por instâncias desses elementos, as quais devem ser representadas através da introdução de elementos de modelagem capazes de expressar as características relevantes relacionadas execução e gerenciamento dessas instâncias. Em outras palavras, para que as instâncias dos componentes das aplicações possam ser gerenciadas, é necessário representá-las de forma explícita nos modelos de tempo de execução. Para que essa representação seja possível, é necessária a introdução de elementos correspondentes no meta-modelo, os quais são apresentados na **Figura 24**.

Como se pode observar, cada instância possui um nome, que a identifica unicamente no contexto da aplicação, e dois conjuntos de pontos de comunicação (*PortInstance*s). Cada

um desses pontos representa: (1) a necessidade de conexão com um serviço requerido (*Binding*), ou (2) uma instância de serviço provido (*ServiceInstance*). A descrição das interfaces e restrições aplicáveis aos serviços correspondentes a cada um desses pontos está na na meta-classe *Port* que descreve o tipo da porta em questão.

Para identificar as demais características das instâncias que devem ser representadas nos modelos em execução deve-se considerar as ações que podem ser realizadas sobre elas do ponto de vista do gerenciamento. Como vimos, essas ações estão relacionadas às atividades de supervisão e adaptação. Do ponto de vista da supervisão, as ações dizem respeito à observação do estado da aplicação e dos seus elementos. Para disponibilizar essas informações, os modelos devem incluir elementos descritivos, os quais podem ser consultados dinamicamente pelo gerente. Por outro lado, as ações de adaptação possuem a intenção de modificar o estado atual do sistema. Essas ações devem utilizar os elementos prescritivos dos modelos para estabelecer o estado desejado, promovendo, via conexão causal, uma alteração da aplicação em execução.

6.2.1.1 Elementos Descritivos

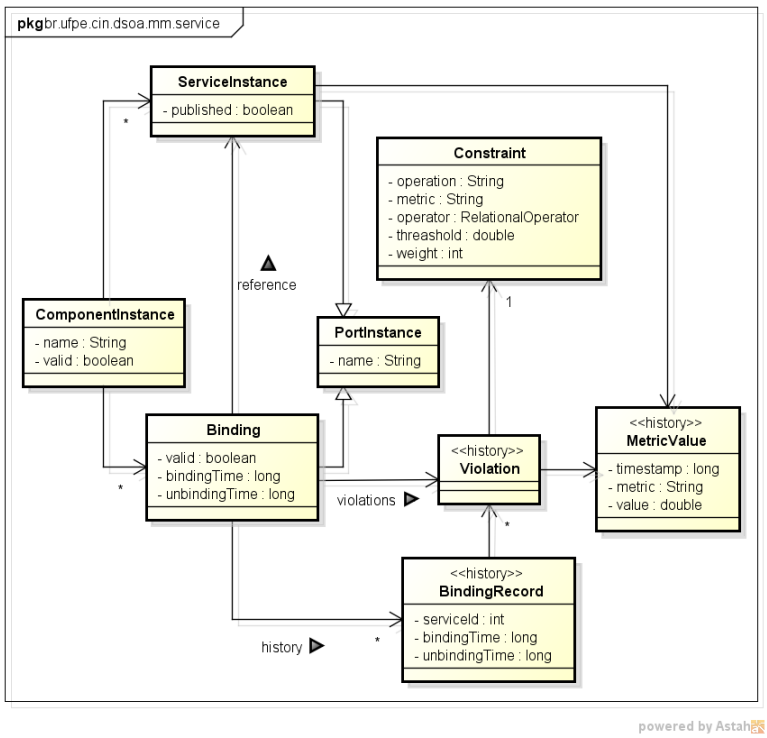
Os elementos descritivos modelados na plataforma DSOA estão representados na **Figura 25**. No contexto da plataforma, para que uma instância de componente possa entrar em execução e, eventualmente, fornecer seus próprios serviços, ela deve estar válida. Uma instância de componente somente pode ser considerada válida se todos os serviços requeridos por ela estiverem disponíveis e conectados às respectivas portas de comunicação.

De forma similar, uma instância de porta que representa uma ligação a um serviço requerido (*Binding*) pode estar válida ou inválida. Essa porta é considerada válida se ela estiver conectada a uma instância de serviço (*ServiceInstance*) que possua características compatíveis com a sua especificação. Essa compatibilidade requer dois elementos. Em primeiro lugar, o serviço requerido pela ligação e a instância de serviço devem possuir uma mesma interface funcional. Além disso, as caraterísticas não-funcionais dessas portas, representadas através de *Constraint*s, devem ser compatíveis, de forma que as restrições de qualidade anunciadas pela instância de serviço sejam suficientes para atender às restrições associadas à ligação em questão. Para representar essas informações, as meta-classes *ComponentInstance* e *Binding* devem possuir um indicador de validade. Mais ainda, um *Binding* válido deve possuir uma referência para a instância de serviço correntemente utilizada.

Visando prover mais informações ao gerente de adaptação, incluímos no *Binding* informação temporal. Em particular, foram introduzidos dois campos descritivos *bindingTime* e *unbindingTime* denotando o momento em que foi realizada a última ligação com um serviço, e quando esse último serviço foi desconectado. Quando um *Binding* está válido, o *unbindingTime* deve ser nulo.

Para permitir o registro de informações históricas foram introduzidos dois elementos

Figura 25 – Elementos descritivos



Fonte: o autor

no meta-modelo. O elemento *Violation* representa uma violação de uma restrição e possui o valor observado para a métrica, junto com a etiqueta de tempo, e uma referência para a restrição violada. Assim, uma ligação válida com um serviço (*Binding*) possui uma lista de violações. Quando a ligação se torna inválida, é gerado um registro (*BindingRecord*) que armazena a lista de violações observadas enquanto a ligação estava válida. A introdução desse conjunto de elementos nos modelos em tempo de execução visa possibilitar que o gerente tenha mais informação para decidir se uma ligação deve ou não permanecer válida após uma violação. Em caso negativo, o gerente pode observar os dados históricos para selecionar o próximo serviço a ser utilizado.

No processo de seleção, deve-se identificar as instâncias de serviço que estão correntemente disponíveis e que, em princípio, são adequadas (de acordo com as restrições estabelecidas). Assim, cada instância de serviço contém um indicador que denota se ela está ou não publicada no registro e, portanto, disponível. No modelo da plataforma, quando uma instância de componente está válida, os serviços são automaticamente publicados, sendo removidos quando a instância é invalidada (e.g., em virtude da indisponibilidade de um serviço requerido). Adicionalmente, uma instância de serviço possui um mapa, associando valores monitorados às métricas que o serviço utiliza para declarar suas restrições.

Por fim, o meta-modelo da plataforma DSOA provê, ainda, operações associadas às respectivas meta-classes, as quais podem ser acessadas programaticamente para que um gerente de adaptação possa recuperar as informações descritas.

6.2.1.2 Elementos Prescritivos

Na plataforma DSOA, os elementos prescritivos modelados até o momento objetivam possibilitar a validação e invalidação das instâncias de componentes e das ligações. Como explicado previamente, a invalidação de uma ligação ocorre quando não existe uma instância de serviço disponível capaz de satisfazer às necessidades da instância de componente que possui a ligação em questão. Essa invalidação da ligação provoca a invalidação da própria instância de componente e a remoção dos serviços que esta instância provê do registro.

Por outro lado, a validação de uma ligação ocorre quando da seleção de uma instância de serviço que atenda aos requisitos estabelecidos. Estando todas as ligações válidas, a própria instância de componente se torna válida, e os serviços por ela oferecidos são publicados no registro. Para que um gerente de adaptação possa realizar uma ligação com um serviço selecionado e, eventualmente, desfazer essa ligação, foram introduzidas as operações de *bind* e *unbind* na representação da ligação.

Na plataforma DSOA, essas operações estão disponíveis apenas para o gerente de adaptação, que pode realizá-las durante a execução do seu laço fechado de controle. O acesso direto a essas operações pelos administradores e desenvolvedores foi intencionalmente removido visando minimizar a possibilidade de deixar o sistema em um estado inconsistente.

É importante ressaltar que, apesar de não poderem manipular diretamente as ligações, os desenvolvedores podem participar do laço de controle, interferindo nos processos de análise (responsável pela decisão acerca de substituir ou não um serviço em utilização), e de seleção de serviços. Para tanto, os desenvolvedores devem implementar políticas de avaliação e de seleção. Essas políticas são indicadas no modelo através do nome de classes Java (que devem implementar interfaces pré-estabelecidas), as quais são instanciadas via reflexão durante a configuração do gerente de adaptação.

Do ponto de vista da análise, a política possui um método chamado *evaluate* que recebe o registro histórico da ligação, a restrição violada, e o último valor observado para a métrica correspondente. O resultado desse método é um valor lógico, indicando se o serviço em utilização deve ou não ser substituído. Caso seja necessária a substituição, o gerente de adaptação utiliza a política de seleção. Atualmente, a interface da política de seleção compreende um único método que recebe a lista de restrições da ligação e uma lista de instâncias de serviços candidatos. Por fim, cabe mencionar que, embora a interface atual dessas políticas seja simples, elas fornecem uma boa flexibilidade, permitindo que o desenvolvedor participe ativamente dos processos de análise e seleção.

6.2.1.3 Comunicação com o Modelo e Conexão Causal

Segundo Lehmann Lehmann et al. (2011), uma etapa importante durante a concepção de um meta-modelo para modelos de tempo de execução corresponde à identificação dos elementos de comunicação entre esses modelos e a plataforma subjacente (na qual a aplicação está em execução). Através desses elementos, a plataforma é informada acerca de modificações nos elementos prescritivos dos modelos, podendo refletir tais modificações na aplicação em execução. No sentido inverso, os elementos de comunicação permitem que a plataforma atualize os elementos descritivos dos modelos, de forma que esses possam refletir o estado atual da aplicação.

Na plataforma DSOA, os modelos de execução formam uma camada de abstração entre o gerente de adaptação e o ambiente de execução, e, para exercer o seu papel, o modelo precisa interagir com ambos. Por exemplo, quando é necessário substituir um serviço em utilização, o gerente de adaptação seleciona uma instância de serviço disponível. Para efetivar essa substituição na aplicação em execução, o gerente deve atualizar a representação da ligação no modelo. Visando manter a conexão causal, essa ação deve ser refletida na própria aplicação em execução.

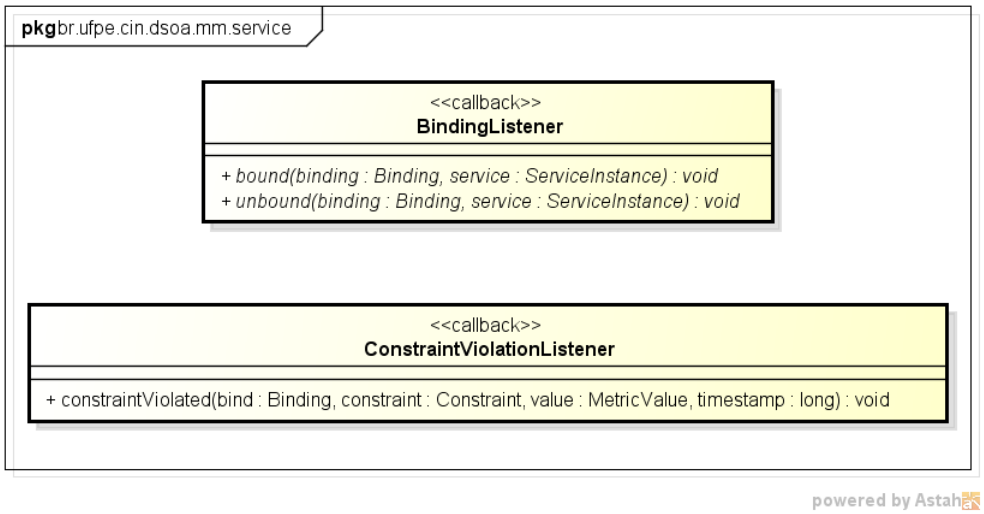
Por outro lado, quando uma restrição é violada na aplicação, essa violação é identificada pelo mecanismo de monitoração da plataforma, que deve comunicá-la ao modelo, de forma que o elemento descritivo correspondente possa refletir a violação ocorrida. Nesse momento, o modelo deve transmitir a informação ao gerente para que este avalie se é necessária, ou não, a substituição do serviço em utilização.

A despeito dessa necessidade de comunicação, os modelos não devem, idealmente, possuir ciência acerca do gerente de adaptação ou do ambiente de execução específico no qual o modelo está situado. Visando manter os modelos independentes, de forma a facilitar a sua reutilização, a plataforma DSOA representa em seu meta-modelo interfaces que devem ser implementadas pelo gerente e pelo ambiente subjacente para que esses possam ser devidamente notificados. A **Figura 26** apresenta as interfaces que foram introduzidas no meta-modelo.

Em particular, para que um ambiente de execução seja notificado quando uma ligação é realizada ou desfeita no modelo, a interface *BindingListener* deve ser implementada. Por outro lado, para que um gerente de adaptação seja notificado pelo modelo acerca de uma violação de restrição em uma ligação, ele deve implementar a interface *ConstraintViolationListener*. Essa interface funciona como um mecanismo de *callback*, e possui apenas um método através do qual o gerente é informado sobre a ligação na qual aconteceu a violação, a restrição que foi violada, o valor observado para a métrica indicada na restrição, e uma etiqueta de tempo.

Por fim, o próprio modelo deve implementar interfaces para que este seja notificado acerca dos eventos relevantes ocorridos ao longo da execução da aplicação. Neste sentido, a interface *MetricUpdatedListener* permite que uma representação de uma instância de

Figura 26 – Interfaces para notificação



Fonte: o autor

serviço no modelo seja informada quando da computação de uma métrica associada a uma de suas operações. Do ponto de vista das notificações de violação, a própria interface *ConstraintViolationListener* é utilizada para que as representações das ligações no modelo possam ser notificadas.

Como se pode observar, as métricas de qualidade possuem um papel central no gerenciamento da adaptação realizado no contexto da plataforma DSOA. Embora a presente seção tenha apresentado como tais métricas podem ser utilizadas no estabelecimento das restrições associadas aos serviços, até aqui não apresentamos como essas métricas são definidas e organizadas na plataforma.

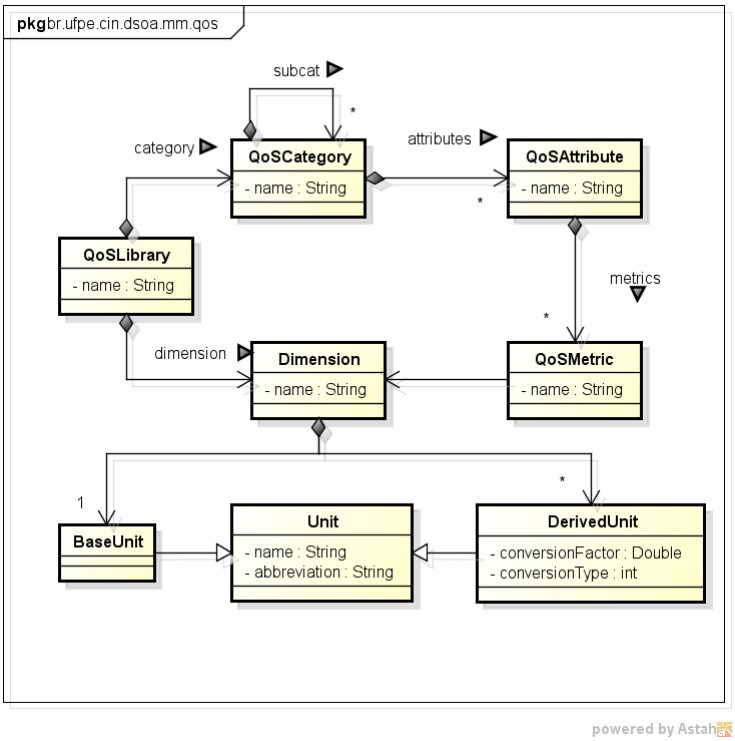
6.2.2 Domínio de Qualidade

Como discutido anteriormente, a despeito da relevância das características de qualidade no contexto de SOA, não existe um modelo de qualidade de serviço único e universalmente aceito (KRITIKOS et al., 2013; ORIOL; MARCO; FRANCH, 2014). Visando oferecer uma solução flexível para esse problema, a plataforma DSOA permite que cada aplicação estabeleça seu próprio vocabulário através da definição de um modelo de qualidade customizado.

Para viabilizar essa definição, a plataforma DSOA propõe a introdução de um conjunto de elementos representando, de forma simplificada, os conceitos do domínio de qualidade. Esses elementos são apresentados na **Figura 27** e representam conceitos genéricos, tais como atributos e métricas de qualidade.

Na plataforma DSOA, cabe a um perfil de desenvolvedor específico, referenciado como especialista em qualidade, a responsabilidade pela identificação das características de qualidade relevantes do ponto de vista de uma aplicação e pela representação dessas

Figura 27 – Taxonomia de qualidade



Fonte: o autor

características através de um modelo de qualidade.

Enquanto isso, os desenvolvedores da aplicação se concentram na lógica de negócio, podendo fazer uso do modelo de qualidade para estabelecer as restrições definidas como parte dos requisitos não-funcionais da aplicação. Como vimos anteriormente, essas restrições são associadas aos serviços requeridos e providos pelos componentes, estabelecendo o nível de qualidade esperado desses elementos.

Para ser capaz de "compreender" esses modelos, a própria plataforma foi concebida em torno de elementos de qualidade genéricos, definidos como parte de seu meta-modelo. Assim, os modelos de qualidade, concebidos em tempo de desenvolvimento, são traduzidos em uma hierarquia de objetos mantida em execução como parte do repositório de serviços ciente de qualidade. A plataforma DSOA permite o acesso aos modelos de qualidade através de ferramentas de administração, viabilizando não só a visualização das características de qualidade já definidas, como a definição dinâmica de novas características.

Para permitir a elaboração de novos modelos de qualidade, o meta-modelo introduz a meta-classe *QoSLibrary*. Cada modelo de qualidade, identificado na plataforma através de um nome único, possui uma coleção de categorias, as quais permitem organizar os atributos de qualidade de forma hierárquica. Uma categoria pode conter sub-categorias, mas somente as folhas da hierarquia devem conter atributos de qualidade.

Na visão da plataforma DSOA, um atributo de qualidade representa uma propriedade não-funcional inerentemente dinâmica e que pode ser efetivamente mensurada, como por

exemplo, *tempo de resposta*. De uma forma geral, a avaliação de um atributo de qualidade pode ser realizada através de diferentes pontos de vista. Cada ponto de vista é representado no meta-modelo por uma métrica de qualidade. Na plataforma DSOA, uma métrica pode ter uma dimensão. Neste caso, cada valor associado à métrica deve indicar uma unidade correspondente. Cada dimensão possui uma unidade base e um conjunto de unidades derivadas associadas a um fator de conversão. Considerando-se essa estrutura simples, cada métrica na plataforma pode ser unicamente identificada através da composição dos nomes do modelo, das categorias, do atributo e da própria métrica.

Para ajudar a compreensão dos conceitos de atributo e métrica de qualidade, cabe um exemplo. Suponha que o *tempo de resposta* de um serviço utilizado seja importante para uma aplicação. Neste contexto, o conceito de tempo de resposta pode ser representado através de um atributo da categoria de desempenho. Contudo, existem diferentes formas de se avaliar tempo de resposta. Uma possibilidade é através da média, outra é através de um valor máximo aceitável. Cada uma destas perspectivas pode ser representada na plataforma através de uma métrica distinta.

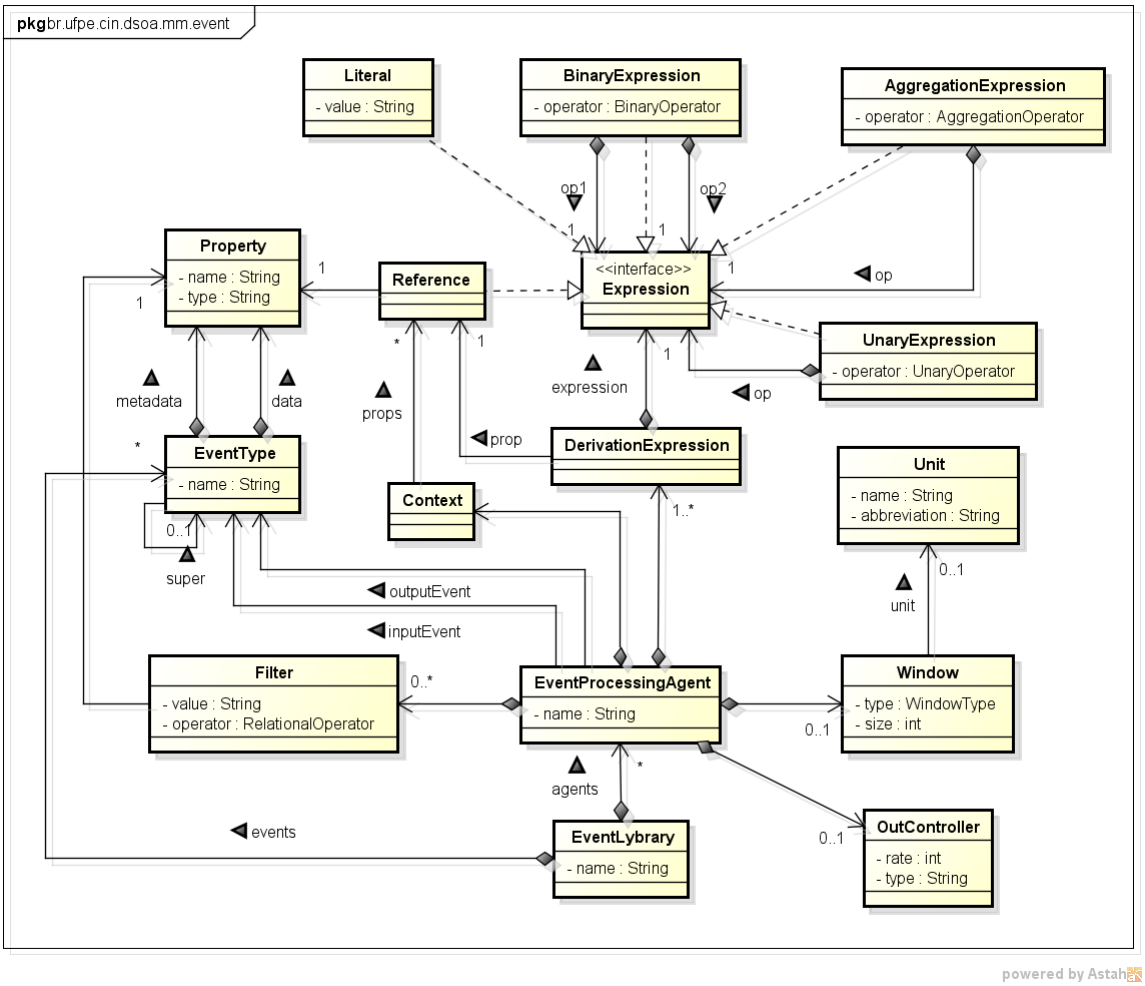
Como se pode perceber, os elementos de qualidade representados no meta-modelo são genéricos, permitindo que cada aplicação defina sua própria taxonomia de qualidade através da concepção de um modelo de qualidade simples e que pode ser facilmente estendido através da criação dinâmica de instâncias das meta-classes correspondentes.

Por fim, deve-se destacar que, embora os conceitos representados no meta-modelo sejam semelhantes aos apresentados em outras propostas (e.g., (KRITIKOS et al., 2013)), há uma diferença fundamental: o modelo de qualidade da plataforma DSOA não descreve como as métricas devem ser computadas. Para entendermos o porquê, continuemos com o exemplo de tempo de resposta avaliado através de uma métrica como tempo médio.

Existem diferentes formas de calcular o tempo médio de resposta. Uma forma simples é utilizar a média aritmética dos tempos observados. Outra é a utilização de uma métrica ponderada de forma a indicar que os valores obtidos recentemente tenham maior relevância. Mesmo no caso de uma média simples, uma pergunta razoável seria, quantas requisições devem ser consideradas para a obtenção da média? Diversas respostas seriam possíveis, tais como todas as requisições feitas até então, ou ainda, as requisições feitas na última hora. Todos estes pontos de vista são possíveis e válidos.

Para resolver esse problema, a plataforma separou a definição das métricas de qualidade da lógica responsável pela sua computação. A definição dessa lógica é um aspecto fundamental da semântica de uma métrica. Do ponto de vista da plataforma DSOA, cada aplicação pode decidir como suas métricas de qualidade devem ser computadas, estabelecendo a semântica adequada para a métrica no seu contexto de negócio. Como veremos, essa definição é realizada através da utilização de conceitos pertencentes ao domínio de eventos.

Figura 28 – Domínio de eventos



Fonte: o autor

6.2.3 Domínio de Eventos

Na visão da plataforma DSOA, as métricas de qualidade devem ser computadas a partir das ocorrências em tempo de execução, as quais são representadas através de eventos. Para viabilizar essa perspectiva, a plataforma foi concebida de forma que as ocorrências relevantes, tais como a publicação ou a invocação de um serviço, produzam eventos (referenciados como eventos primitivos) que podem ser observados para extrair a informação necessária para a computação das métricas de qualidade.

A plataforma DSOA permite que os desenvolvedores utilizem os eventos primitivos para definir seus próprios eventos através de modelos. Para viabilizar a definição desses modelos, a plataforma introduziu os conceitos de eventos em seu meta-modelo (vide **Figura 28**). No centro desses modelos estão os conceitos de eventos e agentes de processamento.

O conceito de evento é representado pela meta-classe *EventType*. Cada instância dessa meta-classe especifica a definição de um tipo de evento, que é identificado por um nome

único no contexto de uma biblioteca. Um tipo de evento possui duas coleções de tipos de propriedades, referenciadas como dados e meta-dados. Do ponto de vista conceitual, as propriedades declaradas como dados visam detalhar o quê efetivamente aconteceu, enquanto aquelas declaradas como meta-dados objetivam descrever as circunstâncias nas quais o evento ocorreu.

Uma vez que cada tipo de evento é representado, em execução, por uma instância da meta-classe *EventType*, é possível a definição dinâmica de novos tipos de eventos sem que haja necessidade de modificação no código da plataforma. Assim, para que uma aplicação possa definir um novo tipo de evento, ela deve indicar o seu tipo e as propriedades que um evento desse tipo deve conter.

Visando simplificar a especificação de novos tipos de eventos, a plataforma incorporou, em seu meta-modelo, um modelo de herança, definindo que um tipo de evento pode ter um super-tipo. Essa abordagem, além de eliminar a necessidade de declaração de propriedades redundantes, permite que os modelos armazenem relações de generalização e especialização entre eventos. Por sua vez, essas relações permitem que consumidores de eventos possam receber eventos de mais de um tipo através de uma única subscrição para receber evento de um super-tipo comum.

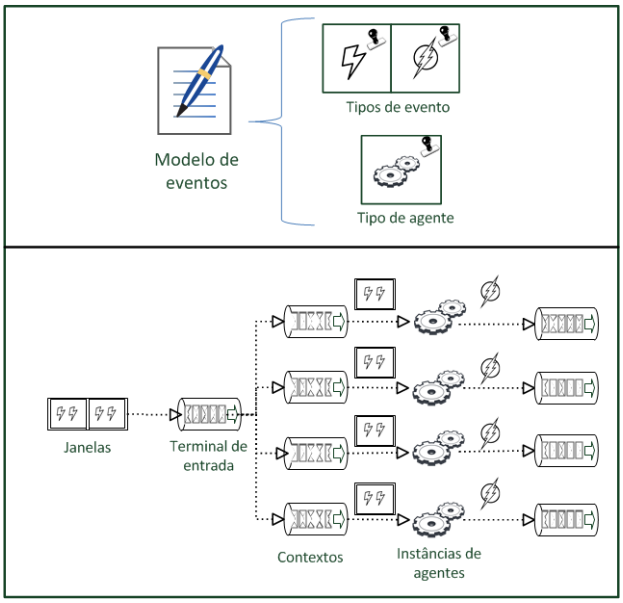
Apesar da introdução da meta-classe *EventType* tornar bastante simples a definição de novos tipos de eventos, essa representação não é suficiente para suportar as necessidades de um mecanismo de monitoração capaz de supervisionar o nível de qualidade dos serviços. De fato, a informação de qualidade é, em geral, de mais alto nível do que aquela obtida diretamente dos eventos primitivos emitidos pelos sensores da plataforma. Assim, torna-se necessária a definição de um mecanismo de processamento capaz de transformar os eventos primitivos em eventos de mais alto nível contendo as informações relativas às métricas de qualidade.

Para representar essa capacidade de processamento, foi introduzida no meta-modelo da plataforma uma representação do conceito de agente de processamento (*EventProcessingAgent*). Um agente de processamento é responsável por realizar transformações entre tipos de eventos. Cada agente possui um nome, uma descrição, um terminal de entrada, um terminal de saída, uma coleção de "expressões de derivação", e, opcionalmente, um contexto, uma janela de entrada, e um controle de frequência de saída.

O terminal de entrada define o "fluxo" (do inglês, *stream*) de eventos que o agente recebe para processar. Na plataforma DSOA, a especificação do terminal de entrada é realizada através da indicação do tipo de evento que o agente tem interesse em receber. Adicionalmente, um agente pode especificar uma coleção de filtros sobre as propriedades dos eventos de entrada, definindo as características particulares dos eventos a serem processados.

Os eventos recebidos pelo terminal de entrada passam por um processo de transformação para dar origem às instâncias de um novo tipo de evento, as quais são encaminhadas

Figura 29 – Agentes de processamento: janelas e contexto



Fonte: o autor

através do terminal de saída do agente. Para realizar essa transformação, o agente aplica uma coleção de expressões de derivação às propriedades dos eventos de entrada. Cada expressão de derivação computa o valor de uma propriedade do evento de saída. Conforme pode-se observar na **Figura 28**, a linguagem utilizada nessas expressões permite o uso de literais, operações aritméticas, e operações de agregação, tais como *sum, min, max, average, e standard deviation.*

Além desses elementos, a especificação de um agente pode incluir elementos de particionamento dos eventos de entrada através dos conceitos de janela e contexto (ETZION; NIBLETT, 2010). Esses conceitos são representados esquematicamente na **Figura 29**.

Uma janela é utilizada para particionar o "fluxo" de eventos de entrada com base no tempo ou na quantidade de eventos recebidos. As janelas são tipicamente utilizadas para computar métricas agregadas, tais como tempo médio de resposta. Assim, cada grupo de eventos recebidos ao longo de uma janela é processado em conjunto.

Por outro lado, um contexto especifica um conjunto de propriedades do tipo de evento de entrada que são utilizadas para segmentar o "fluxo" de eventos. Em tempo de execução, uma definição de agente incluindo uma especificação de contexto origina uma coleção de instâncias de agentes, cada uma responsável por processar uma partição de eventos criada com base no conteúdo das propriedades que compõem a definição do contexto (vide **Figura 29**).

Por exemplo, uma única declaração de agente é necessária para computar o tempo médio de resposta de cada serviço individual. Para tanto, deve-se definir um agente que recebe, através do terminal de entrada, os eventos que reportam as invocações de serviços. Esses eventos, representados na plataforma na forma de um evento primitivo (*InvocationE-*

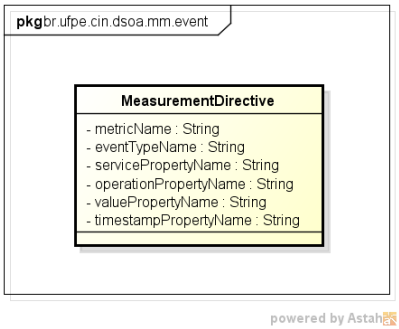
vent), contém um identificador do serviço requisitado. Tal identificador pode ser utilizado como um elemento de contexto na declaração de um agente. Em tempo de execução, essa informação é utilizada de forma que as invocações de serviços distintos sejam processadas por diferentes instâncias de agentes, isolando o processamento dos dados referentes a cada serviço individual.

Por fim, uma definição de agente pode incluir um controle de saída. A finalidade desse controle é determinar a frequência com a qual os eventos computados pelos agentes de processamento devem ser emitidos através do seu terminal de saída, controlando a quantidade total de eventos em circulação. Um controle é configurado através de dois parâmetros: (1) uma taxa definindo a dimensão do controle (frequência de saída), e (2) um tipo, definindo se essa frequência corresponde a uma quantidade de eventos ou ao tempo. Por exemplo, o controle de saída pode ser utilizado para especificar que o tempo médio de resposta de um serviço, computado por um agente com janela de 60 minutos, esteja disponível a cada 30 minutos. Desta forma, a cada 30 minutos o agente produz um evento complexo informando o tempo médio de resposta computado com base nos eventos detectados ao longo dos últimos 60 minutos.

Na plataforma DSOA, os agentes de processamento são responsáveis por transformar os eventos primitivos (tais como evento de invocação, publicação, e ligação) em eventos de mais alto nível contendo os valores computados para as métricas de qualidade. Para que se possa especificar como os eventos produzidos pelos agentes podem ser transformados em valores de métricas de qualidade, o desenvolvedor utiliza uma diretiva de monitoração (vide **Figura 30**).

Como se pode observar, uma diretiva de monitoração possui uma indicação do nome da métrica a ser computada e do tipo de evento que transporta a informação da métrica. Mais ainda, uma diretiva deve indicar as propriedades que contêm: (1) o nome do serviço, (2) o nome da operação, (3) o valor computado para a métrica, e (4) a etiqueta de tempo que indica quando o evento ocorreu. Essas informações possuem duas finalidades. Primeiramente, elas são utilizadas pelo serviço de monitoração para identificar qual tipo de evento ele deve escutar para monitorar uma dada métrica de qualidade. Em segundo lugar, eles são utilizados para converter as informações do evento em valores de métricas que devem ser utilizados para atualizar os modelos.

Figura 30 – Diretiva de monitoração



Fonte: o autor

6.3 MODELOS DE DESENVOLVIMENTO E LINGUAGENS

Embora os meta-modelos descritos tenham a finalidade de representar os modelos mantidos em tempo de execução, grande parte da informação preenchida nesses modelos advém de modelos de desenvolvimento. Como mencionado anteriormente, esses modelos são descritos em linguagens específicas de domínio implementadas utilizando *xText* e transformados em modelos específicos de plataforma que são interpretados pelo serviço de configuração.

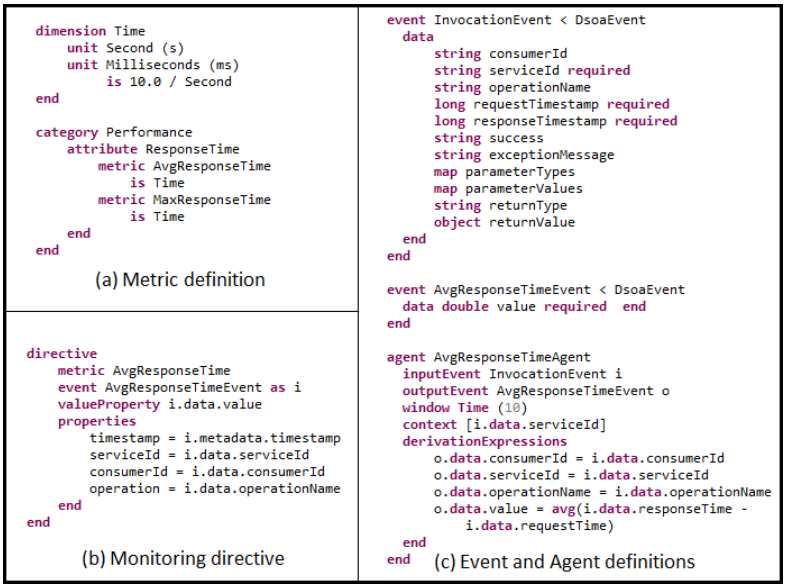
Dado que os modelos de desenvolvimento devem conter grande parte das informações que compõem os modelos de execução, é natural que um meta-modelo representando a sintaxe abstrata desses modelos de desenvolvimento possua elementos similares aos que compõem os meta-modelos de tempo de execução.

Apesar da similaridade há importantes diferenças, as quais estão relacionadas ao propósito desses meta-modelos. Um modelo de tempo de execução é similar ao conceito de modelo semântico apresentado por (FOWLER, 2010). Segundo os autores, um modelo semântico é uma representação, tal como um modelo de objetos, dos conceitos de um domínio, mantido em tempo de execução. Assim, a informação deve ser estruturada de tal forma que sua utilização seja natural.

Um modelo semântico, além de estruturar os dados, pode conter comportamento. Por outro lado, os meta-modelos que descrevem a sintaxe abstrata (e parte da semântica estática) das DSLs são normalmente utilizados para definir os elementos que fazem parte da árvore sintática da linguagem, sendo naturalmente orientados ao aspecto sintático. Em geral, nem todo elemento do meta-modelo de execução faz parte do meta-modelo de desenvolvimento, e vice-versa.

Exemplos de elementos do meta-modelo de tempo de execução execução que não fazem parte do meta-modelo de desenvolvimento são os elementos que detalham as instâncias de serviços e as ligações. De fato, embora os modelos de desenvolvimento incluam uma descrição detalhada dos tipos de componentes e dos serviços requeridos e fornecidos por eles, a descrição das instâncias de componentes se limita a uma indicação do nome e do

Figura 31 – Sintaxe concreta das DSLs



Fonte: o autor

tipo de componente correspondente. As demais informações, tais como se a instância está ou não válida, e quais instâncias de serviços estão sendo utilizadas, são obtidas em tempo de execução e controladas pelo gerente de adaptação embutido no contêiner. Uma vez que essa informação é dinâmica e conhecida somente em tempo de execução, não há motivo para representar esses elementos no meta-modelo de desenvolvimento.

No sentido inverso, o meta-modelo de desenvolvimento possui elementos para permitir a descrição dos métodos que fazem parte das interfaces dos serviços, incluindo os respectivos parâmetros e tipo de retorno. Embora essas informações sejam úteis para a geração código da interface em tempo de desenvolvimento e para a validação dos modelos que definem as restrições associadas aos serviços (as quais devem indicar a operação alvo), esse nível de detalhamento não é utilizado em tempo de execução. De fato, visando facilitar a navegação no modelo de tempo de execução e restringir o número de classes carregadas em memória, a interface é representada nos modelos de execução através do seu nome.

Para introduzir, de forma simples, as DSLs utilizadas na plataforma, a **Figura 31** apresenta uma parte da configuração de uma aplicação. Como pode ser observado, a sintaxe concreta das linguagens é declarativa, e os seus elementos são bastante próximos dos conceitos apresentados nos meta-modelos de execução.

A **Figura 31** (a) apresenta a definição de duas métricas relacionadas ao tempo de resposta: tempo máximo de resposta (*MaxResponseTime*) e tempo médio de resposta (*AvgResponseTime*) utilizando-se DSOA-QL, a linguagem para descrição de qualidade na plataforma DSOA. Embora a sintaxe abstrata de DSOA-QL não tenha sido apresentada, pode-se perceber que os elementos que compõem essa linguagem são equivalentes àqueles representados no meta-modelo de qualidade. De fato, as características dos modelos de

qualidade utilizados em tempo de execução advém do tempo de desenvolvimento, de forma que os elementos da sintaxe abstrata dessa linguagem correspondem aos que são representados no meta-modelo dos modelos de qualidade mantidos em tempo de execução. Essa similaridade de representação simplifica o processamento realizado pelo serviço de configuração, o qual basicamente lê o modelo de desenvolvimento e cria os objetos equivalentes em execução, os quais são mantidos no repositório de qualidade da plataforma.

Como discutido previamente, as definições das métricas são parte de uma taxonomia concebida para especificar os atributos e métricas de qualidade que são utilizados na especificação dos requisitos não-funcionais das aplicações. Deve-se observar que essas definições não especificam como as métricas podem ser computadas.

O ponto de partida para entender o processo de computação das métricas é apresentado na **Figura 31**(c). Essa figura apresenta a definição em DSOA-EL do agente *AvgResponseTimeAgent*, o qual recebe e processa eventos que representam invocações de serviços (*InvocationEvent*s). Para computar a média, esses eventos são agrupados em janelas de dez segundos (unidade *default* de janelamento). Para que esses valores possam ser comunicados aos demais elementos da plataforma, eles são embutidos em eventos do tipo *AvgResponseTimeEvent*, os quais são encaminhados através do terminal de saída do agente de processamento. Uma observação importante concernente ao agente que computa a métrica é que ele utiliza o identificador da instância de serviço como elemento de contexto. Como vimos, a definição de um contexto faz com que uma mesma definição de agente dê origem a diversas instâncias, cada uma responsável por agregar e computar a média de um serviço específico.

Os elementos que compõem a sintaxe abstrata de DSOA-EL não são explicitamente apresentados. Novamente essa decisão decorreu do mapeamento direto que podemos fazer entre os elementos utilizados na declaração dos agentes e eventos, e os elementos representados no meta-modelo de execução. A razão dessa similaridade é a mesma, as informações dos modelos de desenvolvimento são todas transportadas para os modelos de tempo de execução e, os modelos de tempo de execução não representam tipos de informações diferentes daquelas que advém do desenvolvimento. Dessa forma, o meta-modelo que representa a sintaxe abstrata (e semântica estática) de DSOA-EL possui elementos semelhantes àqueles representados no meta-modelo de execução.

Por fim, uma vez que um evento contendo um valor computado para uma métrica é gerado no terminal de saída do agente, ele pode ser distribuído para os demais elementos da plataforma através de um barramento interno de eventos. Nesse contexto, os consumidores interessados em receber os valores das métricas devem realizar uma subscrição. Como veremos no capítulo seguinte, esse é o mecanismo utilizado pelo serviço de monitoração da plataforma. Ao ser solicitado para monitorar uma instância de serviço, o serviço de monitoração identifica as métricas associadas às restrições de qualidade descritas e se registra para ouvir os eventos correspondentes no barramento. Por exemplo, ao receber

um evento do tipo *AvgResponseTimeEvent*, o serviço de monitoração converte o evento em um valor de métrica (*MetricValue*), o qual é adicionado ao elemento de modelo que descreve a instância de serviço equivalente em tempo de execução. Adicionalmente, caso alguma ligação com a instância de serviço relacionada ao evento tenha suas restrições de qualidade violadas, o serviço de monitoração cria uma representação dessa violação (*Violation*) e a comunica à representação da ligação no modelo através do acionamento do método *constraintViolated*. Desta forma, os elementos descritivos do modelo são atualizados, realizando-se um lado da conexão causal.

O repositório de serviços ciente de qualidade e o barramento interno de eventos fazem parte do conjunto de serviços de suporte que compõem o ambiente de execução da plataforma, tendo sido concebidos para apoiar os contêineres, e em particular, os gerentes de adaptação neles embutidos, na condução do processo de adaptação dinâmica das QSBAs. Esses elementos são apresentados em detalhes no capítulo seguinte.

6.4 CONSIDERAÇÕES FINAIS

O presente capítulo apresentou em detalhes os meta-modelos dos modelos de tempo de execução utilizados na plataforma DSOA, os quais incorporam elementos dos domínios de componentes e serviços, qualidade, e eventos. Modelos em conformidade com esses meta-modelos são utilizados para representar as aplicações em execução, mantendo uma conexão causal com a mesma. Desta forma, um gerente de adaptação pode utilizar esses modelos para manter-se informado acerca do estado da aplicação e dos seus elementos. Pode ainda, via conexão causal, promover uma reconfiguração da aplicação, visando a garantia dos requisitos não funcionais estabelecidos através de restrições impostas à qualidade dos serviços.

Mais ainda, ao permitir a representação dos conceitos de eventos e agentes nos modelos de desenvolvimento, e a sua utilização para a criação de modelos equivalentes mantidos em tempo de execução, a plataforma permite que os desenvolveres informem ao gerente de adaptação como este pode “compreender” os objetivos de alto nível informados através das restrições de qualidade. De fato, esses conceitos permitem a criação de uma ponte de abstração entre as ocorrências de baixo nível verificadas na plataforma, e as informações de alto nível descritas como requisitos da aplicação.

O detalhamento de como esses modelos são incorporados e utilizados em tempo de execução é realizado no capítulo seguinte, que descreve a implementação dos elementos que compõem a plataforma DSOA.

7 VISÃO DE IMPLEMENTAÇÃO

> *“ Eu ouço e eu esqueço. Eu vejo e eu lembro. Eu faço e eu entendo. ”*
>
> Provérbio Chinês,

O presente capítulo tem como objetivo apresentar os elementos que compõem a plataforma DSOA, detalhando como esses elementos interagem com os modelos previamente apresentados.

7.1 VISÃO GERAL

O capítulo anterior apresentou os elementos que constituem o meta-modelo dos modelos de tempo de execução adotados na plataforma DSOA. Como vimos, embora a plataforma utilize diferentes tipos de modelos para apoiar a atividade de gerenciamento, a adaptação em si é realizada pelo gerente através de um único modelo arquitetural reflexivo (causalmente conectado).

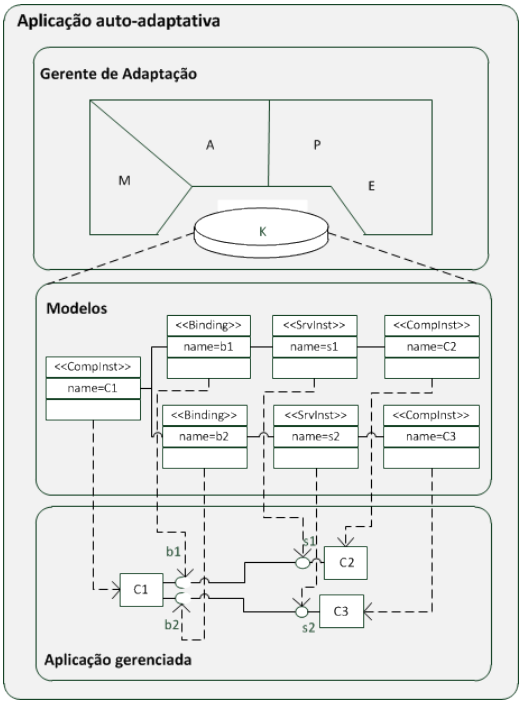
Na visão da plataforma DSOA, os modelos arquiteturais devem formar uma API de gerenciamento composta por elementos descritivos e prescritivos, os quais permitem, não só que o gerente de adaptação obtenha informações acerca do estado atual da aplicação, mas também que ele modifique esse estado, e, consequentemente, a aplicação em execução. A **Figura 32** apresenta a relação entre o gerente de adaptação, o modelo arquitetural, e uma aplicação em execução.

Para realizar essa visão de modelos como API de gerenciamento, os modelos de execução da plataforma DSOA foram projetados como “modelos ricos”, compreendendo dados e comportamento. Visando agregar esses aspectos, optou-se por utilizar classes Java para definir os tipos de meta-objetos que compõem os modelos de tempo de execução. Essa abordagem possui importantes implicações.

Um primeiro aspecto a ser observado é que evoluções na visão de como um componente deve ser representado em tempo de execução geram a necessidade de alteração das classes que reificam os componentes no plano meta. Outra consequência é que as ferramentas típicas de MDE não podem ser utilizadas em execução, uma vez que elas não são capazes de “compreender” as meta-classes definidas pela plataforma. Mais ainda, não é possível exportar/importar, de forma automática, a descrição dos tipos de componente presentes na plataforma.

Por outro lado, a utilização de meta-modelos para definir somente estruturas de dados promove uma separação entre dados e comportamento (FOWLER, 2010). Em geral, os trabalhos que utilizam essa abordagem (e.g., (HUBER et al., 2014a; WAIGNIER et al., 2008)) se baseiam em mecanismos de notificação para fazer com que um modelo comunique à parte responsável pelo comportamento, quando da ocorrência de mudanças de estado.

Figura 32 – Aplicação auto-adaptativa



Fonte: o autor

Uma consequência dessa separação é tornar o desenvolvimento mais complexo e dificultar eventuais manutenções.

No contexto da plataforma DSOA, os conceitos representados nos modelos arquiteturais são bastante genéricos e relativamente estáveis, podendo ser mapeados, sem grandes dificuldades, para conceitos equivalentes em outras plataformas. Além disso, a utilização de *frameworks* como EMF em tempo de execução tem gerado discussão, principalmente em função de questões relacionadas à demanda por recursos e desempenho (e.g., (FOUQUET et al., 2014)). Nos cenários envolvendo dispositivos com pequena capacidade, esse é um ponto importante. Diante desses aspectos, a utilização de classes para representar os tipos de meta-objetos nos pareceu uma escolha adequada.

Definido como as meta-classes seriam realizadas, o passo seguinte na implementação da plataforma foi decidir como os componentes e serviços deveriam ser implementados no plano base, viabilizando o mapeamento dos conceitos presentes no meta-modelo. Neste ponto, duas opções seriam possíveis. A primeira consiste em implementar diretamente as abstrações definidas no meta-modelo de tempo de execução, propondo um novo modelo de componentes orientados a serviços e ciente de qualidade. Essa abordagem é bastante complexa e trabalhosa. A outra opção consiste em estender uma plataforma existente criando uma nova camada de abstração composta pelos modelos de tempo de execução apresentados. No contexto desta tese, essa segunda abordagem foi a escolhida.

Diante do que foi exposto, o ponto de partida para a implementação do ambiente de execução da plataforma DSOA foi a definição do modelo de componentes orientado a

serviços a ser utilizado como base. A etapa seguinte consistiu em estabelecer uma forma de mapear os conceitos do modelo arquitetural genérico definido na plataforma para o modelo de componentes selecionado. Por fim, definido o modelo de componentes orientado a serviços e o mapeamento, foi possível integrar esses elementos em um ambiente de execução de forma que nem o gerente de adaptação nem os próprios modelos fiquem fortemente acoplados ao modelo de componentes subjacente.

7.2 DEFINIÇÃO DO MODELO DE COMPONENTES

Visando selecionar um modelo de componentes orientados a serviços de código aberto para ser utilizado como base, estabelecemos um conjunto de critérios.

7.2.1 Critérios

De imediato, um critério fundamental diz respeito ao suporte nativo aos conceitos de componentes e serviços e, consequentemente, ao padrão arquitetural SOA, o qual se baseia no conceito de registro de serviços para viabilizar a descoberta e a ligação dinâmica. Conforme vimos na Seção 2.2.3.3, um registro de serviços ideal deve, além de manter a descrição dos serviços disponíveis, suportar as operações de publicação, retirada, e atualização de serviços, assim como permitir a consulta dos serviços disponíveis e a notificação quando da publicação, retirada, ou modificação dos serviços registrados.

Em geral, nos modelos de componentes orientados a serviços, o código de negócio das aplicações não acessa diretamente o registro de serviços. Nesses modelos, o acesso a esse registro é realizado a partir de um contêiner, o qual é responsável por publicar, descobrir, e selecionar os serviços utilizados pelos componentes das aplicações. Esses contêineres implementam um mecanismo de injeção de dependências, através do qual as aplicações informam as características dos serviços requeridos, cabendo ao contêiner a seleção e injeção desses serviços na própria aplicação.

Neste contexto, um aspecto importante diz respeito à extensibilidade do contêiner. De fato, esse contêiner deve ser aberto e projetado para permitir a definição de extensões personalizadas, sem que essas tenham que ser embutidas diretamente no modelo original. Em particular, deve ser possível criar extensões que redefinam as interações entre o contêiner e o registro de serviços, assim como entre provedor e consumidor de serviços, permitindo a incorporação de características que não são nativamente suportadas por esse registro. No contexto da plataforma DSOA, a intenção dessas extensões é tornar o registro de serviços "ciente de qualidade".

Para viabilizar a reconfiguração dinâmica das aplicações, uma característica fundamental do modelo de componentes utilizado como base é a oferta de capacidades reflexivas, as quais, combinadas com o fraco acoplamento decorrente do uso de serviços, permitem a substituição dinâmica dos elementos de uma aplicação. De forma geral, uma extensão

de um contêiner permite que este seja capaz de tratar um aspecto não-funcional que não é nativamente suportado (e.g., desempenho, persistência, e segurança). Neste cenário, é essencial que o mecanismo utilizado para estender o contêiner permita que as próprias extensões possam definir novas capacidades reflexivas, as quais estão relacionadas à reificação dos aspectos tratados.

Considerando-se a natureza das QSBAs, outra característica importante é que o modelo utilizado como base não imponha grandes requisitos em termos de capacidade de memória e processamento, inviabilizando a execução dessas aplicações em dispositivos com poucos recursos. Um outro aspecto relacionado diz respeito ao desempenho. Em particular, é importante que tanto a plataforma utilizada como base, quanto as extensões implementadas pela plataforma DSOA, introduzam um impacto pequeno no desempenho, de forma a não comprometer significativamente a experiência dos usuários, em virtude, por exemplo, de uma redução na responsividade das aplicações.

Do ponto de vista do gerenciamento de mudanças, um aspecto considerado essencial é que não deve ser necessária a modificação do código fonte do modelo de componentes utilizado como base. Em geral, modificações dessa natureza inviabilizam a execução de aplicações construídas diretamente sobre essa plataforma. Mais ainda, provocam uma necessidade recorrente de manutenção em função de evoluções na plataforma adotada. Nesse contexto, uma característica importante diz respeito à capacidade de "modularização", ou seja, espera-se que eventuais extensões possam ser tratadas através da introdução de módulos separados, os quais devem ser facilmente integráveis.

Outro aspecto importante diz respeito ao modelo de programação associado ao modelo de componentes. Como vimos anteriormente, os modelos de componentes são concebidos em torno de um modelo abstrato responsável por definir os elementos conceituais suportados pelo modelo. Para que esses conceitos possam ser utilizados no desenvolvimento de aplicações, eles devem ser representados em alguma linguagem de programação. Essa representação é materializada em um modelo de programação, o qual estabelece a API utilizada no desenvolvimento. Visando simplificar o desenvolvimento e tornar as aplicações mais portáveis, a plataforma DSOA propõe que os componentes sejam desenvolvidos com base em um modelo de programação fundamentado em objetos Java puros, normalmente referenciados como POJO, sendo as dependências dos componentes em termos de serviços tratadas através de um mecanismo de injeção de dependências.

Por fim, considerando-se o papel fundamental do conceito de eventos na plataforma DSOA, espera-se que uma plataforma de suporte ofereça facilidades para a representação e comunicação desses eventos.

7.2.2 Plataforma iPojo

Como vimos na Seção 2.2.4.1, iPojo é uma plataforma de código aberto, fundamentada em um modelo de componentes orientados a serviço baseado em objetos Java puros. Nesse

modelo, os componentes são gerenciados por contêineres customizáveis e extensíveis, os quais foram projetados para serem “leves”, consumindo poucos recursos de memória e processamento.

Apesar da grande flexibilidade decorrente das diferentes possibilidades de configuração, a plataforma iPojo possui importantes limitações no contexto das QSBAs. Como vimos, a própria plataforma assume controle total sobre os serviços providos e requeridos, restando às aplicações a possibilidade de escolha de parâmetros que pareçam adequados. Observa-se ainda, que há um forte viés na plataforma quanto ao tratamento da característica de disponibilidade. Se por um lado, essa abordagem minimiza a possibilidade de erros decorrentes de acesso a serviços não mais disponíveis, por outro, limita a capacidade das aplicações de definir novos critérios de seleção.

De fato, para que se tenha um maior controle dos serviços providos e requeridos, é necessária a implementação de novos tratadores, os quais devem ser utilizados em substituição àqueles utilizados por padrão na plataforma. Contudo, o desenvolvimento de tratadores é delicado, uma vez que esses elementos participam ativamente do gerenciamento das instâncias de componentes, devendo tratar aspectos complexos como validação e invalidação das instâncias e controle de concorrência. Delegar esses aspectos às aplicações vai de encontro ao princípio de separação de interesses.

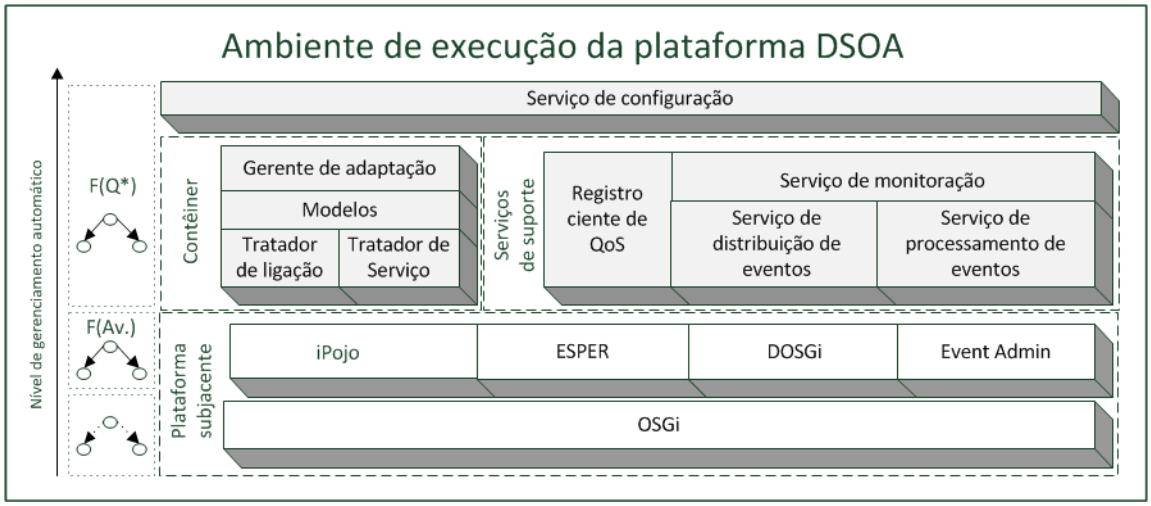
A despeito dessa limitação, a plataforma iPojo atende aos demais critérios relacionados, fornecendo uma plataforma de baixo custo computacional e com um modelo de programação simples e intuitivo. De fato, ao estabelecer um mecanismo capaz de injetar os serviços necessários em uma aplicação, a plataforma iPojo permite que o desenvolvimento seja focado no negócio, de forma que a gerência dos serviços utilizados pela aplicação é realizada sem que ela tenha conhecimento da plataforma.

Em função dessas características, o presente trabalho propõe que a plataforma iPojo seja utilizada como base para a criação da visão de componentes e serviços definida pelos modelos da plataforma DSOA. Nesse contexto, a plataforma DSOA estende a plataforma iPojo, ampliando a capacidade de gerenciamento automático das aplicações, ao permitir que estas adotem outros critérios de qualidade, além da disponibilidade, como gatilho para o processo de adaptação. Ao mesmo tempo, pretende-se manter essas aplicações sob controle da plataforma, evitando que elas manipulem diretamente os serviços e minimizando os riscos inerentes ao dinamismo. Por fim, espera-se que o conjunto de características de qualidade compreendidas pela plataforma resultante seja dinamicamente extensível, e que as aplicações possam definir como essas características devem ser dinamicamente avaliadas.

7.3 AMBIENTE DE EXECUÇÃO DA PLATAFORMA DSOA

A **Figura 33** tem uma dupla finalidade. Por um lado, ela é utilizada para representar o aumento do nível de automatização no gerenciamento das aplicações decorrente da

Figura 33 – Ambiente de execução



Fonte: o autor

utilização da plataforma DSOA. De outro lado, ela apresenta os principais elementos que compõem o ambiente de execução da plataforma.

7.3.1 Gerenciamento Automático

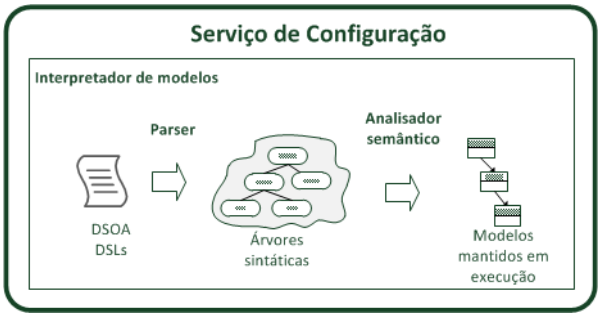
Para representar o aumento do nível de gerenciamento automático, incluímos na **Figura 33** um eixo vertical à esquerda. Esse eixo aponta o aumento nas capacidade de automatização com base nas características de qualidade. Na figura, uma aplicação é representada como um grafo no qual os nós representam seus componentes, os quais são conectados através dos serviços providos e requeridos por eles.

O grafo apresentado no canto inferior esquerdo da figura representa uma aplicação desenvolvida diretamente na plataforma OSGi. Como podemos observar, o grafo não apresenta nenhuma "anotação", indicando que a plataforma OSGi não realiza nenhuma atividade de gerenciamento sobre os elementos que compõem a aplicação. Em outras palavras, não há gerenciamento automático, de forma que eventuais necessidades de adaptação devem ser tratadas pela própria aplicação.

Com a introdução da plataforma iPojo sobre OSGi, a capacidade de gerenciamento automático é incrementada, uma vez que as adaptações relacionadas à disponibilidade dinâmica dos serviços são tratadas pelo contêiner da plataforma. Na **Figura 33**, essa automatização é representado pela "anotação" *F(Av.)*, indicando que o gerenciamento automático realizado pelo contêiner é realizado em função da disponibilidade (i.e., *Availability (Av.)*).

Neste contexto, a proposição da plataforma DSOA promoveu um novo nível de gerenciamento automatizado. De fato, a partir da introdução dessa plataforma é possível automatizar o gerenciamento das aplicações com base em características de qualidade definidas pelas próprias aplicações, oferecendo uma grande flexibilidade. Na **Figura 33**,

Figura 34 – Serviço de configuração



Fonte: o autor

adotou-se a "anotação" *F(Q*)* para indicar que qualquer atributo de qualidade (*Q*) pode ser utilizado no gerenciamento automático de uma aplicação.

7.3.2 Elementos da Plataforma DSOA

Como vimos na Seção 3.1, a construção de um ambiente de execução em uma plataforma de suporte às aplicações auto-adaptativas é uma tarefa complexa, envolvendo desde a concepção de um gerente, responsável pela condução do processo de adaptação, até a definição de um conjunto de serviços de suporte, os quais tem a finalidade de tratar dos diferentes aspectos não-funcionais.

O restante desta seção apresenta os diferentes elementos que foram concebidos para viabilizar a implementação da plataforma DSOA sobre as plataformas iPojo e OSGi.

7.3.2.1 Serviço de Configuração

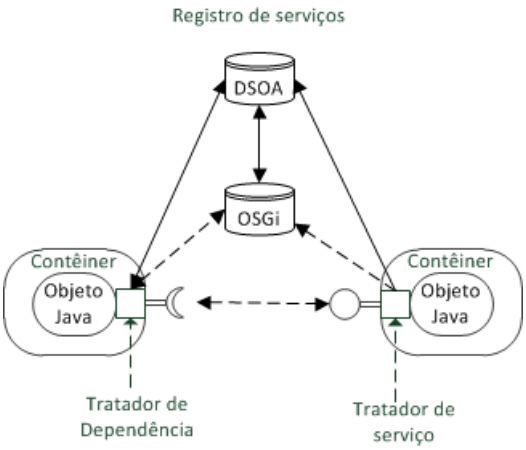
O serviço de configuração, representado na **Figura 34**, é um elemento central na plataforma DSOA. Ele é inicializado junto com o próprio ambiente de execução, sendo responsável por processar os meta-dados definidos em tempo de desenvolvimento, através de modelos específicos de plataforma (i.e., arquivos de configuração) e/ou de anotações, e utilizá-los para (1) instanciar e configurar os elementos que compõem uma aplicação, (2) configurar o gerente de adaptação, e (3) povoar os modelos de tempo de execução, que reificam a aplicação em execução e o gerente de adaptação.

O processo completo de instalação e configuração de aplicações, incluindo a criação dos componentes de execução e das suas meta-representações, será apresentado em detalhes na Seção 7.4.

7.3.2.2 Registro, Tratadores de Ligação e de Serviço

Embora parte das informações necessárias para povoar os modelos de execução tenha origem nos modelos de desenvolvimento, outra parte é inerentemente dinâmica e visa refletir o estado atual da aplicação, devendo ser coletada em tempo de execução.

Figura 35 – Tratadores de serviços e ligação na plataforma DSOA



Fonte: o autor

Uma vez que os componentes e serviços representados nos modelos arquiteturais são conectados aos elementos correspondentes em execução na plataforma subjacente, algumas informações de estado podem ser obtidas desses elementos, como, por exemplo, se o elemento está ou não válido. Contudo, a plataforma iPojo não possui informação acerca da qualidade dos serviços. De fato, embora as aplicações construídas sobre iPojo possam se adaptar de maneira autônoma com relação à disponibilidade dinâmica dos serviços, a plataforma não é capaz de mensurar esse atributo, limitando-se a reagir quando um serviço é publicado, modificado, ou removido do registro.

Como vimos, a plataforma iPojo é construída sobre OSGi e utiliza o registro de serviços fornecido por essa plataforma. Esse registro também não representa o conceito de qualidade, de forma que, embora seja capaz de notificar quando um serviço torna-se disponível e indisponível, não há nenhuma notificação acerca de variações nas demais características de qualidade. De fato, não há na plataforma OSGi/iPojo sensores capazes de produzir informações acerca da qualidade dos serviços. Assim, é necessário estender essa plataforma para introduzir suporte às características de qualidade.

Uma vez que todos os serviços são publicados e descobertos a partir do registro, o primeiro passo para realizar a extensão é controlar os acessos a esse registro. Como mencionado anteriormente, em iPojo, esse acesso é realizado pelos tratadores de dependências (ligação) e de provimento de serviços. Desta forma, a realização da plataforma DSOA envolve a implementação de novos tratadores, os quais devem ser utilizados em substituição aos tratadores padrão.

Os tratadores implementados interagem com o registro de serviços da própria plataforma DSOA, o qual mantém os modelos de qualidade, sendo capaz de "compreender" as características declaradas nesses modelos. Essas características são utilizadas para descrever restrições associadas ao nível de qualidade dos serviços mantidos pelo registro e consultados pelas aplicações. A implementação atual do registro da plataforma publica os

serviços no registro OSGi subjacente, armazenando as restrições de qualidade sob forma de propriedades associadas ao serviço publicado. Esse processo é representado resumidamente na **Figura 35**.

7.3.2.3 Sensores, Serviços de Distribuição e de Processamento de Eventos

Embora os elementos apresentados permitam a descrição dos serviços providos e requeridos por uma aplicação, incluindo a definição das restrições de qualidade correspondentes, eles não são suficientes para a realização da visão da plataforma DSOA. De fato, considerando-se a natureza dinâmica das características de qualidade, é essencial que elas sejam monitoradas em tempo de execução, de forma que os meta-objetos representados nos modelos arquiteturais possam refletir o estado atual da aplicação.

O primeiro passo para realizar essa monitoração é conceber um conjunto de sensores para coletar e disponibilizar dados que possam ser utilizados na computação das métricas de qualidade. Em sua versão atual, a plataforma conta com três sensores, responsáveis por notificar as ocorrências relevantes no ambiente de execução.

Um desses sensores está ligado diretamente ao registro de serviços da plataforma, sendo responsável por notificar a publicação e remoção de serviços, além da modificação nas propriedades que descrevem os serviços já registrados. Um outro sensor é acoplado ao elemento que representa uma ligação entre um componente e um serviço, sendo responsável por notificar sempre que essa ligação se torna válida ou inválida. Por fim, um terceiro sensor é posicionado entre consumidor e provedor de serviço através da utilização de um *proxy*, sendo capaz de notificar os acessos às operações desse serviço. Esse *proxy* será apresentado em detalhes na Seção 7.3.2.4.

Um aspecto importante a ser observado, é que os dados extraídos diretamente dos sensores são, em geral, de baixo nível, devendo ser processados para que possam ser comparados com os objetivos de alto nível relacionados aos requisitos das aplicações. Como discutido, os conceitos do domínio de eventos fornecem uma solução adequada para representar, comunicar, e processar as informações obtidas de sensores, sendo fundamental embutir na plataforma elementos capazes de manipular tais conceitos. Visando suprir essa necessidade, a plataforma incorporou elementos em diferentes níveis.

No nível da plataforma OSGi foram instalados serviços de distribuição e de processamento de eventos já disponíveis e de código aberto. Na implementação atual, o serviço de distribuição utilizado no nível de OSGi é o *Event Admin*, que é um serviço padrão dessa plataforma. Esse serviço funciona como um barramento interno de eventos, sendo capaz de distribuir, de forma síncrona ou assíncrona, os eventos nele publicados. A ideia é que os eventos gerados pelos sensores, referenciados como eventos primitivos, sejam transmitidos por esse barramento.

Para filtrar, analisar, e transformar os eventos gerados pelos sensores, incluímos no

Figura 36 – Serviço de distribuição de eventos

Fonte: o autor

ambiente um motor de processamento (em particular, o ESPER [1]). É dentro deste motor que os agentes de processamento de eventos, representados nos modelos da plataforma, se encontram em execução. No caso do ESPER, cada agente é materializado sob forma de uma consulta em uma linguagem própria, referenciada como *Event Processing Language* (EPL).

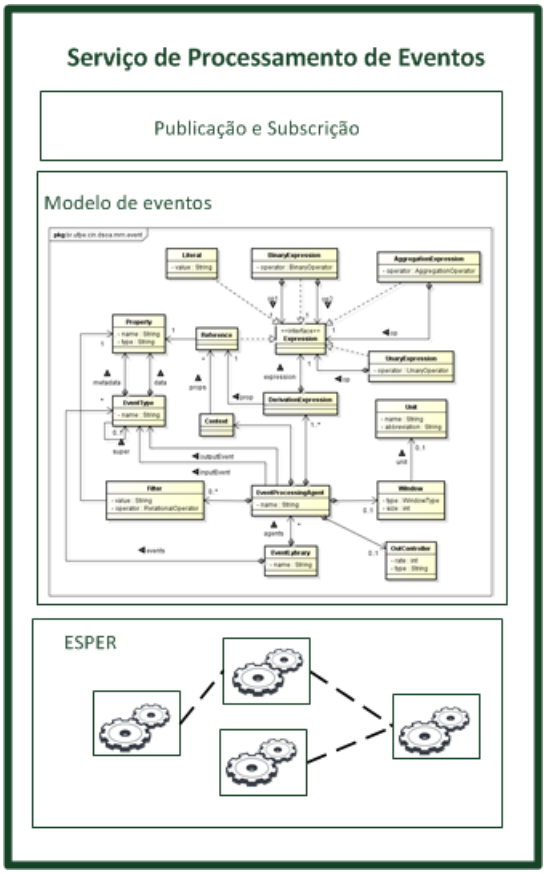
Visando tornar as camadas superiores independentes desses elementos, a própria plataforma define um serviço de distribuição e um serviço de processamento, os quais funcionam como envoltórios, viabilizando a substituição dos serviços de baixo nível por outros capazes de realizar funcionalidade equivalente.

Em particular, o serviço de distribuição de eventos implementado na plataforma, mostrado na **Figura 36**, disponibiliza uma interface própria que permite a publicação de eventos e a criação de subscrições, desvinculando os elementos da plataforma, da API do *Event Admin*. Esse serviço possui, ainda, um conjunto de importadores e exportadores de eventos, os quais visam possibilitar o recebimento de eventos produzidos por sensores externos, assim como a exportação dos eventos produzidos pela plataforma para consumidores externos.

Já o serviço de processamento de eventos da plataforma, representado na **Figura 37**, é responsável por manter os modelos de eventos e criar, para cada agente de processamento representado, a consulta que materializa o agente dentro do motor. Para tanto, o serviço de processamento da plataforma deve transformar a meta-representação do agente descrita no modelo, na consulta a ser mantida no motor. É importante observar que, uma vez que os modelos de eventos são mantidos em execução, se a meta-representação de um agente

[1] http://www.espertech.com/esper/

Figura 37 – Serviço de processamento de eventos



Fonte: o autor

é alterada no modelo, o objeto em execução no motor também é modificado. Para que a consulta que materializa o agente passe a receber os eventos emitidos pelos sensores, o serviço de processamento da plataforma subscreve um consumidor de eventos no serviço de distribuição. Cabe a esse consumidor entregar os eventos ao agente em execução no motor para que eles possam ser processados.

Por fim, para disponibilizar acesso aos eventos de alto nível produzidos pelos agentes, o serviço de processamento implementa uma API, através da qual os demais elementos da plataforma podem se subscrever para receber os eventos complexos.

7.3.2.4 Serviço de Monitoração

Com os agentes posicionados e em execução, cabe ao serviço de monitoração escutar os eventos complexos e traduzir as informações neles contidas para valores de métricas de qualidade, as quais são utilizadas para atualizar os modelos de execução. Para entendermos como isso é realizado, devemos compreender o contexto no qual o serviço de monitoração é normalmente utilizado.

Em geral, esse serviço é acionado por um gerente de adaptação quando este conecta um meta-objeto que representa uma ligação a outro que representa uma instância de

serviço em execução. Neste contexto, o gerente responsável pela ligação aciona o serviço de monitoração, informando ambos os meta-objetos, e solicita que ele mantenha os meta-objetos atualizados com as informações de qualidade.

Para isso, o serviço de monitoração identifica as restrições associadas aos meta-objetos e seleciona as métricas que fazem parte dessas restrições. A cada uma dessas métricas corresponde um tipo de evento complexo produzido por um agente de processamento. Esse tipo de evento é identificado a partir das diretivas de monitoração mantidas pelo próprio serviço de monitoração.

Uma vez identificados os tipos de eventos complexos, o serviço de monitoração registra consumidores de eventos junto ao serviço de processamento. Esses consumidores são responsáveis por ouvir os eventos complexos, extrair deles os valores das métricas de qualidade, e atualizar os meta-objetos.

Quando os valores obtidos para as métricas violam as restrições estabelecidas na ligação, o meta-objeto que a representa é notificado. Neste momento, esse meta-objeto armazena internamente a violação e a comunica ao gerente, que deve decidir se a ligação com a instância de serviço deve ou não ser mantida.

Para que todo esse fluxo possa funcionar, é necessário que os agentes recebam os eventos primitivos utilizados como entrada. Para que esses eventos primitivos sejam gerados e encaminhados, o serviço de monitoração utiliza *proxies*.

Como vimos, quando o serviço de monitoração é acionado para monitorar uma ligação, ele recebe os meta-objetos que representam a ligação e a instância de serviço selecionada. Nesse momento, o serviço de monitoração gera um *proxy* para a instância de serviço. É este *proxy*, contendo uma referência para instância de serviço, que é efetivamente conectado ao meta-objeto que representa a ligação. Na prática, os acessos às operações da instância passam pelo *proxy*, que deve notificar essa invocação.

Neste ponto, cabe explicar como a invocação de um serviço no plano base chega até o *proxy*. Para compreendermos esse mecanismo, é necessário entendermos como funcionam os contêineres de componentes na plataforma iPojo.

A plataforma iPojo possui um modelo de programação no qual os componentes são definidos através de classes Java "puras", nas quais não aparece nenhum código referente ao modelo de componentes. Um exemplo de código de aplicação que requer um serviço é apresentado na **Figura 38**. Como podemos observar, o serviço é utilizado diretamente, sem ser descoberto pela própria aplicação. De fato, cabe ao contêiner injetar o serviço selecionado no atributo correspondente da classe, não sendo necessário que o desenvolvedor da aplicação tenha ciência acerca do registro de serviços.

Ainda durante o desenvolvimento, a classe que realiza o código de negócio passa por um processo de manipulação, utilizando-se uma ferramenta da plataforma. Essa ferramenta modifica os *bytecodes* da classe, injetando neles código próprio da plataforma iPojo. Uma parte do código manipulado é apresentado na **Figura 39**.

Figura 38 – Código de aplicação utilizando serviço

```
package br.ufpe.cin.exemplos;

public class Aplicacao {
    private Servico servico;

    public void usarServico() {
        this.servico.acionarOperacao();
    }
}
```

Fonte: o autor

Figura 39 – Código manipulado

```
1  public class Aplicacao implements Pojo
2  {
3    private InstanceManager __IM;
4    private boolean __Fservico;
5    private Servico servico;
6    private boolean __MusarServico;
7
8    Servico __getservico()
9    {
10     if (!this.__Fservico)
11       return this.servico;
12     return (Servico)this.__IM.onGet(this, "servico");
13   }
14
15   public void usarServico() {
16     try {
17         this.__IM.onEntry(this, "usarServico", new Object[0]);
18         __usarServico();
19         this.__IM.onExit(this, "usarServico", null);
20     } catch (Throwable localThrowable) {
21         this.__IM.onError(this, "usarServico", localThrowable);
22         throw localThrowable;
23     }
24   }
25   private void __usarServico() {
26     __getservico().acionarOperacao();
27   }
```

Fonte: o autor

Como podemos observar, o método `usarServico` original foi modificado. Esse método agora se inicia com uma chamada ao contêiner (`InstanceManager`) para informar a ele que o método de negócio está sendo iniciado (linha 17). Após essa chamada, ocorre uma chamada a um novo método criado no processo de manipulação, o método `__usarServico`. Ao final do método, em caso de sucesso ou de erro, o contêiner é novamente acionado (linhas 19 e 21).

O novo método criado corresponde ao método de negócio original, sendo que o serviço é requisitado ao contêiner através da chamada do método `onGet` (linha 12). Assim, cabe ao contêiner retornar o serviço a ser utilizado. Internamente, esse contêiner mantém uma lista de objetos que possuem interesse em indicar o serviço que deve ser utilizado. Na plataforma iPojo original, o tratador de dependências cria, para cada dependência de

serviço, um objeto que deve responder qual serviço que deve ser utilizado. Esse objeto é registrado junto ao contêiner e acionado quando o serviço é requisitado.

Na plataforma DSOA, substituímos o tratador de dependências por um tratador de ligação próprio. Esse tratador registra o meta-objeto que representa uma ligação junto ao contêiner, fazendo com que o serviço a ser utilizado seja requisitado ao modelo de execução. Neste ponto, o meta-objeto que representa a ligação retorna o *proxy*, que foi injetado nele pelo serviço de monitoração.

Quando uma operação é requisitada no *proxy*, este a encaminha à instância de serviço correspondente, e gera um evento primitivo de invocação que é encaminhado para processamento via serviço de distribuição. Como mencionamos anteriormente, consumidores ligados ao serviço de processamento de eventos recebem esse evento primitivo e o encaminham para os agentes em execução no motor de processamento. Por fim, eventos complexos computados a partir dos eventos primitivos são produzidos pelos agentes e encaminhados para o serviço de monitoração. De forma resumida, é assim que o nível meta da arquitetura reflexiva da plataforma DSOA conduz o processo de adaptação.

Apresentados os elementos estruturais que compõem a plataforma, passamos a discutir como as aplicações são instaladas e configuradas para que possam entrar em execução.

7.4 INSTALAÇÃO E CONFIGURAÇÃO

Para ser instalada na plataforma DSOA, uma aplicação deve ser empacotada. Cada pacote instalável contém, além do código da aplicação, modelos específicos de desenvolvimento, representando os conceitos relacionados à qualidade, eventos, serviços, e monitoração. Esses pacotes são processados pelo serviço de configuração, que realiza diversas atividades como descrito na sequência.

7.4.1 Configuração de Qualidade

As características de qualidade são representadas por um modelo simples, que foi definido com o intuito de permitir que cada aplicação especifique seus próprios atributos e métricas. Para que essas características sejam transferidas para o ambiente de execução, o modelo de qualidade é incluído no pacote de instalação da aplicação com um nome e localização pré-estabelecidos.

Quando uma aplicação é instalada na plataforma, o serviço de configuração é capaz de recuperar e interpretar o modelo, criando instâncias das classes que representam os conceitos de qualidade. Em particular, essas instâncias são fundamentalmente estruturas de dados organizadas para compor uma taxonomia de qualidade, refletindo as propriedades descritas no modelo de qualidade.

Por fim, o serviço de configuração armazena esses objetos no registro de serviços ciente de qualidade, ampliando o conjunto de características de qualidade conhecidas pela

plataforma. Essas características são verificadas quando da publicação e descoberta dos serviços. Assim, não é possível publicar um serviço contendo uma restrição associada a uma métrica de qualidade desconhecida. Mais ainda, ao tentar descobrir os serviços candidatos especificando restrições associadas a uma métrica desconhecida, o registro não retornará nenhum candidato.

7.4.2 Configuração dos Eventos e Agentes de Processamento

Para dar significado às métricas de qualidade, uma aplicação utiliza um modelo representando eventos e agentes de processamento. De forma similar ao modelo de qualidade, o modelo de eventos também é inserido no pacote de instalação e interpretado pelo serviço de configuração.

Durante o processo de interpretação, o serviço de configuração cria instâncias das meta-classes que representam os tipos de eventos e os agentes declarados no modelo. Essas meta-classes correspondem diretamente àquelas representadas no meta-modelo e discutidas em detalhes no capítulo anterior. As instâncias criadas e povoadas pelo serviço de configuração são registradas no serviço de processamento de eventos da plataforma, o qual é responsável por manter um catálogo dos eventos e agentes conhecidos e traduzir esses elementos em conceitos similares presentes no motor de processamento ESPER.

Uma vez instanciados e registrados no serviço de processamento de eventos, os agentes definidos nos modelos entram em execução, processando os eventos declarados em seu terminal de entrada e transformando esses em eventos do tipo declarado na definição do terminal de saída. Um evento produzido por um agente pode ser utilizado como entrada por outro agente. Dessa forma, o conjunto de agentes de processamento em execução na plataforma compõe uma rede de processamento de eventos.

É importante mencionar que novos tipos de eventos e agentes podem ser incluídos em tempo de execução. Mais ainda, os meta-objetos que representam os agentes em execução na plataforma podem ser dinamicamente modificados, promovendo uma alteração na forma como os eventos são processados. Em particular, é possível modificar tipo e tamanho da janela utilizada pelo agente de processamento, assim como a taxa de saída desse agente (vide capítulo anterior). Para viabilizar essas modificações, a plataforma expõe os agentes de processamento através de *Java Management Extensions* (JMX) e de um conjunto de comandos próprios da plataforma, os quais foram implementados sobre o interpretador de comandos de OSGi.

7.4.3 Configuração das Diretivas de Monitoração

Definidas as métricas de qualidade, os tipos de eventos, e os agentes de processamento, é necessário indicar para a plataforma como cada métrica pode ser mensurada. Neste contexto, são utilizadas diretivas, as quais configuram o serviço de monitoração da plataforma

de forma que este saiba qual tipo de evento contém informação sobre uma determinada métrica.

Como vimos na seção anterior, quando o serviço de monitoração é acionado para supervisionar um serviço ou uma ligação, ele instrumenta esse elemento, criando um *proxy* que funciona como um sensor, sendo responsável por gerar eventos primitivos representando as invocações de operações. Os eventos primitivos gerados são encaminhados através do serviço de distribuição para serem processados pelos agentes, os quais, como vimos, computam efetivamente as métricas de qualidade.

7.4.4 Configuração e Reconfiguração da Arquitetura

Os modelos arquiteturais de execução representam os elementos estruturais que compõem uma aplicação através de meta-objetos posicionados em dois níveis ontológicos distintos: (1) nível de tipos, reificando os tipos de componentes e serviços utilizados em uma aplicação, e (2) nível de instâncias, reificando as instâncias de componentes que formam, efetivamente, as aplicações em execução. Para entendermos como esses elementos são criados, é necessário compreendermos como os modelos arquiteturais de desenvolvimento são utilizados para criar os componentes no plano base.

7.4.4.1 Componentes e Serviços

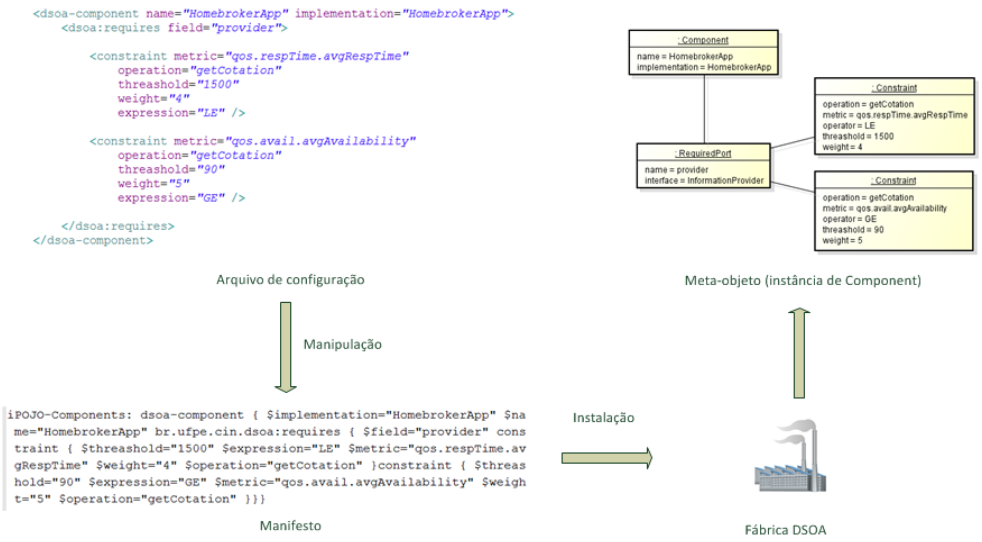
Na plataforma DSOA, os elementos que compõem uma aplicação são declarados em um arquivo de configuração, o qual pode ser visto como um modelo arquitetural de desenvolvimento. Ao processar esse modelo, o serviço de configuração identifica os tipos de componentes utilizados pela aplicação. Em tempo de execução, deve existir uma instância de fábrica associada a cada um desses tipos. Essa instância de fábrica possui a responsabilidade de criar, quando necessário, as instâncias do componente.

Para que os componentes da plataforma DSOA pudessem ser adequadamente construídos, estendemos a plataforma iPojo, através da definição de um novo tipo de fábrica de componentes, a fábrica de componentes DSOA. Para denotar que um tipo de componente deve ser tratado por esse tipo de fábrica, definimos um meta-dado próprio através da criação de uma nova anotação (`dsoa-component`).

Dessa forma, sempre que um componente anotado com a marcação `dsoa-component` é declarado em um modelo arquitetural de desenvolvimento, uma instância da fábrica DSOA é criada, configurada, e registrada. Durante o processo de configuração, a instância da fábrica processa os meta-dados definidos no modelo e cria um meta-objeto do tipo *Component* (descrito no capítulo anterior), representando o tipo de componente. Uma única instância desse meta-objeto é criada e mantida junto à fábrica.

O processo de configuração de uma fábrica envolve ainda a identificação dos tratadores que devem compor as instâncias do componente. Para tanto, são verificados os meta-

Figura 40 – Criação de componente DSOA



Fonte: o autor

dados especificados na declaração do tipo de componente. Cada tratador é responsável por processar seus meta-dados, podendo participar da definição do tipo de componente.

Assim, o tratador de serviços providos e o tratador de ligações devem processar as declarações dos serviços providos e requeridos. Cada uma dessas declarações compreende, além da interface funcional, um conjunto de restrições associadas às operações desse serviço, especificando o nível de qualidade esperado das mesmas. A partir dessas definições, os tratadores completam o meta-objeto *Component*, incluindo as portas. O processo de criação de um novo tipo de componente é representado na **Figura 40**.

7.4.4.2 Instâncias de Componente

Para gerenciar as instâncias de componente com base nas características de qualidade, a plataforma DSOA definiu seu próprio contêiner. Desta forma, quando uma fábrica DSOA recebe uma solicitação para criar uma instância de componente, ela instancia o contêiner da plataforma, e este cria um meta-objeto, do tipo *ComponentInstance*, responsável por reificar essa instância no modelo arquitetural de execução. Uma vez criada a instância, a fábrica deve realizar a sua configuração e inicialização.

Para configurar uma instância, cada tratador indicado na definição do tipo de componente é acionado, tendo a possibilidade de participar do processo de configuração. No caso da plataforma DSOA, os tratadores de ligação e de serviços completam a representação da instância, inserindo os *Binding*s e *ServiceInstance*s no meta-objeto que representa a instância em execução.

Uma vez criada e configurada, a fábrica deve inicializar a instância de componente. Novamente, os tratadores são envolvidos durante esse processo. Na plataforma DSOA, os tratadores de dependência e de serviço avisam aos elementos do modelo (meta-objetos)

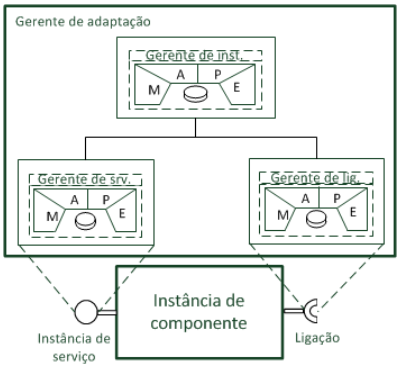


Figura 41 – Instância de componente DSOA

que eles devem ser inicializados. Por sua vez, os meta-objetos informam aos respectivos gerentes de adaptação sobre a inicialização através de uma interface de notificação. Como mostrado na **Figura 41**, cada um desses meta-objetos está associado a um gerente, de forma que o gerenciamento da instância é compartilhado.

No caso de uma ligação (*Binding*), o gerente correspondente (*BindingManager*), consulta o registro de serviços da plataforma informando a interface do serviço e as restrições de qualidade. O registro retorna um conjunto de meta-objetos (*ServiceInstance*s), representando os serviços candidatos. Cada uma deles contém uma referência para o serviço real.

Os candidatos retornados pelo registro são apresentados a uma política de seleção, a qual deve indicar o serviço que deve ser conectado à ligação. Na plataforma DSOA, essa política pode ser especificada pela aplicação através da definição de uma classe que implementa uma interface definida pela plataforma. Caso a aplicação não especifique uma política, a plataforma utiliza uma política de seleção padrão, a qual se baseia em Ponderação Simples Aditiva (do inglês, *Simple Additive Weighting* (SAW)), que é um método baseado em pesos para a tomada de decisões envolvendo múltiplos critérios. Desta forma, a política padrão constrói uma matriz de decisão a partir dos valores das métricas de qualidade dos serviços monitorados e dos pesos indicados junto às restrições de qualidade definidas nos modelos arquiteturais.

Com a instância de serviço selecionada, o gerente responsável pela ligação solicita ao serviço de monitoração que a mesma seja supervisionada. Como vimos, o serviço de monitoração instrumenta a ligação, ligando-a um *proxy* que intermedeia o acesso ao serviço e notifica as invocações através de eventos do tipo *InvocationEvent*. Cada um desses eventos contém o serviço acionado, a operação requisitada, etiquetas de tempo correspondentes à data/hora de requisição e retorno, e informações referentes a uma eventual exceção. Por fim, com a ligação conectada à instância de serviço (via *proxy*), esta torna-se válida. Do ponto de vista do tratador de provimento de serviço, nenhuma ação é necessária nesse momento.

Após todos os tratadores terem finalizado o processo de inicialização, o contêiner so-

licita que cada tratador verifique se ele está válido. Na plataforma DSOA, cada tratador afere sua validade analisando os elementos do modelo arquitetural de execução. Em particular, o tratador de dependências pergunta a cada ligação se ela está válida. Caso todas estejam, o tratador fica válido. Quanto ao tratador responsável pelos serviços providos, ele está sempre válido, sendo que o serviço correspondente só será publicado quando a instância como um todo estiver válida. Essa instância estará válida se todos os tratadores estiverem válidos, sendo essa validação refletida no meta-objeto que representa a instância do componente em execução.

7.4.4.3 Reconfiguração

Durante a execução de uma aplicação, os elementos do modelo arquitetural são atualizados pelo serviço de monitoração. Caso algum dos serviços conectados às ligações viole as restrições estabelecidas no modelo, o gerente responsável pela ligação é notificado através de uma interface de *callback*. Nesse momento inicia-se uma avaliação que pode identificar a necessidade de reconfiguração.

Para avaliar se há essa necessidade, o gerente aciona uma política de avaliação. De forma similar à política de seleção, uma aplicação pode definir uma política de avaliação implementando uma interface definida pela plataforma (vide Capítulo 6). A avaliação realizada pela política é representada através de uma operação que retorna um valor lógico. Um retorno verdadeiro indica que uma reconfiguração é necessária, cabendo ao gerente obter os serviços candidatos e encaminhá-los para a política de seleção. Como vimos na subseção anterior, uma vez selecionado um serviço, o gerente solicita que ele seja monitorado.

Por fim, cabe ressaltar que, caso uma aplicação não indique expressamente a política de avaliação a ser utilizada, a plataforma utiliza uma política padrão, a qual assume que uma violação implica, necessariamente, em uma reconfiguração.

7.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou como a plataforma DSOA foi implementada. Visando suportar a adaptação com base nos modelos apresentados anteriormente, partimos de um modelo de componentes orientados a serviço de código aberto e incluímos elementos para representar os conceitos de qualidade e eventos, os quais são explicitamente representados nos modelos da plataforma.

Ao longo do capítulo, apresentamos, detalhadamente, os elementos estruturais que compõem a plataforma e a relação deles com os modelos mantidos em execução. Por fim, discutimos como uma aplicação é instalada, configurada, e executada na plataforma.

8 EXPERIMENTO E RESULTADOS

> *Todo o conhecimento humano começou com intuições, passou daí aos conceitos e terminou com ideias.*
>
> Immanuel Kant,

Este capítulo apresenta uma avaliação da plataforma DSOA através da realização de um conjunto de experimentos. Pretende-se mostrar que a plataforma suporta a execução de aplicações capazes de se adaptar dinamicamente em função de qualidade dos serviços por ela consumidos. Conforme vimos anteriormente, a forma de computar as métricas de qualidade não é pré-definida na plataforma, permitindo que os desenvolvedores definam como esse processo deve ser realizado em função dos eventos que ocorrem em tempo de execução. Neste contexto, pretende-se mostrar que a forma de computação das métricas de qualidade impacta o processo de adaptação, tendo reflexo direto no desempenho da aplicação. Por fim, pretende-se avaliar o impacto observado na perspectiva do cliente decorrente da monitoração das métricas de qualidade introduzida pela plataforma.

8.1 APLICAÇÃO *HOMEBROKER*

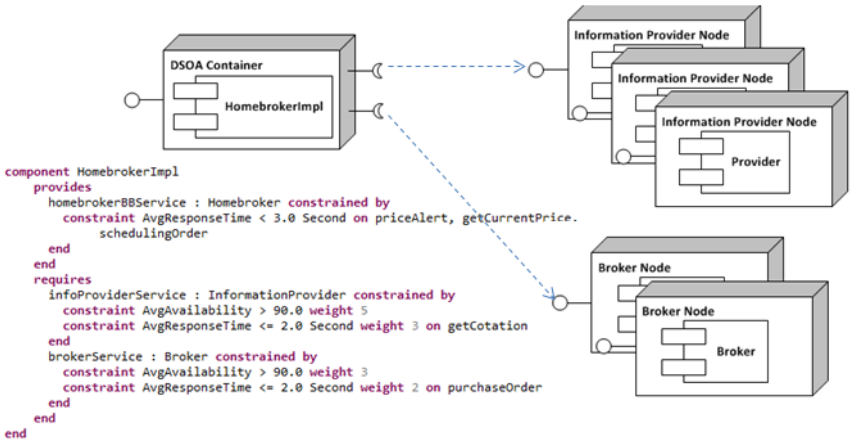
Para realizar os experimentos necessários para a avaliação da plataforma DSOA, é necessário escolhermos um tipo aplicação na qual a qualidade dos serviços seja determinante para o sucesso do negócio. Neste contexto, selecionamos como estudo de caso uma aplicação do "tipo *homebroker*".

Essas aplicações são utilizadas para operar no mercado de ações. De uma forma geral, elas obtém as cotações das ações a partir de serviços disponibilizados por agências especializadas. Os valores obtidos para essas cotações são frequentemente utilizados para apoiar decisões acerca da realização de operações de compra e venda de ações, as quais podem ser realizadas de forma automática ou manual.

No mercado de ações, as informações são extremamente sensíveis aos aspectos temporais, sendo essencial que as informações obtidas representem o estado atual do mercado. Para isso, as aplicações de *homebroker* devem utilizar servidores com alta disponibilidade e com um baixo tempo de resposta. De fato, eventuais problemas de conexão ou de atrasos podem gerar grandes prejuízos financeiros. Nesse contexto, a capacidade de identificar rapidamente falhas e promover adaptações no sentido de substituir um serviço com desempenho inadequado é uma característica essencial. A natureza desse tipo de aplicação é interessante no contexto da plataforma DSOA, a qual foi especialmente projetada para suportar adaptações dinâmicas em função das características de qualidade.

A nossa aplicação exemplo consome dois serviços representados pelas interfaces *InformationProvider* e *Broker*. A aplicação e os serviços requeridos são representados na

Figura 42 – Aplicação Homebroker



Fonte: o autor

Figura 42. Como podemos verificar, a aplicação *homebroker* executa em um contêiner DSOA, no qual está embutido o gerente responsável pela condução do processo de adaptação da aplicação. Quando a aplicação é instalada, cabe a esse gerente descobrir e selecionar os serviços que devem ser injetados na aplicação.

Como vimos, a plataforma DSOA permite que uma aplicação participe do processo de seleção de serviços, implementando uma política de seleção que é acionada pelo gerente uma vez que os serviços candidatos são descobertos no registro. Nos experimentos apresentados nesta tese, não utilizamos essa flexibilidade (que foi explorada em outras publicações [(CAVALCANTI; SOUZA; ROSA, 2013)][(SOUZA et al., 2014)]), deixando que o próprio gerente utilize a política de seleção padrão da plataforma. No capítulo anterior, mencionamos que essa política é baseada na técnica denominada *Ponderação Simples Aditiva*. Os pesos utilizados no contexto dos experimentos são apresentados na **Figura 42**.

Durante a execução da aplicação, o gerente DSOA monitora o nível de qualidade dos serviços utilizados, visando garantir que os requisitos estabelecidos pela aplicação sejam satisfeitos. Como pode ser observado na **Figura 42**, a aplicação requer serviços que realizem suas operações em um tempo médio de 2 segundos e que tenham uma disponibilidade média de pelo menos 90 porcento. Para que essas métricas possam ser monitoradas, a aplicação define modelos especificando, de forma declarativa, dois agentes de processamento, um responsável por monitorar o tempo médio, outro por monitorar a disponibilidade média. Esses modelos são utilizados pelo gerente para instanciar os agentes em tempo de execução.

Caso os limites estabelecidos sejam violados, o gerente notifica a aplicação, que pode decidir acerca da necessidade de substituir o serviço implementando uma política de avaliação. Uma vez que o intuito dos experimentos é verificar a adaptação dinâmica, a política de avaliação implementada define que um serviço deve ser substituído sempre que não atender às restrições estabelecidas. Sendo indicada a necessidade de substituição,

cabe ao gerente a responsabilidade por selecionar um novo serviço e injetá-lo na aplicação.

Um aspecto importante a ser destacado diz respeito à estratégia de monitoração adotada nos experimentos. Como vimos no capítulo anterior, a monitoração na plataforma é realizada através da interposição de *proxies* entre a aplicação e os serviços consumidos. Esses *proxies* atuam como sensores, sendo responsáveis pela emissão de eventos de invocação, os quais são utilizados pelos agentes de processamento declarados pelas aplicações com a finalidade de computar as métricas de qualidade. Sem esses eventos as métricas de qualidade associadas às interações entre aplicação e serviços não podem ser computadas. Em outras palavras, se um serviço não está sendo utilizado, a plataforma não consegue coletar métricas como tempo médio de resposta. Em particular, a ausência de informações acerca de algumas métricas pode levar o gerente a selecionar serviços que não estão, no momento, apresentando valores interessantes para essas métricas. Em geral, isso não representa um problema, pois se esses serviços não se comportarem dentro das restrições especificadas, eles serão substituídos.

Contudo, a plataforma possui uma opção de configuração que pode ser utilizada nos cenários onde essa característica representa, efetivamente, um problema. Em particular, essa configuração permite indicar que a plataforma deve realizar uma monitoração ativa que consiste na utilização de uma rotina disparada por um temporizador configurável, o qual fica fazendo solicitações para os serviços que podem ser utilizados por uma aplicação. Esses serviços candidatos são identificados a partir da sua interface funcional e das características de qualidade publicadas por eles no registro.

Uma vez que o intuito dos dois primeiros experimentos apresentados nesse capítulo é avaliar a plataforma do ponto de vista das capacidades de monitoração e adaptação, optamos pela utilização da monitoração ativa, permitindo que o gerente recebesse informações atualizadas acerca do estado de cada serviço no que diz respeito às métricas de qualidade utilizadas pela aplicação. Por fim, no terceiro experimento, avaliamos o impacto no desempenho observado pelos clientes quando a monitoração passiva é utilizada.

8.2 EXPERIMENTOS E RESULTADOS

8.2.1 Primeiro Experimento

8.2.1.1 Objetivo

O primeiro experimento realizado visa avaliar a plataforma DSOA segundo uma perspectiva puramente funcional. Em essência, o objetivo é verificar se de fato a plataforma é capaz de utilizar os modelos de agentes, definidos pelas aplicações, para instanciar agentes de processamento em tempo de execução, de forma que esses agentes possam computar as métricas de qualidade com base nos eventos disparados durante a execução da aplicação. Adicionalmente, pretende-se observar se a plataforma é capaz de utilizar as métricas

computadas pelos agentes para realizar adaptações na aplicação em execução, através da substituição de serviços que não atendam às características de qualidade definidas.

8.2.1.2 Configuração

Para fins de computação das métricas, a aplicação *homebroker* inclui a definição de dois agentes: (1) *AvgAvailabityAgent*, o qual computa a disponibilidade média do serviço, e (2) *AvgResponseTimeAgent*, o qual computa o tempo médio de resposta. Como o objetivo do primeiro experimento é mostrar que uma aplicação é capaz de determinar como as métricas de qualidade devem ser computadas, definimos, enquanto desenvolvedores da aplicação *homebroker*, que ambos os agentes agrupam os eventos em janelas com tamanho fixo de 100 eventos. Essa definição é feita nos modelos da aplicação que especificam esses agentes.

Uma vez que o experimento não tem por objetivo avaliar o desempenho da plataforma DSOA, o cenário do experimento envolve um único usuário virtual fazendo seguidas requisições à aplicação para obter a cotação de ações. Após cada requisição, o usuário dá uma pausa de 1 segundo antes de enviar a requisição seguinte. Ao receber a requisição, a aplicação a encaminha para um provedor de informações que é selecionado pelo contêiner da plataforma utilizando o registro de serviços ciente de qualidade. Os serviços candidatos são classificados com base na descrição das características não-funcionais informadas no registro e em pesos especificados pelos desenvolvedores da aplicação.

Uma vez que os atributos de qualidade são inerentemente dinâmicos, a qualidade dos serviços ofertados pelos provedores muda ao longo da execução. Para impor algum controle sobre o cenário proposto, desenvolvemos um simulador que pode ser configurado para representar serviços fornecendo características de qualidade estabelecidas. No nosso primeiro experimento, configuramos esse simulador para criar três provedores de informação distintos (*S01*, *S02*, e *S03*), os quais possuem suas características de qualidade variando ao longo do tempo. Inicialmente, *S01* foi configurado para ser o serviço mais rápido, sendo, portanto, escolhido pelo contêiner e injetado na aplicação. Depois de algum tempo, o serviço *S01* se torna indisponível. Ao perceber esse fato, o contêiner substitui o serviço S01 pelo serviço *S02*, o qual é mais lento que *S01* era inicialmente, mas está disponível e atende às restrições de qualidade impostas pela aplicação. Eventualmente, o tempo de resposta computado pelo contêiner se torna maior do que aquele que é estabelecido nas restrições de qualidade, levando o contêiner a substituir o *S02* pelo *S03*. No nosso cenário, *S03* era inicialmente o serviço mais lento, contudo o tempo de resposta dele é estável, e ele está sempre disponível.

8.2.1.3 Resultados

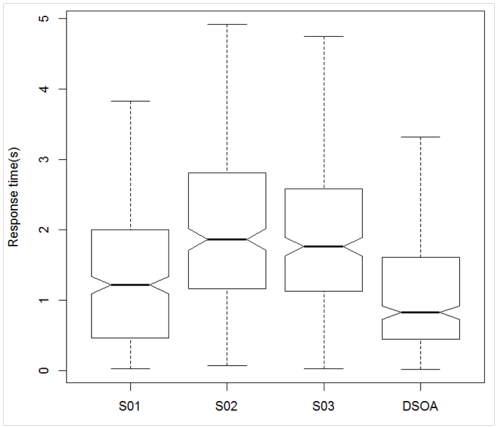
Os resultados do primeiro experimento realizado são resumidos na **Tabela 2** e apresentados graficamente na **Figura 43** através de um *box plot*, que representa o tempo de resposta de cada serviço provedor de informação e da aplicação executando na plataforma. A partir

Tabela 2 – Sumário estatístico (em segundos)

	Média	Mínimo	1o Quartil	Mediana	3o Quartil	Máixmo
S01	1,426	0,028	0,467	1,218	2,000	3,827
S02	2,173	0,080	1,168	1,870	2,808	4.918
S03	2.022	0,031	1,132	1,764	2,588	4.748
DSOA	1,201	0,028	0,447	0,830	1,611	3,325

Fonte: o autor

Figura 43 – Tempo de resposta em segundos



Fonte: o autor

do experimento foi possível não somente verificar que o contêiner é capaz de computar as métricas de qualidade necessárias, mas também que ele é capaz de substituir os serviços, de forma a tentar garantir os requisitos não-funcionais estabelecidos.

Como podemos verificar, o tempo de resposta observado com a aplicação em execução no contêiner DSOA apresenta os menores valores para média e mediana. Mais ainda, podemos observar que a diferença entre o primeiro e o terceiro quartis da aplicação executando na plataforma (representada pela área da caixa) também é a menor, comprovando que o contêiner pode controlar o tempo de resposta da aplicação através da substituição dos serviços em utilização quando esses não atendem aos critérios previamente estabelecidos. É importante verificar que os valores que ficam fora dos bigodes (*whiskers*) são considerados *outliers*, não sendo objeto de avaliação.

Ainda na **Figura 43**, podemos observar que não há sobreposição entre os entalhes dos serviços e da aplicação na plataforma, mostrando que há uma evidência forte de que suas medianas são diferentes (CHAMBERS et al., 1983). Em outras palavras, isso significa que a mediana do tempo de resposta da aplicação é menor do que a de qualquer serviço individual, ratificando a importância da capacidade de reconfiguração dinâmica disponibilizada pela plataforma. Do ponto de vista da disponibilidade, o único serviço que foi "programado" para se tornar indisponível foi o serviço *S01*. Como mencionado, essa

indisponibilidade levou o contêiner a substituir *S01* por *S02* na composição.

A despeito da relevância dos resultados apresentados, os quais demonstram a capacidade de adaptação dinâmica das aplicações em execução na plataforma DSOA, há alguns aspectos importantes que devem ser mencionados. Em primeiro lugar, deve-se destacar que *não* é objetivo da plataforma escolher sempre o melhor serviço em atividade. De fato, embora os tempos observados no primeiro experimento mostrem que a aplicação executando na plataforma conseguiu ter um tempo de resposta médio melhor do que se utilizasse um único serviço durante todo o experimento, esse fato decorreu do cenário simulado, no qual as características dos serviços foram piores em momentos distintos, de forma que a aplicação foi capaz de identificar e utilizar os "melhores momentos" de cada um. Como veremos no experimento seguinte, esse não é sempre o caso.

Em particular, a plataforma não tem a pretensão de selecionar o melhor serviço a cada requisição. Como discutido por [(DUSTDAR et al., 2010)], esse tipo de seleção envolve um custo relativamente alto a cada requisição, uma vez que requer a execução da política de seleção sempre que uma requisição precisa ser enviada para um serviço. Desta forma, a plataforma DSOA optou por não adotar essa estratégia. Na plataforma DSOA, um serviço é selecionado para utilização por uma aplicação e permanece sendo utilizado por ela enquanto estiver atendendo às restrições estabelecidas, não sendo realizada uma seleção a cada requisição. Somente quando há uma violação, a aplicação é interpelada quanto à necessidade de substituição. Ainda assim, a aplicação pode optar por continuar utilizando o mesmo serviço, por exemplo, em função do histórico observado. Acreditamos que essa solução é mais natural, uma vez que, se um serviço está atendendo os requisitos, não há, em princípio, uma necessidade de substituição. Naturalmente, a identificação de cenários nos quais essa característica seja necessária pode motivar futuras extensões.

8.2.2 Segundo Experimento

8.2.2.1 Objetivo

O segundo experimento projetado visa mostrar que a habilidade de definir como as métricas de qualidade são computadas representa um aspecto importante no desenvolvimento das QSBAs. Uma vez que diferentes formas de computação de métrica geram diferentes valores para as mesmas, e que esses valores são o principal gatilho para a adaptação das aplicações em execução na plataforma, a escolha da forma de computar a métrica (realizada através da definição declarativa de um agente de processamento) tem um impacto direto no processo de adaptação, definindo os momentos em que essa adaptação deve ser efetivamente realizada.

Tabela 3 – Serviços simulados com distribuição uniforme

	Serviços candidatos		
Intervalo	S01 (violeta)	S02 (laranja)	S03 (verde)
0 a 3 minutos	Mínimo: 450 ms Máximo: 900 ms	Mínimo: 1200 ms Máximo: 1500 ms	Mínimo: 700 ms Máximo: 1100 ms
3 a 6 minutos	Mínimo: 1100 ms Máximo: 1700 ms	Mínimo: 800 ms Máximo: 1000 ms	Mínimo: 1300 ms Máximo: 1500 ms
6 a 9 minutos	Mínimo: 600 ms Máximo: 950 ms	Mínimo: 900 ms Máximo: 1100 ms	Mínimo: 1000 ms Máximo: 1600 ms
9 a 12 minutos	Mínimo: 450 ms Máximo: 900 ms	Mínimo: 1200 ms Máximo: 1500 ms	Mínimo: 700 ms Máximo: 1100 ms

Fonte: o autor

8.2.2.2 Configuração

Nesse experimento, configuramos o simulador para representar três provedores de informação com características distintas e variando ao longo do tempo, denominados *S1*, *S2*, e *S3*. Com o intuito de possibilitar uma representação adequada do tempo de resposta dos serviços e da aplicação consumidora em execução na plataforma através de um gráfico que exibisse o decorrer do tempo do experimento, optamos por configurar o simulador de tal forma que os tempos de resposta observados para os serviços candidatos seguisse uma distribuição uniforme. É importante observar que experimentos semelhantes foram realizados com distribuição exponencial, mas a grande variação de valores torna o gráfico que representa os tempos de resposta obtidos durante o experimento pouco legível, sendo difícil acompanhar as adaptações. De fato, nos cenários envolvendo distribuições exponenciais, o uso de *boxplot* nos pareceu mais adequado.

O comportamento dos serviços utilizados no segundo experimento é resumido na **Tabela 3**. Na definição de cada serviço consta a cor da linha que será utilizada para representar o serviço graficamente mais adiante. Para que os resultados do experimento pudessem ser representados graficamente, optamos por apresentar apenas doze minutos de experimento, divididos em quatro intervalos de três minutos cada.

Na execução do experimento, a aplicação definiu, via arquivo de configuração (visto como um modelo específico de plataforma), a utilização da métrica de qualidade denominada *tempo médio de resposta*. É importante ressaltar que a definição do que isso significa não está implícita na plataforma, que desconhece o conceito dessa métrica.

Para dar significado à métrica, a aplicação definiu um agente de processamento para os eventos de invocação e configurou esse agente de forma que ele computasse a média da diferença entre os tempos de resposta e de requisição de cada invocação de serviço. Para mostrar a importância de deixar a aplicação definir como a métrica deve ser computada,

a mesma aplicação foi executada com duas configurações de agentes distintas.

Na primeira configuração do agente, a aplicação definiu que deveria ser considerada uma janela de 25 eventos no cálculo da média. Mais ainda, definiu que a saída do agente deveria ser produzida a cada 10 requisições processadas. Assim, a cada 10 requisições, o agente computa a média do tempo das últimas 25. Ressalto que esses valores de parâmetros são definidos pela aplicação e que a plataforma não tem nenhuma pretensão de indicar quais valores seriam interessantes. De fato, a plataforma foi concebida exatamente para permitir que cada aplicação defina suas métricas e como essas devem ser computadas de acordo com sua natureza e seus requisitos.

Na segunda configuração, a aplicação definiu um agente com janela de tamanho maior, compreendendo 100 eventos, e mantendo a mesma taxa de saída. Dessa forma, a cada 10 requisições processadas, o agente computa o tempo médio utilizando as últimas 100 requisições. Esse tamanho de janela foi escolhido para ficar razoavelmente distante da janela da primeira configuração que compreendia 25 eventos. O objetivo é mostrar que o tamanho da janela tem uma influência determinante no valor computado para métrica e, como veremos, na identificação dos momentos em que a adaptação deve ser realizada. Ressalto que selecionamos apenas dois valores para o parâmetro tamanho de janela para não comprometer a visualização dos resultados no gráfico.

Em ambas as configurações, caso a média computada seja superior à restrição estabelecida pela aplicação (1200 ms), o serviço de monitoração atualiza o meta-objeto representando a ligação da aplicação com o serviço requerido no modelo em execução. Nesse momento, o meta-objeto aciona, via mecanismo de *callback*, o gerente de adaptação que pergunta à aplicação se o serviço deve ser substituído. Como mencionado, a aplicação foi configurada de forma que sempre que houvesse violações, o serviço deveria ser trocado.

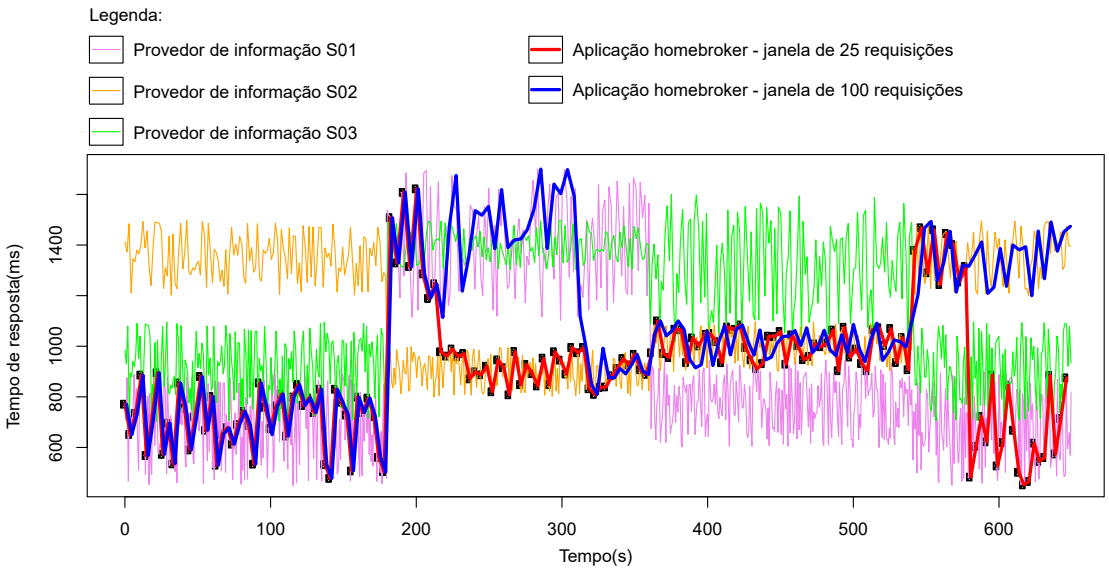
8.2.2.3 Resultados

O comportamento dos serviços candidatos e da aplicação com as duas configurações é apresentado na **Figura 44**. A linha vermelha representa a aplicação configurada com o agente de janela de tamanho 25, enquanto que a linha azul representa a configuração com o agente com janela de tamanho 100.

Como podemos observar, a aplicação (em ambas as configurações) se inicia escolhendo aquele que é, inicialmente, o melhor serviço candidato, no caso, *S1*. Durante os três primeiros minutos do experimento, as variações observadas no tempo de resposta do serviço não são suficientes para sinalizar a necessidade de reconfiguração, de forma que a aplicação permanece com o serviço selecionado.

Ao final do terceiro minuto, há um acréscimo significativo no tempo de resposta do serviço *S1*, ultrapassando os 1200 ms estabelecidos pela aplicação. Esse é um ponto importante da figura. Como se pode observar, a aplicação com janela menor percebe mais rapidamente a violação e se reconfigura de forma a começar a utilizar o serviço *S2*. Por

Figura 44 – Adaptação e configuração de monitoração



Fonte: o autor

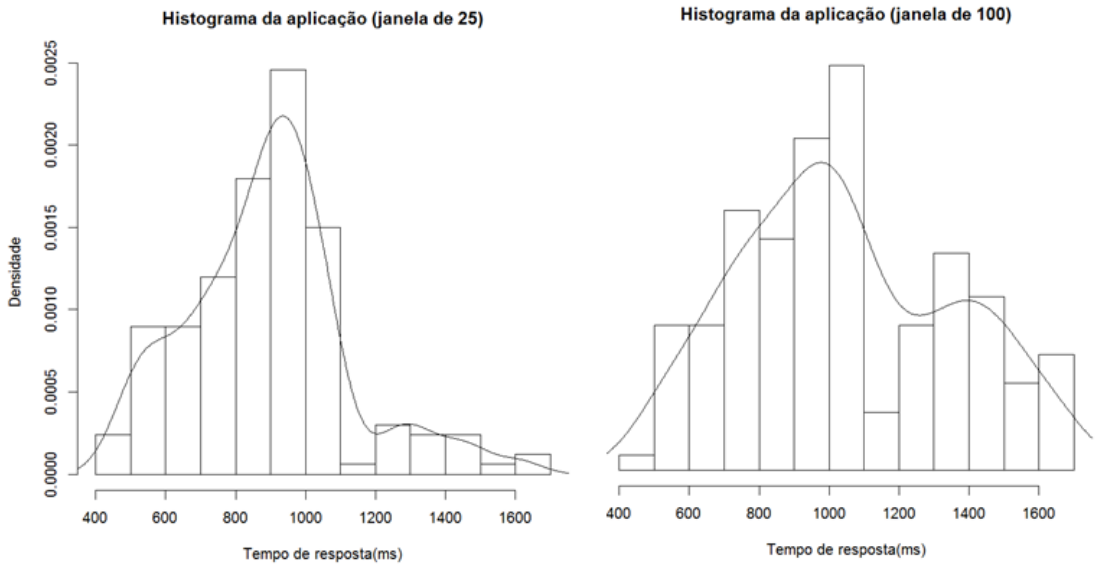
outro lado, a aplicação com janela maior demorou mais de um minuto para perceber o problema, continuando com *S1* uma vez que a média das últimas 100 requisições ainda parecia razoável. De toda forma, em ambos os casos, a violação provocou uma adaptação, sendo novamente escolhido o serviço que os agentes de monitoração indicavam como apresentando um melhor tempo de resposta (métrica de interesse da aplicação) naquele momento.

Observe que na faixa entre 360 e 540 segundos, o melhor serviço não é escolhido pelas aplicações. Como dissemos anteriormente, na plataforma DSOA, uma mudança na qualidade de um serviço que não está em uso por uma aplicação (como foi o caso) não provoca uma nova seleção. Somente quando o serviço em uso não atender às restrições estabelecidas, esse processo de seleção pode ser disparado.

Diante do exposto, cabe uma observação importante. A plataforma DSOA não pretende determinar se o agente com janela menor é melhor ou pior que o de janela maior. Há diferentes aspectos a serem considerados. Por exemplo, pode-se considerar que um agente com janela maior é mais interessante pois permite que serviços que vêm sendo utilizados há um bom tempo tenham menos chances de ser substituídos. Contudo, em outros cenários, como por exemplo, numa aplicação de *homebroker*, essa janela pode ser problemática.

Foi exatamente por perceber que não há uma resposta correta que a plataforma DSOA foi concebida. A intenção é exatamente deixar que cada aplicação, de acordo com sua própria natureza e seus requisitos, defina como quer que as métricas de qualidade sejam computadas. Como vimos, a flexibilidade vai além, cada aplicação pode definir suas métricas através de um modelo customizado de qualidade. É importante ressaltar, ainda,

Figura 45 – Histogramas com janelas diferentes



Fonte: o autor

que a configuração dos agentes pode ser dinamicamente alterada via ferramenta de administração.

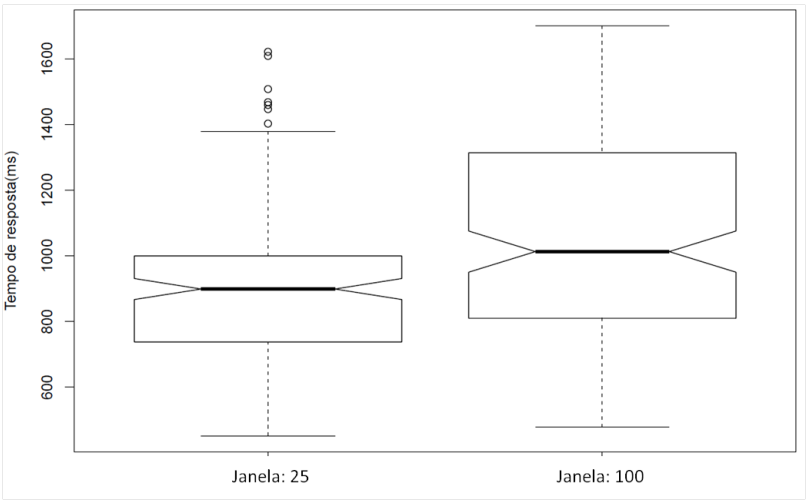
Visando dar uma melhor noção do impacto dessas configurações, a **Figura 45** apresenta os histogramas da aplicação com as duas configurações discutidas.

Como podemos observar, a aplicação com uma janela de 100 requisições acaba realizando uma quantidade bem maior de requisições para o serviço depois que ele ultrapassa o limite desejado pela aplicação. Esse fato pode ser percebido pelas barras altas à direita do tempo de resposta de 1200 ms. Por outro lado, a aplicação com janela de 25 é capaz de se adaptar mais rapidamente, observando-se proporcionalmente poucas requisições com tempo mais elevado.

Uma outra representação que ajuda a visualizar o impacto dessa configuração é apresentada na **Figura 46**. Como podemos observar, o intervalo entre o primeiro e o terceiro quartis (representado pela largura da caixa) com a janela de 25 requisições é bem menor que o intervalo equivalente com janela de 100. Uma vez que o intervalo entre esses quartis corresponde à metade das observações, percebe-se que uma janela menor tende a promover um menor espalhamento dos tempos medidos, uma vez que a necessidade de adaptação é percebida rapidamente. Observa-se, ainda, que as medianas são razoavelmente diferentes, podendo ser um fator relevante dependendo do contexto da aplicação.

De fato, as janelas grandes dificultam a percepção de variações rápidas nas métricas de qualidade monitoradas. Dessa forma, esse parâmetro dinamicamente configurável pode ter um impacto direto e significativo na qualidade dos serviços percebida pelas aplicações. Considerando-se a relevância desse parâmetro nos valores computados, a proposição de uma plataforma que ofereça facilidades para realizar esse tipo de configuração em tempo

Figura 46 – Boxplot com janelas diferentes



Fonte: o autor

de execução representa um importante diferencial. Os dados observados são apresentados de forma numérica na **Tabela 4**.

Tabela 4 – Sumário estatístico

Janela	Mín. (ms)	1o Qu. (ms)	Média (ms)	Mediana (ms)	3o Qu. (ms)	Máx. (ms)
25	452,0	738,0	899,0	892,6	999,5	1622,0
100	478,0	810,0	1013,0	1047,0	1314,0	1700,0

Fonte: o autor

8.2.3 Terceiro Experimento

8.2.3.1 Objetivo

Por fim, o terceiro experimento foi projetado para avaliar o impacto introduzido pela monitoração *passiva* realizada na plataforma DSOA. Para tanto, elaborou-se um cenário baseado na aplicação *homebroker* no qual os usuários fazem sucessivas requisições para obter a cotação de um ativo a partir de um provedor de informações. Uma vez que a finalidade do experimento é avaliar o mecanismo de monitoração, o cenário projetado envolve um único serviço candidato, não havendo nenhuma adaptação por parte da aplicação.

Para fins de avaliação foram consideradas duas métricas do ponto de vista do usuário: *throughput* e tempo de resposta. Neste cenário não utilizamos o simulador de serviços, mas sim serviços reais desenvolvidos para consultar a cotação do ativo em uma base de dados local. Em suma, pretende-se avaliar o impacto introduzido pela monitoração dos serviços e pela computação das métricas de qualidade na plataforma adotando o ponto de vista do cliente que realiza as requisições.

8.2.3.2 Configuração

Esse experimento foi realizado em uma rede isolada composta de 3 máquinas: (1) um *Pentium dual-core* com 2 GB emulando os consumidores de serviços, cada um executando em sua própria *thread*, (2) um *Pentium dual-core* com 3 GB de memória executando a aplicação na plataforma DSOA, e (3) um *Pentium dual-core* com 1 GB executando o serviço primitivo responsável por informar a cotação do ativo.

Diferentes cargas de trabalho foram projetadas variando-se o número de consumidores simultâneos. Em particular, realizou-se experimentos com: 1, 25, 50, 75, 100, 125, e 150 consumidores, cada um programado para submeter 1 requisição por segundo. No total, cada consumidor realiza 1000 requisições ao longo do experimento.

Para que o impacto da monitoração realizada pela plataforma DSOA pudesse ser mensurado, os experimentos foram realizados em duas configurações distintas: (1) plataforma com monitoração passiva habilitada, de forma que os eventos primitivos são gerados pelos sensores e encaminhados para processamento pelos agentes, (2) plataforma sem monitoração habilitada, onde o *proxy* simplesmente encaminha a requisição para o serviço sem gerar evento para processamento.

Na configuração com a monitoração habilitada, foram utilizados três agentes de processamento configurados para computar os tempos máximo, mínimo, e médio de processamento em janelas de 500 eventos.

8.2.3.3 Resultados

A **Figura 47** apresenta o *throughput* medido com e sem a monitoração habilitada. Como podemos observar, apesar do número crescente de consumidores, o *throughput* médio percebido por esses consumidores com e sem monitoração é bastante similar, sendo uma indicação que a infraestrutura de monitoração, quando utilizada no modo *passivo*, tem um baixo impacto no desempenho do sistema.

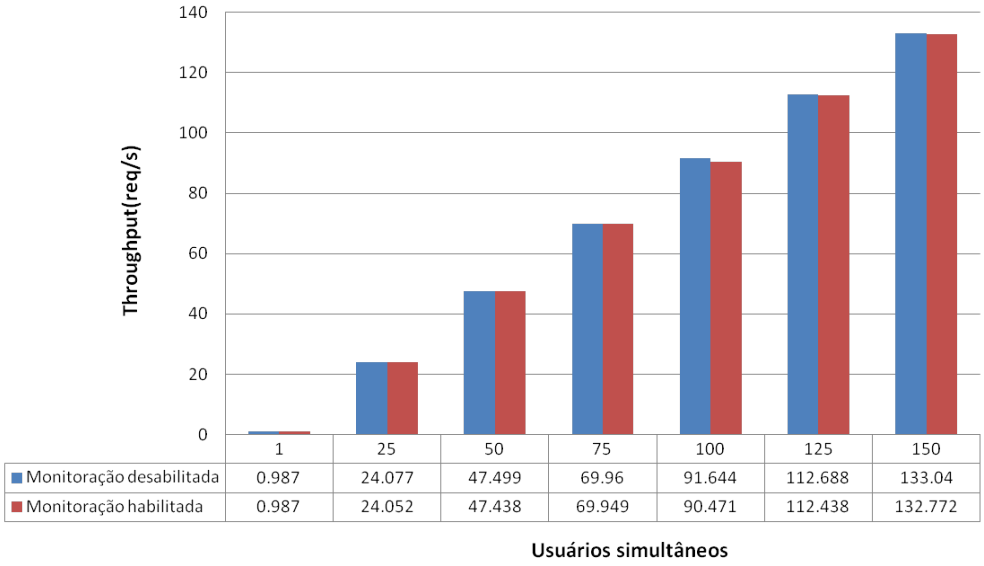
De fato, dentre as configurações de carga consideradas, aquela na qual houve uma maior diferença entre as configurações com e sem monitoração foi a que utilizou 100 usuários simultâneos. Mesmo nesse contexto, a diferença observada entre as configurações foi baixa (aproximadamente 1,3 porcento).

Além do *throughput*, também medimos nesse experimento o tempo médio de resposta observado pelos clientes. Os dados mensurados são apresentados na **Figura 48**.

Como podemos observar, os valores medidos para o tempo de resposta com e sem monitoração são próximos. De fato, a maior diferença observada foi de aproximadamente 6 porcento (com 50 usuários simultâneos).

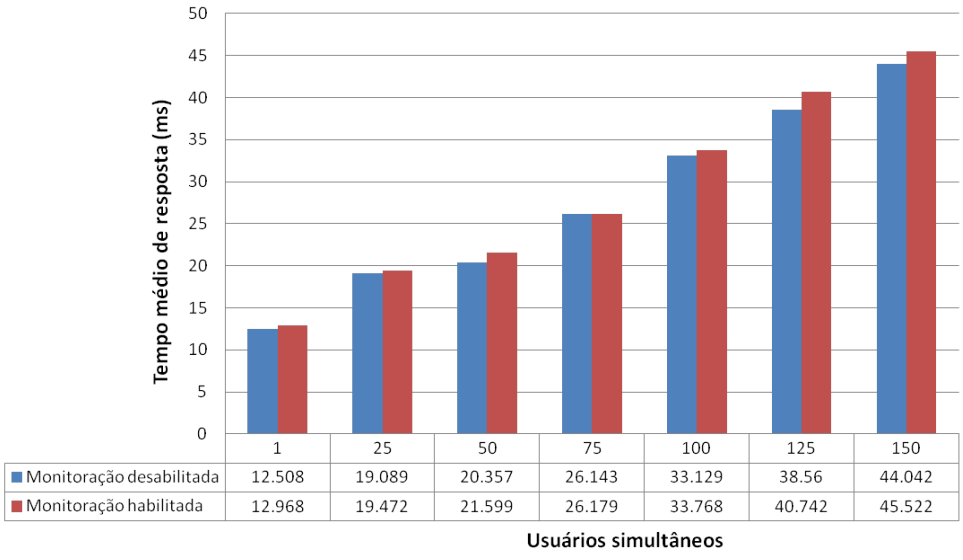
Em suma, analisando os valores mensurados para *throughput* e tempo médio de resposta, consideramos que a introdução da monitoração passiva na plataforma DSOA, além de essencial para suportar a adaptação das QSBAs, não impacta significativamente o de-

Figura 47 – Throughput



Fonte: o autor

Figura 48 – Tempo de resposta



Fonte: o autor

sempenho percebido pelos usuários. Do ponto de vista da monitoração ativa, esse impacto é certamente maior, sendo diretamente influenciado pela frequência de requisições realizadas para fins de monitoração. Desta forma, essa frequência deve ser definida com cautela e, uma vez escolhida, é recomendável a realização de novos experimentos para mensurar o impacto no desempenho introduzido no contexto específico.

8.3 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma avaliação da implementação de referência da plataforma DSOA, com um foco na capacidade de adaptação das aplicações em execução. Nesse sentido, foram projetados experimentos que mostrassem a capacidade adaptativa das aplicações em execução na plataforma. Como observamos, a plataforma foi capaz de promover, com sucesso, a adaptação das aplicações quando as restrições estabelecidas foram violadas. Um aspecto particularmente importante avaliado nos experimentos diz respeito à possibilidade de configuração, por parte das aplicações, das métricas de qualidade e da forma de monitorar essas métricas nos serviços em execução.

Como vimos, a plataforma DSOA permite que as aplicações definam agentes de processamento de eventos que computam as métricas e que podem ser dinamicamente configurados em termos do tamanho (e tipo) da janela de observação e da frequência com que esses entregam as métricas computadas para o mecanismo de monitoração. Para avaliar a relevância desse tipo de configuração, que é característico da plataforma DSOA, foi proposto um experimento no qual o tamanho da janela é variado, observando-se o efeito dessa variação em termos do tempo de resposta percebido pelas aplicações. Como conclusão desse experimento, percebemos que há um impacto direto dessa configuração nos valores observados, podendo esse impacto ser relevante no contexto das aplicações. Essa importância ratifica a proposta da plataforma que deixa a cargo da aplicação definir as métricas relevantes e o processo de computação correspondente.

Por fim, realizamos um experimento visando mensurar o impacto no desempenho observado pelos usuários como consequência da utilização do mecanismo de monitoração passiva da plataforma DSOA. Para tanto, mensuramos o *throughput* e o tempo médio de resposta observado pelos usuários em duas configurações: (1) com a monitoração habilitada, e (2) sem a monitoração habilitada. Com base nos resultados obtidos para essas duas métricas de desempenho, chegamos à conclusão de que o impacto decorrente da monitoração é bastante baixo, principalmente considerando-se os benefícios da monitoração do ponto de vista do gerenciamento das QSBAs.

9 CONCLUSÕES E TRABALHOS FUTUROS

> *“Não se pode criar experiência. É preciso passar por ela.”*
>
> Albert Camus,

Este capítulo apresenta as conclusões do presente trabalho, ressaltando as contribuições da pesquisa para o estado da arte. Ao longo do capítulo, são discutidas as principais limitações identificadas na proposta, assim como possíveis direções para pesquisas futuras.

9.1 VISÃO GERAL

Atualmente, o rápido crescimento em termos de capacidade de processamento e de comunicação dos diferentes dispositivos vem promovendo uma demanda crescente por aplicações capazes de se adaptar sozinhas em tempo de execução. Dentre essas aplicações, uma categoria especial corresponde às aplicações auto-adaptativas baseadas em serviços e cientes de qualidade. Essas aplicações são inerentemente dinâmicas, devendo ser capazes de se reconfigurar em função de variações percebidas no nível de qualidade dos serviços utilizados. Como vimos, um apoio efetivo para a construção dessas aplicações é uma tarefa complexa, evolvendo a necessidade de ambientes apropriados em tempo de desenvolvimento e de execução.

A despeito da relevância dessas aplicações, observamos que os ambientes geralmente oferecem um apoio limitado. Em particular, o universo de atributos e métricas que podem ser utilizados é, em geral, pré-estabelecido e restrito ao conjunto das características reconhecidas pelas plataformas subjacentes. Neste contexto, as aplicações devem se limitar a definir seu processo de adaptação em função das características conhecidas. Não há, assim, espaço para que as aplicações definam suas próprias métricas e o “algoritmo” que deveria ser utilizado para a sua monitoração.

Investigando a origem dessa lacuna, identificamos que alguns trabalhos já davam um passo importante na direção de superá-la, permitindo que as próprias aplicações concebessem, através de modelos abstratos, uma representação de seus elementos (componentes e serviços), assim como das características de qualidade relevantes. Esses modelos abstratos seriam transferidos para um ambiente de execução e interpretados, originando além dos elementos reais que materializam a aplicação, um modelo de qualidade mantido em execução. De uma forma geral, os conceitos definidos no espaço de modelagem pertenciam aos domínios de serviços e qualidade e eram formalizados através da proposição de meta-modelos.

Embora os modelos desenvolvidos com base nesses meta-modelos fossem bastante expressivos, permitindo uma representação adequada das aplicações do ponto de vista dos

desenvolvedores, restava uma importante lacuna. Uma vez que os modelos são interpretados pelas plataformas de execução subjacentes, os próprios modelos deveriam conter meta-dados para configurar os diferentes mecanismos que compõem essas plataformas.

Como mostramos ao longo da presente tese, os modelos utilizados nos diferentes trabalhos da área não compreendem informações suficientes para uma configuração adequada dos mecanismos de monitoração. Isso é especialmente verdadeiro, e fundamental, nos cenários que envolvem a necessidade de introdução de métricas de qualidade próprias de uma aplicação. Neste contexto, é essencial que as representações abstratas das aplicações, sob forma de modelos, incluam meta-dados que permitam que as plataformas "compreendam" como as aplicações desejam que essas métricas sejam efetivamente computadas.

Visando preencher essa lacuna, a presente tese de doutorado identificou a necessidade de introduzir nessas representações abstratas, elementos oriundos do domínio dos eventos. Do ponto de vista conceitual, a ideia é que as métricas de qualidade podem ser computadas a partir das ocorrências em tempo de execução.

Neste sentido, é necessário dar visibilidade a essas ocorrências para que os desenvolvedores possam utilizá-las na concepção de seus modelos abstratos. Em suma, propõe-se que os conceitos do domínio de eventos sejam representados no espaço de modelagem das aplicações, sendo utilizados para atribuir semântica para as métricas definidas pelas aplicações, indicando-se como essas ocorrências podem ser processadas para derivar os valores das métricas de qualidade. Em execução, esses modelos são interpretados, permitindo uma correta configuração do mecanismo de monitoração.

Para viabilizar essa solução é necessário introduzir a dimensão de eventos no nível de meta-modelo, que passa a ser composto por elementos de três domínios distintos e complementares: serviços, qualidade, e eventos.

Para validar essa ideia, propusemos uma plataforma com uma arquitetura reflexiva organizada em três camadas, denominada DSOA. No nível base, encontra-se a aplicação em execução e os elementos do domínio de negócios. No nível meta, encontram-se o gerente de adaptação e os modelos responsáveis por representar a aplicação em execução. Acima deles, no nível meta-meta, encontram-se os modelos que representam os processos de supervisão e adaptação. Cabe a esses modelos representar como as métricas de qualidade devem ser monitoradas.

9.2 CONTRIBUIÇÕES

A contribuição central desta tese é introduzir os conceitos do domínio de eventos no espaço de modelagem das aplicações, possibilitando o desenvolvimento de aplicações auto-adaptativas mais configuráveis e flexíveis. Esses conceitos compõem, juntamente com aqueles pertencentes aos domínios de serviços e qualidade, um espaço multi-dimensional, que permite tanto a representação das aplicações, quanto dos mecanismos internos de um plataforma de execução. Neste contexto, as aplicações são capazes de definir, através

de modelos, quais métricas devem ser utilizadas no processo de adaptação, e como elas devem ser computadas, influenciando, de forma decisiva, a identificação da necessidade de adaptação. Como elementos da contribuição, podemos ainda elencar:

1. Um meta-modelo multi-dimensional, composto a partir de elementos dos domínios de serviços, qualidade, e eventos. Este meta-modelo define os elementos conceituais que formam as QSBAs.

2. Um modelo de componentes orientado a serviços capaz de suportar a composição dinâmica em função de características de qualidade definidas pelas aplicações.

3. Uma nova plataforma para o desenvolvimento e execução de aplicações auto-adaptativas baseadas em serviço e cientes de qualidade. Essa plataforma utiliza modelos em tempo de execução tanto para conduzir a adaptação das aplicações, como dos mecanismos de supervisão e adaptação embutidos na própria plataforma.

4. A proposição de um conjunto de DSLs integradas para suportar a modelagem das aplicações auto-adaptativas.

9.3 LIMITAÇÕES E TRABALHOS FUTUROS

A despeito da presente proposta ter contribuído para ampliar as possibilidades de configuração e gerenciamento das aplicações auto-adaptativas, identificamos algumas limitações ao longo do seu desenvolvimento. Em primeiro lugar, cabe observar que a plataforma considera que os serviços substituídos não guardam informação de estado, de forma que esse aspecto não é abordado.

Outra limitação diz respeito a uma eventual necessidade de integração com sistemas desenvolvidos em outras plataformas. Neste contexto particular, existem diversos aspectos a serem tratados. Um exemplo claro diz respeito aos meta-modelos adotados na plataforma, os quais não são baseados em um meta-meta-modelo, como ocorre quando da utilização de uma linguagem como MOF. A plataforma assume ainda que os serviços utilizados possuem uma interface Java, não sendo tratada a construção automática de adaptadores.

Do ponto de vista dos trabalhos futuros, uma importante área a ser explorada diz respeito à capacidade de predição. Atualmente a plataforma conduz seu processo de adaptação de uma forma reativa, utilizando as violações nas restrições estabelecidas como gatilho. Uma vez que os mecanismos necessários para coleta das informações estão prontos, seria interessante aproveitar as informações obtidas para realizar adaptações com base em predições de desempenho, eventualmente baseadas em modelos de redes de filas ou redes de Petri.

Outro aspecto importante é que a plataforma não explorou as possibilidades de configuração do ponto de vista dos provedores de serviços. Assim, embora as métricas relativas

aos serviços sejam monitoradas, a utilização dos dados obtidos está restrita à substituição dos serviços utilizados pela aplicação. Em particular, a perspectiva do provedor de serviços é bastante rica, podendo envolver reconfigurações em termos do serviço de concorrência ou mesmo de elementos de um ambiente de nuvem.

Por fim, diferentes aspectos concernentes à reconfiguração dinâmica da própria plataforma não foram explorados. Embora a plataforma já seja bastante customizável, uma possibilidade de extensão interessante seria a definição de meta-políticas, ou seja, políticas capazes de avaliar e, eventualmente, substituir as políticas em uso.

REFERÊNCIAS

ALLEN, R.; GARLAN, D. Beyond definition/use: Architectural interconnection. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 29, n. 8, p. 35–45, ago. 1994. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/185087.185101>.

ANDERSSON, J.; LEMOS, R. D.; MALEK, S.; WEYNS, D. Reflecting on self-adaptive software systems. p. 38–47, 2009.

BARESI, L.; Di Nitto, E.; GHEZI, C. Toward open-world software: Issues and challenges. *Computer*, v. 39, n. 10, p. 36–43, 2006. ISSN 00189162.

BEISIEGEL, M.; BOOZ, D.; COLYER, A.; HILDEBRAND, H.; MARINO, J.; TAM, K. *SCA Service Component Architecture*. 2007. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.168.5919>.

BENBERNOU, S.; BRANDIC, I.; CAPPIELLO, C.; CARRO, M.; COMUZZI, M.; KERTÉSZ, A.; KRITIKOS, K.; PARKIN, M.; PERNICI, B.; PLEBANI, P. Service research challenges and solutions for the future internet. In: PAPAZOGLOU, M.; POHL, K.; PARKIN, M.; METZGER, A. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2010. cap. Modeling and Negotiating Service Quality, p. 157–208. ISBN 3-642-17598-8, 978-3-642-17598-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1985668.1985674>.

BENNACEUR, A.; FRANCE, R.; TAMBURRELLI, G.; VOGEL, T.; MOSTERMAN, P. J.; CAZZOLA, W.; COSTA, F. M.; PIERANTONIO, A.; TICHY, M.; AKSIT, M.; EMMANUELSON, P.; GANG, H.; GEORGANTAS, N.; REDLICH, D. Mechanisms for leveraging models at runtime in self-adaptive software. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 8378 LNCS, p. 19–46, 2014. ISSN 16113349.

BETTINI, L. *Implementing Domain Specific Languages with Xtext and Xtend - Second Edition*. 2nd. ed. [S.l.]: Packt Publishing, 2016. ISBN 1786464969, 9781786464965.

BÉZIVIN, J. Model driven engineering:an emerging technical space. *Lecture Notes in Computer Science*, v. 4143, p. 36–64, 2006. ISSN 16113349.

BLAIR, G. S.; BENCOMO, N.; FRANCE, R. B. Models@ run.time. *IEEE Computer*, v. 42, n. 10, p. 22–27, 2009. Disponível em: <http://dx.doi.org/10.1109/MC.2009.326>.

BOCCIARELLI, P.; D'AMBROGIO, A. A model-driven method for describing and predicting the reliability of composite services. *Software & Systems Modeling*, v. 10, n. 2, p. 265–280, May 2011. ISSN 1619-1374.

BRETON, E.; BÉZIVIN, J. Towards an understanding of model executability. *Proceedings of the international conference on Formal Ontology in Information Systems FOIS 01*, p. 70–80, 2001. Disponível em: <http://portal.acm.org/citation.cfm?doid=505168.505176>.

BROSIG, F.; HUBER, N.; KOUNEV, S. Architecture-Level Software Performance Abstractions for Online Performance Prediction. *Elsevier Science of Computer Programming Journal (SciCo)*, Elsevier, v. 90, Part B, p. 71–92, 2014. ISSN 0167-6423. Disponível em: <http://authors.elsevier.com/sd/article/S0167642313001421>.

BRUN, Y.; SERUGENDO, G. M.; GACEK, C.; GIESE, H.; KIENLE, H.; LITOIU, M.; MULLER, H.; PEZZE, M.; SHAW, M. Engineering self-adaptive systems through feedback loops software engineering for self-adaptive systems. v. 5525, p. 48–70, 2009.

BRUNETON, E. The fractal component model and its support in java. *Software - Practice and Experience*, v. 39, n. 7, p. 701–736, 2009. ISSN 00380644.

BRUNI, R.; CORRADINI, A.; GADDUCCI, F.; LAFUENTE, A. L.; VANDIN, A. A conceptual framework for adaptation. In: *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2012. (FASE'12), p. 240–254. ISBN 978-3-642-28871-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-28872-2_17>.

BURNSTEIN, I. *Practical Software Testing: A Process-Oriented Approach*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 1441928855, 9781441928856.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. [S.l.]: Wiley Publishing, 1996. ISBN 0471958697, 9780471958697.

CAVALCANTI, D. J. M.; SOUZA, F. N.; ROSA, N. S. Adaptive and dynamic quality-aware service selection. In: *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013, Belfast, United Kingdom, February 27 - March 1, 2013*. [s.n.], 2013. p. 323–327. Disponível em: <http://dx.doi.org/10.1109/PDP.2013.60>.

CERVANTES, H. *Toward a service-oriented component model to support dynamic availability*. Tese (Theses) — Université Joseph-Fourier - Grenoble I, mar. 2004. Disponível em: <https://tel.archives-ouvertes.fr/tel-00005929>.

CERVANTES, H.; HALL, R. S. Autonomous adaptation to dynamic availability using a service-oriented component model. In: *Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004. (ICSE '04), p. 614–623. ISBN 0-7695-2163-0. Disponível em: <http://dl.acm.org/citation.cfm?id=998675.999465>.

CHAMBERS, J.; CLEVELAND, W.; KLEINER, B.; TUKEY, P. Graphical methods for data analysis. *The Wadsworth Statistics/Probability Series. Boston, MA: Duxury*, 1983.

CHENG, B. H.; LEMOS, R.; GIESE, H.; INVERARDI, P.; MAGEE, J.; ANDERSSON, J.; BECKER, B.; BENCOMO, N.; BRUN, Y.; CUKIC, B.; SERUGENDO, G. M.; DUSTDAR, S.; FINKELSTEIN, A.; GACEK, C.; GEIHS, K.; GRASSI, V.; KARSAI, G.; KIENLE, H. M.; KRAMER, J.; LITOIU, M.; MALEK, S.; MIRANDOLA, R.; MüLLER, H. A.; PARK, S.; SHAW, M.; TICHY, M.; TIVOLI, M.; WEYNS, D.; WHITTLE, J. Software engineering for self-adaptive systems. In: CHENG, B. H.; LEMOS, R.; GIESE, H.; INVERARDI, P.; MAGEE, J. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. Software Engineering for Self-Adaptive Systems: A Research Roadmap, p. 1–26. ISBN 978-3-642-02160-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-02161-9_1>.

CHENG, S.-W.; GARLAN, D. Stitch: A language for architecture-based self-adaptation. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 85, n. 12, p. 2860–2875, dez. 2012. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2012.02.060>.

CHUNG, L.; LEITE, J. C. P. Conceptual modeling: Foundations and applications. In: BORGIDA, A. T.; CHAUDHRI, V. K.; GIORGINI, P.; YU, E. S. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. On Non-Functional Requirements in Software Engineering, p. 363–379. ISBN 978-3-642-02462-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-02463-4_19>.

COMBEMALE, B.; DEANTONI, J.; BAUDRY, B.; FRANCE, R. B.; JEZEQUEL, J.-M.; GRAY, J. Globalizing modeling languages. *Computer*, IEEE Computer Society, Los Alamitos, CA, USA, v. 47, n. 6, p. 68–71, 2014. ISSN 0018-9162.

CONAN, D.; PUTRYCZ, E.; FARCET, N.; DEMIGUEL, M. Integration of non-functional properties in containers. In: *In Proceedings of the 6th International Workshop on Component-Oriented Programming (WCOP*. [S.l.: s.n.], 2001.

COSTA, F. Combining meta-information management and reflection in an architecture for configurable and reconfigurable middleware. *Philosophy, Computing Department Lancaster University*, n. August, 2001. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.6563{&}rep=rep1{&}ty>.

CRNKOVIC, I.; SENTILLES, S.; VULGARAKIS, A.; CHAUDRON, M. R. V. A classification framework for software component models. *IEEE Trans. Software Eng.*, v. 37, n. 5, p. 593–615, 2011. Disponível em: <https://doi.org/10.1109/TSE.2010.83>.

D'AMBROGIO, A. A model-driven wsdl extension for describing the qos of web services. In: *Proceedings - ICWS 2006: 2006 IEEE International Conference on Web Services*. [s.n.], 2006. p. 789–796. ISBN 0769526691; 9780769526690. Cited By 69. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-38949121932&partnerID=40&md5=a1ab2cc907fb8abdedb50d97ab6a9776>.

DAVID, P.-C.; LEDOUX, T. An aspect-oriented approach for developing self-adaptive fractal components. In: LÖWE, W.; SÜDHOLT, M. (Ed.). *Software Composition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 82–97. ISBN 978-3-540-37659-0.

DAVID, P.-C.; LEDOUX, T.; LÉGER, M.; COUPAYE, T. Fpath and fscript: Language support for navigation and reliable reconfiguration of fractal architectures. *annals of telecommunications - annales des télécommunications*, v. 64, n. 1, p. 45–63, Feb 2009. ISSN 1958-9395. Disponível em: <https://doi.org/10.1007/s12243-008-0073-y>.

DAVIS, J. *Open Source Soa*. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2009. ISBN 1933988541, 9781933988542.

DEREMER, F.; KRON, H. H. Programming-in-the-large versus programming-in-the-small. *IEEE Transactions on Software Engineering*, v. 2, n. 2, p. 80–86, June 1976.

Di Nitto, E.; GHEZZI, C.; METZGER, A.; PAPAZOGLOU, M.; POHL, K. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, v. 15, n. 3-4, p. 313–341, sep 2008. ISSN 0928-8910. Disponível em: <http://link.springer.com/10.1007/s10515-008-0032-x>.

DUSTDAR, S.; MICHLMAYR, A.; ROSENBERG, F.; LEITNER, P. End-to-end support for qos-aware service selection, binding, and mediation in vresco. *IEEE Transactions*

on Services Computing, v. 3, p. 193–205, 04 2010. ISSN 1939-1374. Disponível em: <doi.ieeecomputersociety.org/10.1109/TSC.2010.20>.

ESCOFFIER, C.; BOURRET, P.; LALANDA, P. Describing dynamism in service dependencies: Industrial experience and feedbacks. In: *2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*. [s.n.], 2013. p. 328–335. Disponível em: <https://doi.org/10.1109/SCC.2013.82>.

ESCOFFIER, C.; HALL, R. S.; LALANDA, P. ipojo: an extensible service-oriented component framework. In: *2007 IEEE International Conference on Services Computing (SCC 2007), 9-13 July 2007, Salt Lake City, Utah, USA*. [s.n.], 2007. p. 474–481. Disponível em: <http://dx.doi.org/10.1109/SCC.2007.74>.

ESFAHANI, N.; MALEK, S.; SOUSA, J. P.; GOMAA, H.; MENASCÉ, D. A. A modeling language for activity-oriented composition of service-oriented software systems. In: SCHÜRR, A.; SELIC, B. (Ed.). *Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 591–605. ISBN 978-3-642-04425-0.

ETZION, O.; NIBLETT, P. *Event processing in action*. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2010. ISBN 1935182218, 9781935182214.

FAVRE, J.-M. Foundations of meta-pyramids: Languages vs. metamodels - episode ii: Story of thotus the baboon1. In: BEZIVIN, J.; HECKEL, R. (Ed.). *Language Engineering for Model-Driven Software Development*. [S.l.]: Internationales Begegnungs- und Forschungszentrum fur Informatik(IBFI), Schloss Dagstuhl, Germany, 2004. (Dagstuhl Seminar Proceedings, v. 04101).

FERBER, J. Computational reflection in class based object oriented languages. In: *Proc. of the OOPSLA-89: Conference on Object-Oriented Programming: Systems*. Languages and Applications, New Orleans, LA: [s.n.], 1989. p. 317–326.

FLOCH, J.; HALLSTEINSEN, S.; STAV, E.; ELIASSEN, F.; LUND, K.; GJORVEN, E. Using architecture models for runtime adaptability. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 2, p. 62–70, mar. 2006. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2006.61>.

FOUQUET, F.; NAIN, G.; MORIN, B.; DAUBERT, E.; BARAIS, O.; PLOUZEAU, N.; JÉZÉQUEL, J. Kevoree modeling framework (KMF): efficient modeling techniques for runtime use. *CoRR*, abs/1405.6817, 2014. Disponível em: <http://arxiv.org/abs/1405.6817>.

FOWLER, M. *Domain-Specific Languages*. [S.l.]: Addison-Wesley Professional, 2010. ISBN 0321712943.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. *Future of Software Engineering (FOSE '07)*, n. May 2007, p. 37–54, 2007. ISSN 0018067X.

GARLAN, D.; CHENG, S.-W.; HUANG, A.-C.; SCHMERL, B.; STEENKISTE, P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 37, n. 10, p. 46–54, out. 2004. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2004.175>.

GARLAN, D.; SCHMERL, B. R.; CHENG, S.-W. Software architecture-based self-adaptation. In: ZHANG, Y.; YANG, L. T.; DENKO, M. K. (Ed.). *Autonomic Computing and Networking*. Springer, 2009. p. 31–55. ISBN 978-0-387-89828-5. Disponível em: <http://dblp.uni-trier.de/db/books/collections/DYZ2009.html#GarlanSC09>.

GASEVIC, D.; KAVIANI, N.; HATALA, M. On metamodeling in megamodels. In: ENGELS, G.; OPDYKE, B.; SCHMIDT, D. C.; WEIL, F. (Ed.). *MoDELS*. [S.l.]: Springer, 2007. (Lecture Notes in Computer Science, v. 4735), p. 91–05. ISBN 978-3-540-75208-0.

GROUP, O. M. *CORBA Component Model 4.0 Specification*. [S.l.], 2006. Disponível em: <http://www.omg.org/docs/formal/06-04-01.pdf>.

HALLSTEINSEN, S.; GEIHS, K.; PASPALLIS, N.; ELIASSEN, F.; HORN, G.; LORENZO, J.; MAMELLI, A.; PAPADOPOULOS, G. A. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 85, n. 12, p. 2840–2859, dez. 2012. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2012.07.052>.

HEINEMAN, G. T.; COUNCILL, W. T. (Ed.). *Component-based Software Engineering: Putting the Pieces Together*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN 0-201-70485-4.

HEINRICH, R. Architectural run-time models for performance and privacy analysis in dynamic cloud applications. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 43, n. 4, p. 13–22, fev. 2016. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/2897356.2897359>.

HEINRICH, R.; JUNG, R.; ZIRKELBACH, C.; HASSELBRING, W.; REUSSNER, R. Software architecture for big data and the cloud. In: _____. [S.l.]: Elsevier, 2017. cap. An Architectural Model-Based Approach to Quality-aware DevOps in Cloud Applications. To appear.

HINCHEY, M. G.; STERRITT, R. Self-managing software. *IEEE Computer*, v. 39, n. 2, p. 107–109, 2006. Disponível em: <http://dblp.uni-trier.de/db/journals/computer/computer39.html#HincheyS06>.

HNETYNKA, P.; PLASIL, F. Using meta-modeling in design and implementation of component-based systems: the sofa case study. *Software: Practice and Experience*, John Wiley & Sons, Ltd., v. 41, n. 11, p. 1185–1201, 2011. ISSN 1097-024X. Disponível em: <http://dx.doi.org/10.1002/spe.1036>.

HUBER, N.; BROSIG, F.; SPINNER, S.; KOUNEV, S.; BÄHR, M. Model-based self-aware performance and resource management using the descartes modeling language. *IEEE Transactions on Software Engineering (TSE)*, IEEE Computer Society, v. 43, n. 5, 2017. Disponível em: <http://dx.doi.org/10.1109/TSE.2016.2613863>.

HUBER, N.; HOORN, A. van; KOZIOLEK, A.; BROSIG, F.; KOUNEV, S. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. *Service Oriented Computing and Applications*, v. 8, n. 1, p. 73–89, 2014. ISSN 18632386.

HUBER, N.; HOORN, A. van; KOZIOLEK, A.; BROSIG, F.; KOUNEV, S. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. *Service Oriented Computing and Applications Journal*, Springer London, v. 8, n. 1, p. 73–89, 2014.

HURSCH, W. L.; LOPES, C. V. Separation of concerns. Citeseer, 1995.

IBM (Ed.). *An Architectural Blueprint for Autonomic Computing*. [S.l.], 2005.

ISO/IEC. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. [S.l.], 2010.

JÉZÉQUEL, J.-M.; BARAIS, O.; FLEUREY, F. Model driven language engineering with kermeta. In: *Proceedings of the 3rd International Summer School Conference on Generative and Transformational Techniques in Software Engineering III*. Berlin, Heidelberg: Springer-Verlag, 2011. (GTTSE'09), p. 201–221. ISBN 3-642-18022-1, 978-3-642-18022-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1949925.1949931>.

JURETA, I.; HERSSENS, C.; FAULKNER, S. A comprehensive quality model for service-oriented systems. *Software Quality Journal*, v. 17, n. 1, p. 65–98, 2009. Disponível em: <https://doi.org/10.1007/s11219-008-9059-2>.

KELLER, A.; LUDWIG, H. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, Plenum Press, New York, NY, USA, v. 11, n. 1, p. 57–81, mar. 2003. ISSN 1064-7570. Disponível em: <https://doi.org/10.1023/A:1022445108617>.

KRAMER, J.; MAGEE, J. Self-managed systems: An architectural challenge. *FoSE 2007: Future of Software Engineering*, p. 259–268, 2007.

KRITIKOS, K.; CARRO, M.; PERNICI, B.; PLEBANI, P.; CAPPIELLO, C.; COMUZZI, M.; BENRERNOU, S.; BRANDIC, I.; KERTÉSZ, A.; PARKIN, M. A survey on service quality description. *ACM Computing Surveys*, v. 46, n. 1, p. 1–58, 2013. ISSN 03600300. Disponível em: <http://dl.acm.org/citation.cfm?doid=2522968.2522969>.

KRITIKOS, K.; PLEXOUSAKIS, D. Requirements for qos-based web service description and discovery. *IEEE Transactions on Services Computing*, v. 2, n. 4, p. 320–337, 2009. ISSN 19391374.

KRUPITZER, C.; ROTH, F. M.; VANSYCKEL, S.; SCHIELE, G.; BECKER, C. A survey on engineering approaches for self-adaptive systems. *Pervasive Mob. Comput.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 17, n. PB, p. 184–206, fev. 2015. ISSN 1574-1192. Disponível em: <http://dx.doi.org/10.1016/j.pmcj.2014.09.009>.

LALANDA, P.; ESCOFFIER, C. Resource-oriented framework for representing pervasive context. In: *IEEE International Congress on Internet of Things, ICIOT 2017, Honolulu, HI, USA, June 25-30, 2017*. [s.n.], 2017. p. 155–158. Disponível em: <https://doi.org/10.1109/IEEE.ICIOT.2017.27>.

LAU, K.; WANG, Z. Software component models. *IEEE Transactions on Software Engineering*, Institute of Electrical and Electronics Engineers, v. 33, n. 10, p. 709–724, 10 2007. ISSN 0098-5589.

LEHMANN, G.; BLUMENDORF, M.; TROLLMANN, F.; ALBAYRAK, S. Meta-modeling runtime models. In: *Proceedings of the 2010 International Conference on Models in Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2011. (MODELS'10), p. 209–223. ISBN 978-3-642-21209-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2008503.2008532>.

LEMOS, R. de; GIESE, H.; MULLER, H. A.; SHAW, M.; ANDERSSON, J.; BARESI, L.; BECKER, B.; BENCOMO, N.; BRUN, Y.; CUKIC, B.; DESMARAIS, R.; DUSTDAR, S.; ENGELS, G.; GEIHS, K.; GOESCHKA, K. M.; GORLA, A.; GRASSI, V.; INVERARDI, P.; KARSAI, G.; KRAMER, J.; LITOIU, M.; LOPES, A.; MAGEE, J.; MALEK, S.; MANKOVSKII, S.; MIRANDOLA, R.; MYLOPOULOS, J.; NIERSTRASZ, O.; PEZZE, M.; PREHOFER, C.; SCHAFER, W.; SCHLICHTING, R.; SCHMERL, B.; SMITH, D. B.; SOUSA, J. P.; TAMURA, G.; TAHVILDARI, L.; VILLEGAS, N. M.; VOGEL, T.; WEYNS, D.; WONG, K.; WUTTKE, J. Software engineering for self-adaptive systems: A second research roadmap. In: LEMOS, R. de; GIESE, H.; MULLER, H. A.; SHAW, M. (Ed.). *Software Engineering for Self-Adaptive Systems II*. [S.l.]: Springer-Verlag, 2013. v. 7475, p. 1–32. ISBN 978-3-642-35813-5. DOI: 10.1007/978-3-642-35813-5_1.

LIU, Y.; NGU, A. H.; ZENG, L. Z. Qos computation and policing in dynamic web service selection. In: *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &Amp; Posters*. New York, NY, USA: ACM, 2004. (WWW Alt. '04), p. 66–73. ISBN 1-58113-912-8. Disponível em: <http://doi.acm.org/10.1145/1013367.1013379>.

LUCKHAM, D. C. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN 0201727897.

MACíAS-ESCRIVá, F. D.; HABER, R. E.; TORO, R. M. del; HERNáNDEZ, V. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Syst. Appl.*, v. 40, n. 18, p. 7267–7279, 2013. Disponível em: <http://dblp.uni-trier.de/db/journals/eswa/eswa40.html#Macias-EscrivaHTH13>.

MAES, P. Concepts and experiments in computational reflection. In: MEYROWITZ, N. K. (Ed.). *OOPSLA*. ACM, 1987. p. 147–155. ISBN 0-89791-247-0. Disponível em: <http://dblp.uni-trier.de/db/conf/oopsla/oopsla87.html#Maes87>.

MALEK, S.; GOMAA, H.; MENASCé, D.; SOUSA, J. Sassy: A framework for self-architecting service-oriented systems. *IEEE Software*, v. 28, p. 78–85, 01 2011. ISSN 0740-7459. Disponível em: <doi.ieeecomputersociety.org/10.1109/MS.2011.22>.

MCKINLEY, P. K.; SADJADI, S. M.; KASTEN, E. P.; CHENG, B. H. C. Composing adaptive software. *Computer*, v. 37, n. 7, p. 56–64, 2004. ISSN 00189162.

MOOIJ, A.; HOOMAN, J. *Creating a Domain Specific Language (DSL) with Xtext*. 2018. Http://www.cs.kun.nl/J.Hooman/DSL/Creating_a_Domain_Specific_Language_(DSL)_with_Xt

MORIN, B.; BARAIS, O. K@rt: An aspect-oriented and model-oriented framework for dynamic software product lines. *Modelsruntime workshop at MoDELS08*, v. 08, 2008. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.1122{&}rep=rep1{&}ty>.

NITTO, D. Research challenges on adaptive software and services in the future internet: towards an s-cube research roadmap. *Systems Research-Results and Challenges (S-Cube)*, p. 1–7, 2012. Disponível em: <http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000006225501>.

OMG. *OMG Meta Object Facility (MOF) Core Specification, Version 2.5*. 2015. Disponível em: <http://www.omg.org/spec/MOF/2.5>.

OREIZY, P.; GORLICK, M. M.; TAYLOR, R. N.; HEIMBIGNER, D.; JOHNSON, G.; MEDVIDOVC, N.; QUILICI, A.; ROSENBLUM, D. S.; WOLF, A. L. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, v. 14, n. 3, p. 54–62, 1999. ISSN 10947167.

ORIOL, M.; MARCO, J.; FRANCH, X. Quality models for web services: A systematic mapping. *Information and Software Technology*, v. 56, n. 10, p. 1167 – 1182, 2014. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584914000822>.

OVERBEEK, I. J. F. *Meta Object Facility (MOF) investigation of the state of the art*. [S.l.], 2006. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.4092>.

PAPAZOGLOU, M. P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-oriented computing: State of the art and research challenges. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 40, n. 11, p. 38–45, nov. 2007. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2007.400>.

PARASHAR, M.; HARIRI, S. Autonomic computing: An overview. In: *Unconventional Programming Paradigms*. [S.l.]: Springer Verlag, 2005. p. 247–259.

PARRA, C.; BLANC, X.; CLEVE, A.; DUCHIEN, L. Unifying design and runtime software adaptation using aspect models. *Science of Computer Programming*, Elsevier, v. 76, n. 12, p. 1247–1260, jan. 2011. Disponível em: <https://hal.inria.fr/inria-00564592>.

PARRA, C.; BLANC, X.; DUCHIEN, L. Context awareness for dynamic service-oriented product lines. In: *Proceedings of the 13th International Software Product Line Conference*. Pittsburgh, PA, USA: Carnegie Mellon University, 2009. (SPLC '09), p. 131–140. Disponível em: <http://dl.acm.org/citation.cfm?id=1753235.1753254>.

REUSSNER, R. H.; BECKER, S.; HAPPE, J.; HEINRICH, R.; KOZIOLEK, A.; KOZIOLEK, H.; KRAMER, M.; KROGMANN, K. *Modeling and Simulating Software Architectures: The Palladio Approach*. [S.l.]: The MIT Press, 2016. ISBN 026203476X, 9780262034760.

ROMERO, D.; ROUVOY, R.; SEINTURIER, L.; CHABRIDON, S.; CONAN, D.; PESSEMIER, N. Enabling context-aware web services: A middleware approach for ubiquitous environments. In: SHENG, M.; YU, J.; DUSTDAR, S. (Ed.). *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*. Chapman and Hall/CRC, 2010. p. 113–135. Disponível em: <https://hal.inria.fr/inria-00414070>.

ROUVOY, R.; MERLE, P. Leveraging component-based software engineering with fraclet. *Annales des Télécommunications*, v. 64, n. 1-2, p. 65–79, 2009. Disponível em: <https://doi.org/10.1007/s12243-008-0072-z>.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software : Landscape and research challenges. V, n. March, p. 1–40, 2009.

SEINTURIER, L.; MERLE, P.; FOURNIER, D.; DOLET, N.; SCHIAVONI, V.; STEFANI, J.-B. Reconfigurable sca applications with the frascati platform. In: *IEEE SCC*. IEEE Computer Society, 2009. p. 268–275. ISBN 978-0-7695-3811-2. Disponível em: <http://dblp.uni-trier.de/db/conf/IEEEscc/scc2009.html#SeinturierMFDSS09>.

SEINTURIER, L.; MERLE, P.; ROUVOY, R.; ROMERO, D.; SCHIAVONI, V.; STEFANI, J.-B. A component-based middleware platform for reconfigurable service-oriented architectures. *Softw., Pract. Exper.*, v. 42, n. 5, p. 559–583, 2012. Disponível em: <http://dblp.uni-trier.de/db/journals/spe/spe42.html#SeinturierMRRSS12>.

SENTILLES, S. *Managing Extra-Functional Properties in Component-Based Development of Embedded Systems*. Tese (Doutorado) — Mälardalen University, Västerås, Sweden, June 2012. Disponível em: <http://www.es.mdh.se/publications/2563->.

SMITH, B. C. *Reflection and semantics in a procedural language*. Tese (Doutorado), 1982. PHD. Disponível em: <http://opac.inria.fr/record=b1000867>.

SOUZA, F. N.; CAVALCANTI, D. J. M.; SILVA, T. C. D.; ROSA, N. S. Ranking strategies for quality-aware service selection. In: *IEEE International Conference on Services Computing, SCC 2014, Anchorage, AK, USA, June 27 - July 2, 2014*. [s.n.], 2014. p. 115–122. Disponível em: <http://dx.doi.org/10.1109/SCC.2014.24>.

SOUZA, F. N.; LOPES, D.; GAMA, K.; ROSA, N. S.; LIMA, R. M. F. Dynamic event-based monitoring in a SOA environment. In: *On the Move to Meaningful Internet Systems: OTM 2011 - Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part II*. [s.n.], 2011. p. 498–506. Disponível em: <http://dx.doi.org/10.1007/978-3-642-25106-1_6>.

SOUZA, F. N.; SILVA, T. C. D.; CAVALCANTI, D. J. M.; ROSA, N. S.; LIMA, R. M. F. A meta-model for qos monitoring in a dynamic service-component platform. In: *SCC*. [S.l.]: IEEE, 2015. p. 459–466.

SOUZA, F. N.; SILVA, T. C. D.; ROSA, N. S. Monitoring solution in a dynamic service-oriented platform. *Computers and Electrical Engineering*, 2017.

STEINBERG, D.; BUDINSKY, F.; PATERNOSTRO, M.; MERKS, E. *EMF: Eclipse Modeling Framework 2.0*. 2nd. ed. [S.l.]: Addison-Wesley Professional, 2009. ISBN 0321331885.

SZYPERSKI, C. *Component Software: Beyond Object-oriented Programming*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1998. ISBN 0-201-17888-5.

The OSGi Alliance. *OSGi Service Platform Core Specification, Release 4.1*. 2007. <http://www.osgi.org/Specifications>.

VOELTER, M.; BENZ, S.; 0003, C. D.; ENGELMANN, B.; HELANDER, M.; KATS, L. C. L.; VISSER, E.; WACHSMUTH, G. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. [S.l.]: dslbook.org, 2013. 1-558 p. ISBN 978-1-4812-1858-0.

VOGEL, T.; GIESE, H. Adaptation and abstract runtime models. In: *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. New York, NY, USA: ACM, 2010. (SEAMS '10), p. 39–48. ISBN 978-1-60558-971-8. Disponível em: <http://doi.acm.org/10.1145/1808984.1808989>.

VOGEL, T.; GIESE, H. Requirements and assessment of languages and frameworks for adaptation models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 7167 LNCS, p. 167–182, 2012. ISSN 03029743.

VOGEL, T.; GIESE, H. *Model-Driven Engineering of Adaptation Engines for Self-Adaptive Software: Executable Runtime Megamodels*. [S.l.], 2013. Disponível em: <http://opus.kobv.de/ubp/volltexte/2013/6382/>.

VOGEL, T.; NEUMANN, S.; HILDEBRANDT, S.; GIESE, H.; BECKER, B. Incremental Model Synchronization for Efficient Run-Time Monitoring. In: GHOSH, S. (Ed.). *Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers*. Springer-Verlag, 2010, (Lecture Notes in Computer Science (LNCS), v. 6002). p. 124–139. Disponível em: <http://dx.doi.org/10.1007/978-3-642-12261-3_13>.

VOGEL, T.; SEIBEL, A.; GIESE, H. Toward megamodels at runtime. In: *International Workshop on Models@run.time*. CEUR-WS.org, 2010. (CEUR Workshop Proceedings, v. 641), p. 13–24. (best paper). Disponível em: <http://ceur-ws.org/Vol-641/paper_14.pdf>.

VOGEL, T.; SEIBEL, A.; GIESE, H. The role of models and megamodels at runtime. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 6627 LNCS, p. 224–238, 2011. ISSN 03029743.

VOLTER, M.; SCHMID, A.; WOLFF, E. *Server Component Patterns: Component Infrastructures Illustrated with EJB*. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0470843195.

WADA, H.; SUZUKI, J.; OBA, K. Modeling turnpike: A model-driven framework for domain-specific software development. In: *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. New York, NY, USA: ACM, 2005. (OOPSLA '05), p. 128–129. ISBN 1-59593-193-7. Disponível em: <http://doi.acm.org/10.1145/1094855.1094897>.

WAIGNIER, G.; PRAWEE, S.; MEUR, A.-F. L.; DUCHIEN, L. A Framework for Bridging the Gap Between Design and Runtime Debugging of Component-Based Applications. In: *3rd International Workshop on Models@runtime*. Toulouse, France: [s.n.], 2008. Disponível em: <https://hal.inria.fr/inria-00321598>.

WALLS, C. *Modular Java: Creating Flexible Applications with OSGi and Spring*. Raleigh, NC: Pragmatic Bookshelf, 2009. ISBN 978-1-93435-640-1.

WEYNS, D.; MALEK, S.; ANDERSSON, J. Forms:unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, v. 7, n. 1, p. No. 8, 2012. ISSN 15564665. Disponível em: <http://dl.acm.org/citation.cfm?id=2168260.2168268>.

YODER, J. W.; JOHNSON, R. The adaptive object-model architectural style. Springer US, Boston, MA, p. 3–27, 2002.