DANIEL HERÁCLIO OLSEN MAIA DO CARMO

GERÊNCIA DE CONFIGURAÇÃO EM TEMPO DE EXECUÇÃO PARA SISTEMAS AUTOADAPTÁVEIS

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Engenharia de Software.

Orientadores: Prof. Dr. Leonardo Gresta Paulino Murta
Prof. Dr. Orlando Gomes Loques Filho

Niterói
2013

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

C287 Carmo, Daniel Heráclio Olsen Maia do
Gerência de configuração em tempo de execução para sistemas autoadaptáveis / Daniel Heráclio Olsen Maia do Carmo. – Niterói, RJ : [s.n.], 2013.
132 f.

Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2013.
Orientadores: Leonardo Gresta Paulino Murta, Orlando Gomes Loques Filho.

1. Engenharia de software. 2. Sistema autoadaptável. 3. Gerenciamento de configuração de software. I. Título.

CDD 005.1

DANIEL HERÁCLIO OLSEN MAIA DO CARMO

GERÊNCIA DE CONFIGURAÇÃO EM TEMPO DE EXECUÇÃO PARA SISTEMAS AUTOADAPTÁVEIS

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Engenharia de Software.

Aprovada em junho de 2013.

BANCA EXAMINADORA

Prof. Dr. Leonardo Gresta Paulino Murta – Orientador

UFF

Prof. Dr. Orlando Gomes Loques Filho – Orientador

UFF

Prof. Dra. Viviane Torres da Silva

UFF

Prof. Dra. Flávia Coimbra Delicato

UFRJ

Niterói

2013

À minha Família e aos meus orientadores Leonardo Gresta Paulino Murta e Orlando Loques.

Homenagem póstuma à minha querida sogra, Maria Ilca Rodrigues, minha amada avó Ivonilde Maia do Carmo e avô Antônio Maia Pereira.

AGRADECIMENTOS

À minha família pelo apoio e compreensão durante o período dedicado a este trabalho. Em especial a minha esposa Pabla Porto, por sua dedicação e amor, sem os quais eu não poderia viver. Principalmente, agradeço a meu filho Pedro Heráclio, que me motiva a realizar o impossível com um simples sorriso. Agradeço a meus pais, Francisco Heráclio e Maria Inês, cujo apoio incondicional e exemplo de vida foram fundamentais para a minha. Agradeço à minha irmã, Bianca Olsen, por sua ajuda em diversos momentos importantes. Agradeço ao meu avô Antônio Heráclio, por ser o melhor avô que um neto poderia desejar. Agradeço à minha tia Maria Helena, por demonstrar como é possível ter forte caráter e um coração justo. Agradeço à minha tia Marta Maria, por demonstrar a importância de tratar a todos com respeito e agir com docilidade sempre.

Aos meus orientadores Leonardo Gresta Paulino Murta e Orlando Gomes Loques Filho, que me incentivaram e ensinaram como tornar realidade esta pesquisa. O caráter e dedicação de ambos continuarão a me inspirar por toda minha vida.

Aos membros da banca por disporem de seu tempo precioso para realizar a avaliação deste trabalho, principalmente as professoras Flávia Coimbra Delicato e Viviane Torres da Silva.

Ao professor Antônio Tadeu Azevedo Gomes que, mediante participação em projeto de pesquisa paralelo a este trabalho, forneceu apoio financeiro necessário.

Aos meus amigos e colegas de mestrado que me apoiaram ao longo da pós-graduação. Em especial os colegas do GEMS e do Laboratório Tempo, que participaram ativamente de discussões a respeito deste trabalho. Principalmente, agradeço ao amigo Sérgio Teixeira de Carvalho que me orientou em diversos momentos desta pesquisa. Agradeço também ao meu amigo Marcos Vinicius Policarpo Cortês, por sempre estar disposto a ajudar na solução de problemas encontrados durante o desenvolvimento dos protótipos.

A todos os professores e funcionários do Instituto de Computação, com quem tive contato direto ou indireto. O profissionalismo de todos foi fundamental para o andamento deste trabalho.

A CAPES, FAPERJ e CNPQ, pelo apoio financeiro ao longo da realização deste trabalho.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.”

Albert Einstein

RESUMO

Sistemas Autoadaptáveis (SA) são capazes de se modificarem quando mudanças ocorrem em seu contexto de execução, reajustando sua configuração arquitetural para suprir requisitos afetados. Aplicações envolvendo SA estão em ascensão, sendo alvos frequentes de pesquisa por diferentes campos. Entretanto, SA enfrentam desafios significativos relativos a questões distintas. Por exemplo, SA devem lidar com incertezas, ter confiabilidade, permitir verificações, validações e prover garantia sobre suas adaptações. Estes e outros desafios tem origem no comportamento dinâmico dos SA. A evolução deste comportamento pode ser compreendida por meio de um conjunto de questões de interesse de adaptação (i.e., por que, o que, quando, onde, quem, e como). Além disso, estas questões são pertinentes durante as fases de desenvolvimento, operação e pós operação. Logo, a disponibilidade de ferramentas para monitoramento e auditoria é do interesse da comunidade envolvida com SA. Entretanto, soluções atuais têm limitações importantes em sua aplicabilidade. Por exemplo, falta-lhes a capacidade de registrar a informação necessária para responder aos interesses de adaptação, demandando processos extensivos de pré-análise. Este trabalho propõe a aplicação de Gerência de Configuração em tempo de execução para prover meios de monitorar e auditar SA. A abordagem proposta é chamada CM@RT, e é composta de dois módulos complementares. O primeiro registra a evolução de diferentes aspectos do processo de adaptação em tempo de execução. O segundo provê ferramentas de visualização voltadas ao estudo dos dados registrados. Para avaliar a abordagem, cenários extraídos de um sistema pertencente a uma Linha de Produtos de Software Dinâmico chamado SCIADS foram submetidos a estudo por meio da abordagem CM@RT. Os resultados demonstram que a abordagem é capaz de fornecer meios para responder aos interesses de adaptação durante atividades de monitoramento e auditoria. Portanto, fornecendo evidência de benefícios significativos ao confronto de desafios de SA. Entre as contribuições deste trabalho estão: (1) uma alternativa ao uso de mecanismos de log, para registro da evolução de SA em tempo de execução, (2) modelagem capaz de responder perguntas baseadas em questões de interesse de SA, (3) modelagem permite a extração de métricas sobre a evolução dinâmica de SA e sua configuração arquitetural, (4) algoritmo para determinação de diferenças entre configurações arquiteturais e (5) algoritmo para determinação de diferenças entre contextos de execução.
Palavras-chave: Sistemas Autoadaptáveis; Gerência de Configuração; Monitoramento; Auditoria; Linhas de Produto de Software Dinâmico.

ABSTRACT

Self-Adaptive Systems (SAS) are capable of modifying themselves when changes occur on runtime context, adjusting their architectural configuration to cope with affected requirements. SAS based applications are on the rise, been targeted by constant research from several fields. However, SAS face significant challenges related to distinct matters. For example, SAS should deal with uncertainty, achieve dependability, allow verifications, validations and provide trust over adaptations. These and other challenges came from SAS dynamic behavior nature. The behavioral evolution can be understood through a set of adaptation concerns (i.e., why, what, when, where, who, and how). In addition, these concerns are useful during the phases of development, operation e post operation. Thus, the availability of tool for monitoring and auditing are of interest for SAS community. However, current solutions have important application limitations. For example, they lack the capability of recording SAS behavior with required information for adaptation concerns analysis, demanding extensive preprocessing to produce them. This work proposes the use of Software Configuration Management at runtime to provide means for monitoring and auditing SAS. The approach is called CM@RT, and is composed of two modules. The first register the evolution of different aspects of the adaptation process during runtime. The second provides visualization tools for studying registered data. To evaluate the approach, runtime scenarios from a Dynamic Software Product Line SAS called SCIADS were studied with CM@RT support. The results provide evidence that the approach is supplies means for answering adaptation concerns during monitoring and auditing activities. Thus, supplies evidence of significant benefits for tackling SAS challenges. Among the contributions of this work are: (1) an alternative to log mechanism for registering SAS runtime evolution, (2) modeling capable of answering SAS concerns based questions, (3) modeling which allows extraction of metrics over SAS behavior and architecture configuration, (4) algorithm for diffing between architecture configurations and (5) algorithm for diffing between runtime contexts.

Keywords: Self-Adaptive Systems; Configuration Management; Monitoring; Auditing; Dynamic Software Product Lines.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

5W+1H – *Why*, *when*, *who*, *where*, *what* e *how*

CM@RT – *Configuration Management at Runtime*

GC – Gerência de Configuração

LPS – Linha de Produto de Software

LPSD – Linha de Produto de Software Dinâmico

OSGi – *Open Service Gateway initiative*

SA – Sistema Autoadaptável

SCIADS - Sistema Computacional Inteligente de Assistência Domiciliar à Saúde

SCV – Sistema de Controle de Versão

SCM – Sistema de Controle de Mudança

SD – Sistema Dinâmico

SUMÁRIO

CAPÍTULO 1 –INTRODUÇÃO

1.1 PROBLEMA E MOTIVAÇÃO

Sistemas Autoadaptáveis (SA) modificam seu comportamento em reação a mudanças em seu contexto de execução, realizando alterações em sua configuração arquitetural (CHENG *et al.*, 2009; SALEHIE; TAHVILDARI, 2009). Neste caso contexto de execução representa o que o SA é capaz de perceber através de seus sensores e fontes auxiliares de informação, como, por exemplo, modelos internos de sua configuração arquitetural. Autoadaptação pode ser vista sob diferentes pontos de vista, inclusive como uma forma de evolução de software (SALEHIE; TAHVILDARI, 2009) e atualmente, existe uma demanda crescente por SA (PARASHAR; HARIRI, 2005; BRUN *et al.*, 2009; SALEHIE; TAHVILDARI, 2009). Por exemplo, *smartphones* e *tablets* oferecem um universo de aplicativos que precisam se adaptar ao contexto de execução dinâmico imposto pela mobilidade. Além disso, sistemas críticos também têm interesse nos benefícios de SA. No contexto de sistemas de apoio à saúde, a vida de um paciente não deve ser ameaçada sob qualquer circunstância, como, por exemplo, por falha em um de seus componentes. SA são capazes de solucionar estas e outras situações de risco para o paciente sem a necessidade de intervenção humana.

No entanto, SA enfrentam uma gama de desafios (CHENG *et al.*, 2009; SALEHIE; TAHVILDARI, 2009; TAMURA *et al.*, 2013) diretamente relacionados à sua natureza dinâmica. Estes desafios incluem confiabilidade, garantia (do inglês, *assurance*), verificação e validação, modelagem, engenharia, etc. Enfrentá-los demanda meios para estudar a evolução apresentada por SA ao longo de sua operação. Responder a questões de interesse de adaptação permite estudar a evolução dinâmica de SA (CHENG *et al.*, 2009; SALEHIE; TAHVILDARI, 2009). Estas questões são baseadas em seis perguntas conhecidas como 5W+1H (do inglês, *why*, *when*, w*ho*, *what*, *where* e *how*) (KIPLING, 1902). Respostas a elas são necessárias não só durante a operação dos SA, como após a mesma. Auditoria é obrigatória em sistemas críticos (PARNAS, 1994), como, por exemplo, para aprovação de sistemas médicos (MCCAFFERY; CASEY; MCHUGH, 2011). Logo, fornecer meios para registro do comportamento para realização de auditoria também é do interesse da área de SA.

Atualmente, existe um número significativamente pequeno de soluções voltadas ao registro e posterior estudo da evolução dinâmica de SA. A pesquisa por trabalhos relacionados revelou abordagens que registram apenas a evolução da configuração

arquitetural de SA utilizando técnicas de Gerência de Configuração (GC). Além disso, existe a possibilidade de se utilizar ferramentas de log para registrar as ações do SA em tempo de execução. Neste contexto, a análise do comportamento de SA precisa combinar o registro da evolução da configuração arquitetural com o log para obter as informações necessárias ao esclarecimento de questões de interesse de SA. Por exemplo, o que mudou está registrado na configuração arquitetural, mas o porquê e quando está registrado no log. Relacionar ambas é uma tarefa complexa e propensa a erros, visto que não existe integração entre as fontes de informação.

1.2 OBJETIVOS

Partindo da demanda por ferramentas adequadas a realização de estudos sobre a evolução dinâmica de SA, este trabalho tem o objetivo principal de fornecer uma solução capaz de responder a questões de interesse sobre adaptação baseadas nas 5W+1H.. Em complementação ao objetivo principal, existem os seguintes objetivos secundários:

1) Registrar a evolução dinâmica de SA.
2) Fornecer ferramentas de visualização que favoreçam a obtenção de respostas a questões baseadas nas 5W+1H.
3) Permitir monitoramento e auditoria da evolução do SA.

A abordagem proposta se chama CM@RT (do inglês, *Configuration Management at Runtime*). Para atingir os objetivos descritos anteriormente ela realiza dois conjuntos de atividades complementares em aplicações distintas. A primeira aplicação é responsável por registrar a evolução do SA, armazenando a evolução do contexto de execução, motivações para adaptação e modificações na configuração arquitetural. A segunda aplicação é responsável por fornecer funcionalidades de monitoramento e auditoria, oferecendo ferramentas de visualização projetadas em função da semântica das informações representadas. Para avaliar o CM@RT, foram consideradas as seguintes questões de pesquisa:

1) A abordagem é capaz de registrar as informações necessárias ao estudo da evolução dinâmica de um SA?
2) A abordagem é capaz de fornecer respostas à questões de interesse de adaptação baseadas nas 5W+1H??

A avaliação das questões de pesquisa e dos objetivos da abordagem foi realizada por meio de demonstração das funcionalidades sobre cenários extraídos da literatura em um protótipo real de sistema crítico. Ao longo da demonstração, são descritas alternativas para obtenção de respostas a uma variedade de questões de interesse relativas a evolução dinâmica

de SA. A demonstração forneceu indícios da capacidade da abordagem CM@RT em atingir os objetivos traçados. Além disso, foi possível postular questões sobre aspectos específicos do sistema alvo da demonstração.

1.3 ORGANIZAÇÃO

O restante do trabalho está organizado em cinco capítulos. O Capítulo 2 apresenta uma fundamentação geral sobre SA, incluindo suas motivações, propriedades, ferramental de apoio disponível e SA na atualidade. Ademais, discute a questão da evolução dinâmica de SA e aponta a importância de estudar o comportamento em tempo de execução destes sistemas. Além disso, são apresentados trabalhos que aplicam diferentes técnicas e que possibilitam o estudo da evolução dinâmica de SA durante atividades de monitoramento e auditoria. Na parte final são discutidas razões para ampliar a aplicação de técnicas de GC no apoio ao estudo da evolução dinâmica de SA, tanto durante sua execução como após a mesma.

O Capítulo 3 descreve a abordagem CM@RT, que permite o registro e visualização da evolução de SA. Nesse capítulo são apresentados os dois módulos que compõem a abordagem: o CM@RT-Repository e o CM@RT-Visualizer. O primeiro é destinado ao registro de informações que caracterizam a evolução dinâmica de um SA. O segundo é destinado ao estudo destas informações registradas, empregando representações gráficas e métricas de qualidade.

O Capítulo 4 descreve os protótipos da abordagem. São apresentados seus requisitos e arquiteturas. É descrito o funcionamento do CM@RT-Repository e como usá-lo para registro da evolução dinâmica. Também é descrita a utilização do CM@RT-Visualizer, para a realização de estudos sobre as informações disponíveis.

O Capítulo 5 descreve como foram avaliados os benefícios da abordagem CM@RT. A avaliação demandou a construção de protótipos adicionais, a fim de submeter a abordagem a cenários extraídos da literatura relativos a SA críticos. Para estes cenários é demonstrado como obter respostas a questões de interesse.

O Capítulo 6 expõe conclusões obtidas em função da pesquisa realizada. Além disso, são discutidas limitações existentes e contribuições dadas. Ademais, são relatadas oportunidades de trabalhos futuros relevantes.

CAPÍTULO 2 – SISTEMAS AUTOADAPTÁVEIS

2.1 INTRODUÇÃO

O comportamento dos SA evolui dinamicamente ao longo de seu período de operação. Esta evolução dinâmica é fruto de adaptações realizadas de acordo com propriedades e características particulares dos SA. Algumas destas características foram coletadas e descritas em diferentes pesquisas (CHENG *et al.*, 2009; SALEHIE; TAHVILDARI, 2009). Paralelamente, o comportamento de um SA é definido por meio de seu ciclo de adaptação (HORN, 2001). Além destas, a escolha das tecnologias empregadas na construção da abordagem planejada afeta diretamente a complexidade da solução resultante. Por exemplo, *frameworks* para SA reduzem a complexidade da solução ao prover serviços reutilizáveis. No entanto, SA da atualidade ainda não possuem tendências evidentes quanto à combinação destes elementos aqui citados. O fator comum a todos os SA é sua natureza dinâmica e os desafios que ela impõe na busca de diferentes atributos de qualidade, requisitos funcionais e não funcionais.

O restante deste capítulo aprofunda a discussão realizada nesta seção introdutória, da seguinte forma. Na seção 2.2 são explicados conceitos básicos sobre SA. A seção 2.3 descreve ciclos de adaptação, que regem o comportamento de SA. A seção 2.4 descreve o suporte existente para a construção de SA. Na seção 2.5 são discutidos alguns pontos em comum entre SA da atualidade e é descrito o sistema SCIADS de forma breve. A seção 2.6 aborda a evolução dinâmica de SA. A seção 2.7 descreve trabalhos relacionados encontrados na literatura por meio de mecanismos de busca disponíveis na internet. Finalmente, a seção 2.7 apresenta as considerações finais deste capítulo.

2.2 CONCEITOS BÁSICOS

Antes de surgirem os SA, existiam sistemas capazes de suportar adaptações determinadas por intervenção humana em tempo de execução, aqui chamados Sistemas Dinâmicos (SD). Por exemplo, o sistema operacional Linux suporta adaptações em sua configuração arquitetural por meio de gerenciadores de pacotes (TUCKER *et al.*, 2007). Logo, um SA pode ser visto como um SD capaz de se adaptar de forma autônoma.

SA possuem propriedades que determinam sua capacidade de autoadaptação frente a diversas situações, chamadas de propriedades *Self-** em inglês (SALEHIE; TAHVILDARI,

2009). Essas propriedades foram descritas e organizadas hierarquicamente em SALEHIE e TAHVILDARI (2009), como visto na Figura 1.

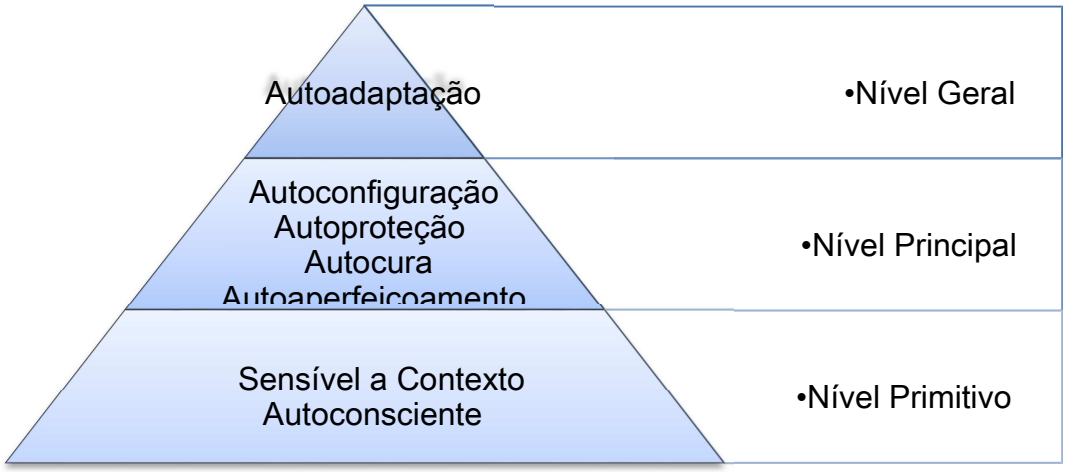


Figura 1 Hierarquia de propriedades de autoadaptação. Adaptado de SALEHIE e TAHVILDARI (2009)

Os níveis definidos na hierarquia ilustrada na Figura 1 podem ser descritos da seguinte forma (SALEHIE; TAHVILDARI, 2009):

Geral: abrangem o comportamento do sistema como um todo. Exemplo: autogovernança, autogerenciamento, autocontrole e auto-organização.

Principal: são as que se assemelham a características biológicas. Exemplo: autoconfiguração, autoproteção, autocura e autoaperfeiçoamento.

Primitivo: são as que dão suporte às demais propriedades. Exemplo: autoconsciência, automonitoramento e sensibilidade ao contexto.

São alvo deste trabalho, SA que apresentam as propriedades de autoconfiguração e sensibilidade ao contexto. Autoconfiguração é a propriedade de alterar a configuração da arquitetura do sistema dinamicamente frente a mudanças, seja removendo, adicionando, alterando ou substituindo entidades. Sensibilidade ao contexto é a capacidade de perceber mudanças no ambiente de execução, significativas o suficiente para causarem impactos sobre os requisitos de funcionamento do SA. Logo, estas propriedades estão diretamente relacionadas à construção de SA que apresentem as demais propriedades. Por exemplo, autoaperfeiçoamento implica em avaliar o contexto de execução e modificar a configuração em busca de melhor desempenho.

Um SA é desenvolvido para realizar adaptações motivadas por suas propriedades, que são selecionadas entre as descritas anteriormente. Entretanto, um SA também está sujeito a outros cenários onde sofre modificações por outras razões. Entre as motivações possíveis estão falhas internas de componentes, manutenção, testes, customização, entre outras. Em GC,

o termo *issue* é utilizado em Sistemas de Controle de Mudança para representar solicitações de modificação feitas sobre as mais diversas razões. A expressividade deste termo não pôde ser encontrada na língua portuguesa. Em função disso, o termo *issue* é utilizado neste trabalho para se referir a adaptações e modificações ocorridas no SA ao longo de sua operação.

SA possuem características, além de propriedades. A especificação destas características determina como o sistema realizará autoadaptações. Em SALEHIE e TAHVILDARI (2009) é definida uma taxonomia capaz de caracterizar amplamente SA, vista na Figura 2. Esta taxonomia é importante para caracterizar o mecanismo de autoadaptação a ser utilizado em um SA que se pretende construir e será descrita a seguir.

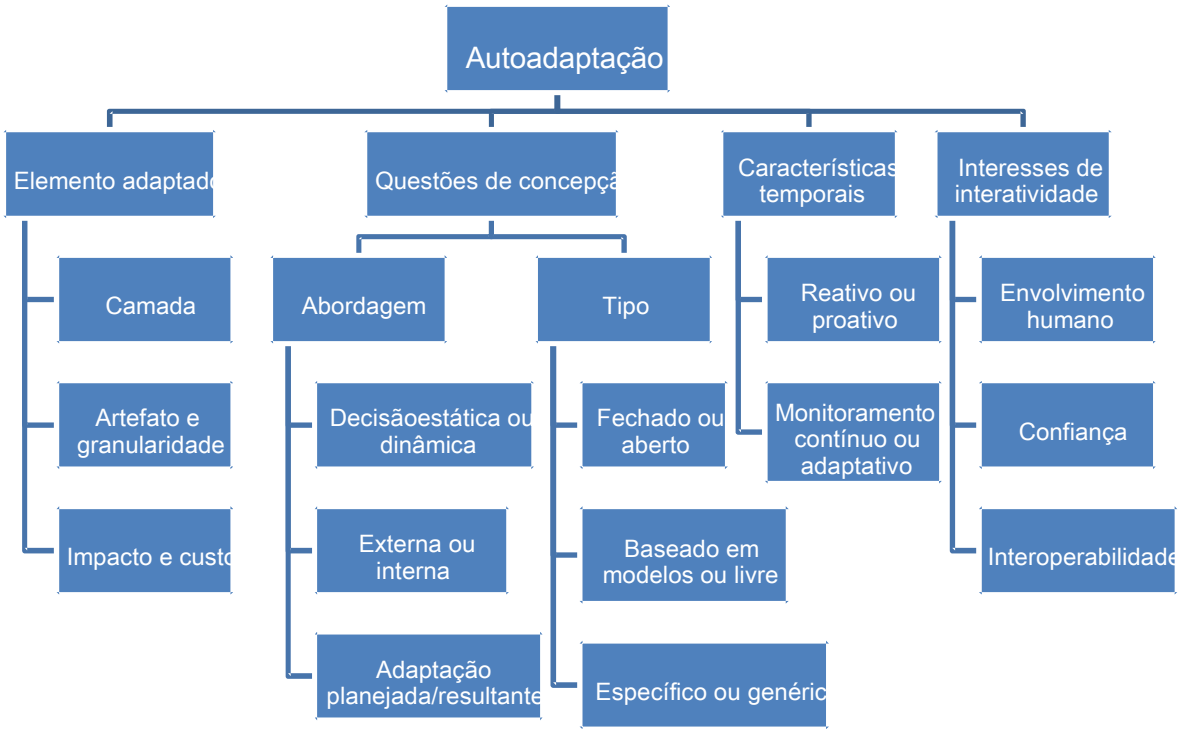


Figura 2 Taxonomia de autoadaptação, adaptada de SALEHIE e TAHVILDARI (2009)

Em **elemento adaptado**, a taxonomia concentra as características do que efetivamente é modificado durante as adaptações. A característica **camada** define o **elemento adaptado** entre as diferentes partes da configuração arquitetural do SA. Por exemplo, pode ser necessário adaptar a camada de comunicação do SA. A característica **artefato e granularidade** define o **elemento adaptado** em termos de suas possíveis decomposições. Por exemplo, um SA pode ser decomposto em módulos, serviços, pacotes, componentes. A característica **impacto e custo** define dois tipos de adaptação com base em suas consequências e complexidade, ditas fracas e fortes. Adaptações do tipo fraca se referem a alterações em parâmetros de funcionamento ou a adaptações de baixo custo e complexidade, como ajustar a velocidade de download por exemplo. Adaptações ditas fortes abrangem

modificações que afetam a composição do sistema, alterando sua configuração arquitetural, como, por exemplo, adicionando e removendo componentes.

Em **questões de concepção** estão inseridas as características da **abordagem** adotada e **tipo** de adaptação realizada pelo SA. No item **abordagem** são caracterizados três elementos da solução adotada, os quais são descritos a seguir. O mecanismo de decisão sobre as adaptações necessárias pode ser **estático** (i.e., parte do código fonte) ou **dinâmico** (definido externamente ao código fonte). Infraestruturas para autoadaptação podem ser **externas** (SA composto de SD mais infraestrutura) ou **internas** (SD e infraestrutura estão integrados). As adaptações podem ser **planejadas** durante o desenvolvimento ou **resultantes** de aprendizado por meio de inteligência artificial. No item **tipo** de adaptação também são caracterizados três elementos, a saber: fechado ou aberto, baseado em modelos ou livre e específico ou genérico. SA do tipo **fechado** realizam um conjunto predeterminado de ações de adaptação, como, por exemplo, remover um componente da configuração arquitetural. SA do tipo **aberto** podem ser estendidos, como, por exemplo, atualizando a infraestrutura de um SA com **abordagem externa**. Adaptações **baseadas em modelos** são realizadas de acordo com representações do contexto de execução e do próprio SA, como, por exemplo, configurações arquiteturais. SA **livres** de modelos utilizam requisitos, objetivos e alternativas conhecidas para guiar as adaptações, como, por exemplo, com uso de aprendizado de máquina. Soluções **específicas** são construídas para fins e domínios particulares, enquanto **genéricas** atendem a diferentes objetivos, desde que sejam configuradas de acordo com o domínio de aplicação.

Características temporais estão características que determinam o momento da tomada de decisão sobre as adaptações necessárias. SA **reativos** aguardam mudanças significativas em seu contexto de execução antes de se adaptar. SA **proativos** procuram antecipar a ocorrência destas mudanças no contexto de execução. Ambos dependem de sua forma de monitoramento do contexto de execução, que pode ser **contínuo** ou **adaptativo**. O monitoramento **contínuo** observa constantemente todo o escopo de contexto de execução. No monitoramento **adaptativo**, o SA observa um conjunto mínimo do contexto de execução e busca mais informações apenas quando surge uma anomalia significativa neste subconjunto.

Em **interesses de interatividade** aborda interações com outros SA, sistemas ou humanos, utilizando interfaces. O **envolvimento humano** cobre tanto a relação entre usuário e SA, quanto administrador e SA. **Confiança** (do inglês, *trust*) descreve a confiabilidade esperada do SA, como, por exemplo, que garantias podem ser fornecidas a terceiros sobre o funcionamento do sistema. **Interoperabilidade** define a extensão e nível de interação entre as

partes que compõem um SA, como, por exemplo, sistemas distribuídos podem apresentar adaptações de extensão global em seus componentes.

Esta taxonomia está diretamente relacionada a compreensão do comportamento exibido por um SA, segundo a pesquisa feita em SALEHIE e TAHVILDARI (2009). A pesquisa propõe o uso das seguintes perguntas, conhecidas como 5W+1H, para obter esta compreensão:

Onde? Questões sobre a localização de problemas ou demandas no SA, i.e., qual a camada deve ser adaptada? Remete a **elemento adaptado**, item **camada**.

Quando? Questões sobre aspectos temporais do SA, i.e., quando adaptar ou com que frequência? Remete a **características temporais.**

O que? Questões sobre a estratégia de adaptação, i.e., realizar uma adaptação fraca ou forte? Remete a **elemento adaptado**.

Por quê? Questões sobre a motivação da adaptação, i.e., qual o objetivo da adaptação? Remete a **questões de concepção.**

Quem? Questões sobre quem solicitou a adaptação, i.e., solicitado automaticamente ou por intervenção humana? Remete a **interesses de interatividade.**

Como? Questões sobre como efetivar as adaptações, i.e., em que ordem realizar cada modificação solicitada? Remete a **questões de concepção.**

Respostas a estas perguntas são necessárias, não só durante o desenvolvimento de SA, mas ao longo de sua operação (SALEHIE; TAHVILDARI, 2009). Em um SA, a avaliação destas perguntas pode ser feita por um humano, que determinaria as adaptações necessárias. Entretanto, utilizar intervenções humanas para realizar adaptações em sistemas complexos é uma tarefa propensa a falhas. Para eliminar a necessidade de intervenção humana podem ser utilizados ciclos de adaptação. Ciclos de adaptação são alternativas adequadas a modelagem do processo de autoadaptação (HORN, 2001; KEPHART; CHESS, 2003; BRUN *et al.*, 2009; TAMURA *et al.*, 2013). A seção 2.3 fornece mais informações sobre o ciclo de adaptação.

2.3 CICLO DE ADAPTAÇÃO

O ciclo de adaptação é um dos mais importantes elementos de um SA (BRUN *et al.*, 2009). Sendo alvo constante de pesquisa, tanto como objeto (HORN, 2001; KEPHART; CHESS, 2003; BRUN *et al.*, 2009; TAMURA *et al.*, 2013), como parte dela sobre SA (GARLAN; SCHMERL; CHENG, 2009; SALEHIE; TAHVILDARI, 2009; ESFAHANI; MALEK, 2012). Ciclos de adaptação também são chamados de ciclos de controle automático

(DOBSON *et al.*, 2006) ou gerenciadores de adaptação (SALEHIE; TAHVILDARI, 2009), sendo mais usual a primeira denominação. Ciclos de adaptação consistem em ciclos fechados com retroalimentação, onde mudanças monitoradas desencadeiam tomadas de decisão sobre adaptações necessárias no sistema (GARLAN; SCHMERL; CHENG, 2009).

Entre os ciclos de adaptação mais importantes está o MAPE-K (KEPHART; CHESS, 2003). Sua arquitetura está representada na Figura 3, juntamente com a integração do ciclo com um SD. O ciclo MAPE-K é composto de quatro fases distintas (MAPE): monitorar, analisar, planejar e executar. Além delas, existe uma base de conhecimento (K) para compartilhamento de dados entre as fases do ciclo (SALEHIE; TAHVILDARI, 2009). A integração entre MAPE-K e SD ocorre por meio de sensores e atuadores, como pode ser visto na Figura 3. Sensores são monitorados em busca de mudanças significativas e atuadores realizam as adaptações no SD. O conjunto formado pelos elementos descritos forma um SA completo.

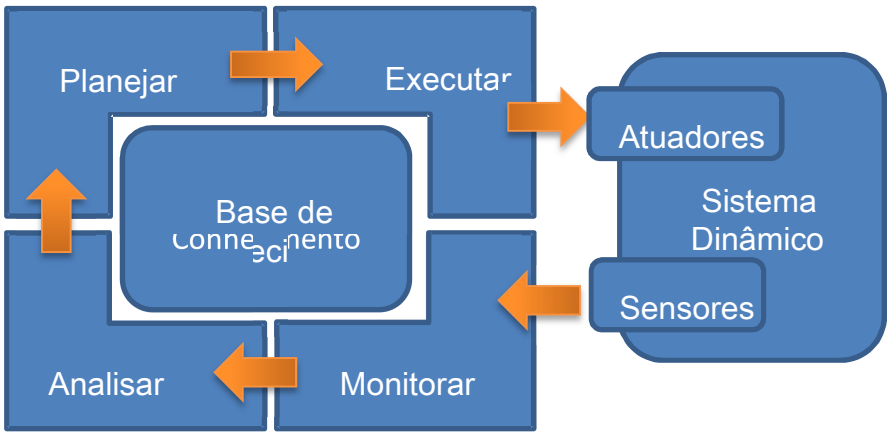


Figura 3 Ciclo de controle automático. Adaptado de KEPHART e CHESS (2003).

Os cinco elementos que compõem o ciclo MAPE-K são descritos nas cinco subseções a seguir. A saber, a ordem das seções é monitoramento, análise, planejamento, execução e base conhecimento.

2.3.1 MONITORAR

A fase de monitoramento é responsável por manter o contexto de execução atualizado. Em relação aos interesses de adaptação, a fase de monitoramento contribui para três questões baseadas nas 5W+1H. O monitoramento é capaz de esclarecer **quando** a adaptação se tornou necessária, uma vez que fornece o **porquê** nos dados coletados. Além disso, fornece indícios sobre **o que** precisa ser adaptado, a partir da fonte das mudanças no contexto de execução (i.e. os sensores) (SALEHIE; TAHVILDARI, 2009).

Para manter o contexto de execução atualizado são realizadas diferentes tarefas durante esta fase. Entre elas está a interação com os sensores e pré-processamento dos dados coletados (SALEHIE; TAHVILDARI, 2009).

Sensores são verificados periodicamente para atualização do contexto de execução. A frequência desta coleta pode variar de acordo com os objetivos do SA. Além disso, sensores possuem características próprias, como, por exemplo, interfaces próprias, precisão dos dados coletados e estabilidade. Portanto, é importante analisar o comportamento destes componentes, conforme o SA opera a fim de determinar sua qualidade.

Os dados coletados destes sensores podem ser submetidos a diferentes técnicas de pré-processamento. Dentre essas técnicas estão análise estatística e mineração de dados (SALEHIE; TAHVILDARI, 2009). Além disso, como citado na seção 2.2, a própria coleta pode ser ajustada. SA podem monitorar um conjunto reduzido de variáveis de contexto. OS dados coletados podem ser ampliados, conforme seja percebida uma demanda clara.

2.3.2 ANALISAR

A fase de análise é responsável por submeter o contexto de execução a um mecanismo de decisão, capaz de inferir adaptações necessárias na configuração arquitetural. Em relação aos interesses de adaptação, a fase de análise contribui para três questões baseadas nas 5W+1H. A análise fornece o **porquê** da adaptação segundo o entendimento do mecanismo de decisão. As adaptações solicitadas determinam **onde** está o problema e parcialmente determinam **o que** será adaptado. O **o que** é parcialmente determinado pois apenas na execução são solucionadas eventuais pendências nas solicitações, como, por exemplo, dependências de componentes.

O mecanismo de decisão faz parte da base do processo de adaptação de um sistema autoadaptável (SALEHIE; TAHVILDARI, 2009). O mecanismo de decisão é responsável por inferir quais ações são necessárias à satisfação dos requisitos do sistema em tempo de execução. As alternativas disponíveis para inferência das adaptações necessárias são variadas, como aquelas baseadas em modelos (BENCOMO, N. *et al.*, 2008; BENCOMO, NELLY, 2009; BLAIR; BENCOMO; FRANCE, 2009; MORIN *et al.*, 2009) ou em contratos de adaptação (equivalentes a regras, estratégias e políticas) (GARLAN *et al.*, 2004; SALEHIE; TAHVILDARI, 2009; ROSENMÜLLER *et al.*, 2011; CARVALHO; MURTA; LOQUES, 2012).

Em BENCOMO (2009), a autora argumenta que SA poderiam se beneficiar da utilização de modelos em tempo de execução (do inglês, *Models@Runtime*). Entre estes

benefícios está a construção de mecanismos de decisão baseados em técnicas de MDE (do inglês, *Model Driven Engineering*). Um exemplo deste caso é abordagem DIVA, que utiliza modelos extensivamente (MORIN *et al.*, 2009).

Os contratos de adaptação permitem a realização de análise formal (BRAGA; CHALUB; SZTAJNBERG, 2009). Além disso, contratos de adaptação têm sido aplicados com sucesso para determinar adaptações em sistemas críticos (CARVALHO; MURTA; LOQUES, 2012). Cada contrato de adaptação é regido por condições internas baseadas no contexto de execução, que determinam as adaptações necessárias em diferentes níveis de abstração (ex.: serviços, componentes, parâmetros). Por exemplo, em ROSENMÜLLER *et al.*, (2011), regras podem ser constituídas por eventos disparadores, restrições, ações adaptativas e manipulação de requisitos.

Independente da alternativa escolhida, a natureza dinâmica de SA pode ocasionar adaptações imprevistas, afetando sua confiabilidade (GEORGAS; VAN DER HOEK; TAYLOR, 2005). Por exemplo, contratos distintos podem entrar em conflito, ao atuarem sobre um mesmo grupo de componentes e criarem instabilidade na configuração arquitetural do sistema. Existem ainda outras duas possibilidades, a ocorrência de adaptações desnecessárias (falsos positivos) ou não realização de adaptações necessárias (falsos negativos). Falsos positivos requerem análise das transições entre configurações arquiteturais realizadas. Acessando o *issue* informado para a transição e confrontando-o com o respectivo contexto de execução é possível confirmar ou não a necessidade de adaptação. Falsos negativos requerem a análise direta dos contextos de execução registrados. Nestes é possível identificar variáveis de contextos fora de limites estipulados pelas especificações do SA, em relação a uma determinada configuração arquitetural.

2.3.3 PLANEJAR

A fase de planejamento é responsável, principalmente, por determinar o momento oportuno para a efetivação da adaptação inferida durante a análise. Além disso, nesta fase pode ser realizado o planejamento de como serão realizadas as modificações necessárias. Em relação aos interesses de adaptação, a fase de planejamento contribui para duas questões baseadas nas 5W+1H. O planejamento fornece **quando** a adaptação deverá ser realizada e se será realizada. Também pode informar **como** ela será concretizada. Existe também a possibilidade de adaptações serem descartadas (por incompatibilidade decorrente de atraso na execução) ou planejadas novamente (por erros durante a fase de execução).

Entretanto, a fase na qual os interesses podem ser esclarecidos pode diferir em função das decisões tomadas durante o desenvolvimento do SA. Por exemplo, a fase de planejamento pode não ser responsável por determinar **como** a adaptação será realizada. Esta decisão pode ser motivada pela complexidade da tarefa.

Ao mesmo tempo em que adaptações arquiteturais provocam efeitos mais significativos no sistema, também têm maior potencial para causar instabilidade sobre o mesmo. Planejar adaptações arquiteturais é um processo que deve ser realizado considerando cuidadosamente seus efeitos. O próprio processo de adaptação da configuração arquitetural pode ser danoso ao sistema, uma vez que os efeitos de sua realização podem não ser imediatos. Por exemplo, durante a substituição de um componente por outro existe a possibilidade do novo apresentar falha e o anterior não ser restaurado com sucesso, deixando o SA sem os serviços dos mesmos até que seja feita a restauração do componente original.

2.3.4 EXECUTAR

A fase de execução é responsável por concretizar a adaptação planejada pela fase anterior. Para isso, atuadores ligados ao SA concretizam as modificações especificadas na adaptação. Em relação aos interesses de adaptação, a fase de execução contribui para três questões, fornecendo **como**, **o que** e **quando**. Nesta fase, sabe-se a ordem de execução das alterações, as dependências entre componentes e o exato instante da adaptação em si. Por exemplo, é durante esta fase que dependências de baixo nível são solucionadas, como aquelas entre componentes da configuração arquitetural do SA.

Adaptações ditas fortes alteram a composição e topologia da configuração arquitetural de um SA. Mudanças na topologia são aquelas que alteram as conexões entre componentes, por exemplo, alternar entre componentes responsáveis por um mesmo serviço. Alterações na composição da configuração arquitetural servem a diversos propósitos, como, por exemplo, atualizações por evolução dos componentes, customização ou ampliação do sistema. Ademais, propriedades de SA tais como, autoaperfeiçoamento, autocura, autoproteção, podem demandar alterações na composição do sistema. Quando estas adaptações ocorrem, diz-se que a configuração arquitetural sofreu evolução, de forma semelhante à evolução sofrida pelo código fonte (VAN DER HOEK *et al.*, 2001).

A evolução da configuração arquitetural é um processo contínuo em SA e em alguns casos, cíclico. As adaptações são motivadas principalmente por modificações no contexto de execução, que possui propriedades de comportamento periódico (ex.: horas) ou cujos valores discretos podem se alternar com certa frequência (ex.: estado de operação). A ocorrência de

ciclos pode ser motivada por diferentes situações, como, por exemplo, sobrecargas em horários específicos e componentes defeituosos.

Por definição, SA livres de modelo não planejam seus estados da configuração arquitetural previamente (SALEHIE; TAHVILDARI, 2009). Consequentemente, estados instáveis atingidos podem ter sérias consequências, principalmente se o SA for crítico. Logo, promover a qualidade das adaptações e permitir a avaliação das configurações arquiteturais assumidas tem importância significativa para SA.

2.3.5 BASE DE CONHECIMENTO

A base de conhecimento atua como uma memória compartilhada. As fases do ciclo de adaptação descritas anteriormente armazenam e recuperam dados da base de conhecimento. Por exemplo, a fase de monitoramento concentra na base de conhecimento o contexto de execução atualizado, permitindo acesso as demais fases (BRUN *et al.*, 2009). Além disso, a fase de análise armazena seus parâmetros de decisão na base de conhecimento, quando estes são dinâmicos (BRUN *et al.*, 2009). Além destes dados, a base de conhecimento pode manter outros modelos do SA. Um exemplo é a configuração arquitetural do SA, necessária à fase de planejamento. Logo, a base de conhecimento é de importância fundamental à compreensão do comportamento de um SA. Seu papel de concentrador da informação favorece o acesso aos dados necessários a realização de análises do comportamento do SA.

2.4 APOIO FERRAMENTAL EXISTENTE

SA é uma área de pesquisa multidisciplinar (CHENG *et al.*, 2009), o que resulta em grande diversidade de tecnologias e técnicas em desenvolvimento. A seguir são descritas algumas destas alternativas, consideradas de importância significativa para este trabalho.

O fato de modelos reduzirem a complexidade ao introduzir abstrações é fundamental para enfrentar alguns dos desafios de SA (BLAIR; BENCOMO; FRANCE, 2009). Por exemplo, SA devem monitorar seus requisitos continuamente, e a utilização de modelos para representar requisitos poderia ser aplicada em tempo de execução (BENCOMO, NELLY, 2009). Existem outras utilizações de modelos, como para a verificação das adaptações realizadas em tempo de execução (TAMURA *et al.*, 2013) e as descritas nos trabalhos a seguir.

A abordagem Genie (BENCOMO, N.; BLAIR, 2009) visa permitir a realização de diferentes etapas do desenvolvimento de um SA, com o auxílio de MDE. Os autores argumentam ainda que os modelos criados no Genie aproximam os mundos de

programadores, arquitetos, engenheiros de requisitos e especialistas de domínio, favorecendo a colaboração entre todos.

Linhas de Produto de Software (LPS) definem arquiteturas capazes de gerar diferentes produtos que compartilham um conjunto de componentes. LPS definem pontos de variabilidade em sua configuração arquitetural para que possam ser gerados diferentes produtos. Para gerar um produto da LPS é preciso solucionar todos os pontos de variabilidade, definindo quais serão os componentes utilizados ou não. Quando esta definição pode ser realizada durante a execução do produto, obtém-se uma Linha de Produto de Software Dinâmico (LPSD).

LPSD permitem abstrair a lógica das adaptações como uma máquina de estado (MORIN *et al.*, 2009), favorecendo sua confiabilidade. A Figura 4 fornece um cenário que exemplifica esta capacidade de abstração. Na Figura 4, cada configuração arquitetural derivável da LPSD é representada por um estado. Transições planejadas são representadas por arestas. Ainda na Figura 4, é possível perceber que esta LPSD em particular não possui transições entre todos os produtos deriváveis. Entretanto, a evolução da LPSD pode aumentar o número de estados e transições possíveis. Consequentemente, aumentando a complexidade do diagrama de estados e afetando negativamente sua compreensão por um arquiteto de software.

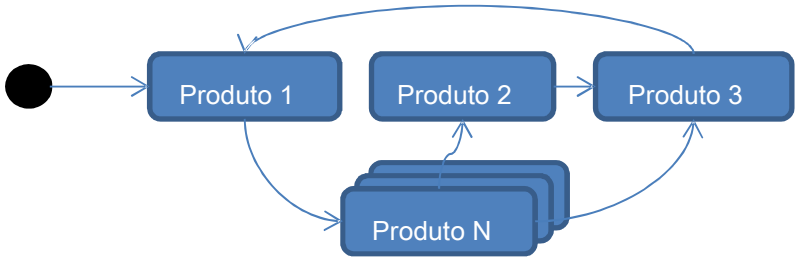


Figura 4 Transições entre Produtos da LPSD

Em MORIN *et al.* (2009), LPSD são utilizadas em conjunto com técnicas de MDE. A combinação é um esforço para lidar com a complexidade de LPSD de larga escala e favorecer a confiabilidade das adaptações. Para este fim, os autores desenvolveram um projeto chamado DIVA[1]. A arquitetura do projeto é composta de cinco componentes: um processador de eventos complexos, um mecanismo de decisão orientado a objetivos (do inglês, *goals*), um verificador de configurações arquiteturais, um injetor de aspectos em modelos (do inglês, *aspect model weaver*) e um gerenciador de configurações. Estes componentes utilizam e se comunicam empregando vários modelos, e em alguns casos realizando transformações entre

[1] Site do projeto: http://www.ict-diva.eu/DiVA

modelos. DIVA é uma ferramenta tanto de desenvolvimento, para especificação dos modelos, quanto de suporte à execução, quando realiza a adaptação do sistema alvo por meio destes modelos.

LPSD têm sido associadas a Contratos de Adaptação Arquitetural em pesquisas recentes (CARVALHO; LOQUES; MURTA, 2010; ROSENMÜLLER *et al.*, 2011). Por meio de Contratos de Adaptação Arquitetural, adaptações podem ser planejadas em termos dos objetivos do SA. Paralelamente, as restrições impostas pela LPSD favorecem a confiabilidade dos estados atingidos pelo SA.

Em termo de infraestrutura de desenvolvimento, as alternativas variam desde recursos no nível da linguagem de programação a frameworks especializados em SA. JPloy (LÜER; VAN DER HOEK, 2004) e AspectJ (LADDAGA, 2001), permitem alterar o comportamento de aplicações Java, modificando seu *bytecode*. Em GEORGAS; VAN DER HOEK e TAYLOR (2009), o autor utiliza conectores especialmente desenvolvidos para suportar adaptações dinâmicas. Em GEORGAS; TAYLOR, (2004) é descrita a abordagem KBAAM. A abordagem fornece um conjunto de ferramentas para gerenciamento de adaptações na configuração arquitetural. Estas ferramentas estão integradas ao ambiente de desenvolvimento ArchStudio 3 (INSTITUTE FOR SOFTWARE RESEARCH, 2005). Por sua vez, o ArchStudio 3 está integrado à IDE Eclipse. Em GARLAN; SCHMERL; CHENG, (2009), os autores desenvolveram um *framework* chamado *Rainbow*. O *framework* que fornece uma infraestrutura flexível de suporte a SA e utiliza entidades semelhantes aos Contratos de Adaptação Arquitetural. Estas soluções surgiram a partir de esforços acadêmicos, porém existe interesse da indústria neste sentido.

Fora do ambiente acadêmico, diversas empresas, dentre elas IBM, Adobe, Red Hat e Oracle, formaram um consórcio chamado OSGi *Alliance*[2]. O consórcio desenvolve a especificação para plataforma OSGi, um ambiente voltado à construção de sistemas que seguem o modelo de componentes e de arquitetura orientada a serviços (TAVARES; VALENTE, 2008). Desde seu surgimento, a plataforma vem sendo empregada para construção de SD, como a IDE Eclipse [3], integrado a aplicações com demanda por carga dinâmica como o Glassfish [4]. Além disso, a aceitação da plataforma tem motivado sistemas a se tornarem compatíveis, como no caso do JBoss[5].

[2] Fonte: http://www.osgi.org/About/Members
[3] Site oficial: http://www.eclipse.org/
[4] A partir da versão 3. Site oficial: http://glassfish.java.net/
[5] Site do projeto: http://www.jboss.org/jbossas/osgi

A plataforma OSGi fornece um ambiente de execução para aplicações escritas em linguagem Java. Neste ambiente, componentes interagem livremente, sem que suas dependências internas afetem os demais. Para isso, as aplicações são encapsuladas em unidades chamadas *Bundles*. Um *bundle* concretiza o conceito de componente, definindo serviços públicos e requeridos. Além disso, *bundles* podem ser manipulados dinamicamente utilizando-se serviços específicos. Por exemplo, ao atualizar um *bundle*, serviços da plataforma notificam seus dependentes e os paralisam até a conclusão da atualização. Existem serviços e ferramentas voltados à simplificação do desenvolvimento dos *bundles*. Por exemplo, o uso de anotações relativas a Serviços Declarativos (ALLIANCE, 2009) evita a necessidade de declarar explicitamente serviços públicos e requeridos. Diversas implementações da especificação estão disponíveis para a plataforma OSGi. Entre as principais estão a Equinox [6] (desenvolvida pelo Eclipse Foundation) e a Felix [7] (desenvolvida pela Apache Foundation). Trabalhos recentes tem explorado esta plataforma também no âmbito acadêmico em pesquisas voltadas a SA (FERREIRA; LEITÃO; RODRIGUES, 2010; FONSECA; BENEDITTO; WERNER, 2012).

2.5 SISTEMAS AUTOADAPTÁVEIS DA ATUALIDADE

SA são relativamente comuns atualmente, podendo estar presentes em dispositivos de uso diário, como *smartphones* e *tablets*. Entretanto, dados sobre propriedades e características de aplicações comercias são de difícil acesso, enquanto que sistemas de pesquisa têm seus dados amplamente divulgados.

Em VILLEGAS *et al.* (2011) e SALEHIE e TAHVILDARI (2009) foram pesquisados diversos SA e seus dados foram usados para gerar a Tabela 1. A Tabela 1 concentra dados relevantes para este trabalho e filtra SA que não participaram de ambos os estudos. Entre estes SA, houve totalidade de soluções utilizando contratos e modelos para realizar adaptações de forma planejada. Além disso, a utilização de adaptações fortes tem número significativo.

Atualmente, está em desenvolvimento na Universidade Federal Fluminense (UFF) um SA que utiliza contratos e modelos para realizar adaptações fortes de forma planejada. O SA é chamado Sistema Computacional Inteligente de Assistência Domiciliar à Saúde (SCIADS) (CARVALHO; COPETTI; LOQUES, 2010). O SCIADS é um sistema crítico de auxílio

[6] Site oficial: http://www.eclipse.org/equinox/
[7] Site oficial: http://felix.apache.org/site/index.html

domiciliar à saúde de idosos, desenvolvido pelo laboratório Tempo [8]. Além das características citadas anteriormente, o SCIADS utiliza o paradigma de LPSD para obter níveis elevados de confiabilidade em suas adaptações, necessários a um sistema crítico.

Tabela 1 Sistemas autoadaptáveis da atualidade (adaptada de VILLEGAS et al. (2011) e SALEHIE e TAHVILDARI (2009))

Projetos	Definição das adaptações (VILLEGAS et al., 2011)	Elemento adaptado	Questões de concepção	
		Impacto e custo	Abordagem	Tipo
		Fraco/forte	Adaptação planejada/ resultante	Baseado em modelos ou livre
Océano (APPLEBY et al., 2001)	contratos	fraca/forte	planejada	modelos
Rainbow (GARLAN et al., 2004)	contratos	fraca/forte	planejada	modelos
ROC (CANDEA et al., 2006)	contratos	fraca	planejada	modelos
K-Component (DOWLING; CAHILL, 2004)	contratos	fraca/forte	planejada/resultante	modelos
CASA (MUKHIJA; GLINZ, 2005)	contratos	fraca/forte	planejada	modelos
J3 (WHITE; SCHMIDT; GOKHALE, 2005)	contratos	fraca	planejada	modelos

Suas adaptações são motivadas por variações ocorridas no contexto de execução, visando a garantia da disponibilidade de serviços essenciais ao paciente monitorado pelo sistema. Características do quadro clínico e estado sintomático do paciente também influenciam nas adaptações do sistema, principalmente no que se refere à diversidade e frequência do monitoramento de seus dados fisiológicos. A ocorrência de falhas no sistema

[8] Site do laboratório: www.tempo.uff.br

poderia ocasionar a perda do paciente. Portanto, existe uma forte demanda por alternativas para realização de análises criteriosas de tais eventos.

Atualmente o SCIADS possui uma versão construída na linguagem C#, e duas outras versões em desenvolvimento. Estas versões são variantes atualizadas da original em C#, ambas construídas na linguagem Java. Um versão é voltada para o sistema operacional Android e outra para a plataforma OSGi. Atualmente, a versão para Android não possui mecanismo de adaptação definido.

2.6 EVOLUÇÃO DINÂMICA DE SISTEMAS AUTOADAPTÁVEIS

Software evolui por meio de processos voltados ao ambiente estático de desenvolvimento. Existem inúmeros modelos de processo de desenvolvimento, como, por exemplo, Cascata (ROYCE, 1970), Espiral (BOEHM, 1988), e Ágil (BECK, 1999). Apesar de suas diferenças, existem atividades comuns. Em todos eles existe o conceito de que é preciso haver uma solicitação para que ocorra mudança. Uma mudança só ocorre se for avaliada como pertinente. Em seguida será implementada e posteriormente liberada para operação. Estas atividades ocorrem sempre com intervenção humana. No entanto, SA evoluem durante sua operação, realizando estas atividades de forma automática em um ambiente dinâmico.

O constante estado de evolução dinâmica cria desafios na área de SA e motiva um numero significativo de pesquisas (SALEHIE; TAHVILDARI, 2009; VILLEGAS *et al.*, 2011; TAMURA *et al.*, 2013). Para viabilizar a solução destes desafios é importante permitir que SA tenham sua evolução dinâmica estudada. Para isso são utilizadas ferramentas que registram a evolução dinâmica do SA e viabilizam seu estudo posterior.

As seções a seguir procuram dar mais detalhes sobre os assuntos abordados no parágrafo anterior. Na seção 2.6.1 são descritos desafios enfrentados pelos SA. A seção 2.6.2 aborda como é registrada a evolução dinâmica de SA na atualidade. Finalmente, na seção 2.6.3 é descrita de que maneira seria possível estudar a evolução dinâmica de um SA.

2.6.1 DESAFIOS

A pesquisa descrita em SALEHIE e TAHVILDARI (2009) fornece outro conjunto de desafios enfrentados por SA, além de um extenso estudo sobre a área de SA. Neste caso, os desafios foram classificados entre as categorias engenharia, propriedades de SA, processo de adaptação (aqui chamado ciclo de adaptação) e interação humana com o SA. A seguir serão descritos desafios pertencentes a estas classes que são de interesse desta pesquisa.

Engenharia envolve desafios no campo de análise de requisitos, modelagem, implementação, realização de testes e avaliação de SA. Estes dois últimos tem recebido pouca atenção da comunidade cientifica, apesar de sua importância. A realização de testes impõe desafios. Por exemplo, a variedade de possibilidades a serem verificadas torna complexa a construção de testes. Consequentemente, este desafio afeta a confiança do sistema, uma vez que atestar a estabilidade após adaptações envolve a realização de testes extensivos. A avaliação da qualidade de um SA por meio de métricas ou outros critérios ainda era incipiente em 2009 (SALEHIE; TAHVILDARI, 2009), porém surgiram trabalhos investigando esta demanda. Em VILLEGAS *et al.* (2011) são propostas diversas métricas, enquanto em TAMURA *et al.* (2013) é proposta uma alternativa para a verificação de métricas de forma integrada ao ciclo MAPE-K. Em VAN DER HOEK, DINCEL e MEDVIDOVIC (2003) são propostas métricas para avaliação da qualidade de uma configuração arquitetural, segundo interesses de um LPS. Estas métricas determinam o número de serviços providos utilizados (PSU, do inglês *provided service utilization*) e de serviços requeridos utilizados (RSU, do inglês *required service utilization*) por cada componente. PSU indica a utilidade do componente para o restante da configuração arquitetural. RSU indica o peso do componente para a configuração arquitetural. A avaliação destes valores depende dos objetivos dos desenvolvedores, que determinam valores aceitáveis para cada uma delas. Em relação a configuração arquitetural como um todo, podem ser calculadas as métricas PSU e RSU médias e PSU e RSU compostas. PSU e RSU compostas medem a coesão da configuração arquitetural e devem ser próximas de 1.

Propriedades de SA, citadas na seção 2.2, raramente se apresentam de forma individual em um SA. Entretanto, coordenar a evolução do SA para que não ocorram conflitos entre estas propriedades é um desafio pouco pesquisado. Especialmente quando consideramos a avaliação da propagação das adaptações, tanto no próprio sistema quanto no contexto de execução. Existe ainda o desafio de avaliar os efeitos sobre o desempenho do sistema ao tentar buscar múltiplas propriedades sem a coordenação adequada, principalmente quanto à relação de custo/benefício para o sistema.

O **ciclo de adaptação** impõe desafios que podem ser subdivididos por fases do ciclo onde estão inseridos. Na fase de monitoramento, foram desenvolvidas técnicas dinâmicas de coleta dos dados a fim de promover eficiência. No entanto, o contexto de execução é uma fonte de incertezas para o SA (ESFAHANI; MALEK, 2012). Logo, estudar a evolução em operação do contexto de execução é importante para o ajuste do monitoramento. Na fase de análise, detectar problemas imprevistos nas informações do contexto de execução é um

desafio significativo. Dados históricos do contexto de execução permitiriam a realização de análises por meio de diferentes técnicas, como, por exemplo, mineração de dados. Uma vez detectados problemas no contexto de execução, realizar a adaptação necessária é outro desafio. Por exemplo, como determinar corretude e completude das adaptações frente às especificações do sistema, como lidar com falhas durante adaptações e como determinar ordem de realização das ações de modificação. Pesquisas neste sentido tem envolvido o uso de métodos formais (BRAGA; CHALUB; SZTAJNBERG, 2009), MDD e outras abordagens, entretanto tendem a ser muito específicas.

A **interação humana** com SA inclui sua interferência nas adaptações, gerenciamento e evolução dinâmica de políticas de adaptação e a sensação de confiança passada pelo sistema a seus usuários. Em especial, favorecer a confiança no sistema é um desafio, pois SA tomam decisões de forma autônoma e sem obrigatoriedade de gerar rastros das razões que os levaram a estas decisões. Este fato prejudica a construção da confiança no sistema, uma vez que as adaptações são transparentes aos usuários. Além disso, a relação do sistema com os serviços de que depende também sofre da mesma falta de rastros sobre adaptações realizadas.

2.6.2 REGISTROS DA EVOLUÇÃO DINÂMICA

Sistemas de grande complexidade e escala compartilham acesso a vários recursos em múltiplos processos, o que dificulta a realização de depuração (SMITH; KOREL, 2001). No caso de SA, usualmente existe uma base de conhecimento compartilhada por todos os componentes (SALEHIE; TAHVILDARI, 2009). Somado a isso, SA realizam inúmeras tarefas paralelamente, como monitoramento de diversos sensores em busca de anomalias. Como alternativa, para viabilizar a análise do comportamento de um SA, usualmente são utilizados mecanismos de registro ou log (SALEHIE; TAHVILDARI, 2009).

Logs podem conter diferentes informações sobre ações e eventos de um sistema, registrando detalhes de seu comportamento em forma de um arquivo texto. Estes arquivos podem facilmente atingir dimensões bastante volumosas e grande complexidade (VALDMAN, 2001). Além disso, analisar arquivos de log pode ser uma tarefa árdua e propensa à falha (SMITH; KOREL, 2001; ROUILLARD, 2004) e também requerer procedimentos complexos e grandes recursos computacionais (VALDMAN, 2001). Por exemplo, (ZAIDMAN; DEMEYER, 2004) propõe um processo heurístico de clusterização para tornar humanamente possível a análise de arquivos de log onde existe certa frequência de eventos semelhantes, onde os clusters formados tornariam evidentes padrões de comportamento relevantes.

Usualmente, logs são registrados de forma assíncrona por diferentes processos em um mesmo arquivo. Outro fato importante é que logs não expressam a semântica da informação neles contida, como relações entre registros individuais. Portanto, correlacionar registros é um grande desafio em arquivos de log, especialmente quando estão registrados em arquivos distintos (ROUILLARD, 2004). Por exemplo, em ROUILLARD (2004) é descrita uma técnica chamada SEC (do inglês, *Simple Event Correlator*) especialmente desenvolvida para arquivos de log gerados por sistemas de redes de computadores.

A utilização de logs para análises em tempo real requer o uso de técnicas de filtragem para reduzir o montante de informação processada (ROUILLARD, 2004). Porém, é possível que filtros afetem os resultados de outras técnicas aplicadas como a SEC ou a clusterização, citadas anteriormente.

2.6.3 ANÁLISE DA EVOLUÇÃO DINÂMICA

A compreensão do comportamento evolutivo apresentado por SA requer conhecimento sobre as respostas das 5W+1H em diferentes fases do ciclo de vida do sistema em função de seus requisitos serem modificados dinamicamente (SALEHIE; TAHVILDARI, 2009). Por exemplo, estas respostas são importantes para atestar a confiança das adaptações realizadas. Para isso, informações detalhadas sobre o comportamento do SA devem estar disponíveis para a realização de análises em diferentes fases do ciclo de vida.

Responder às 5W+1H requer informações potencialmente complexas de se extrair de arquivos de log, como visto na seção 2.6.2. Por exemplo, localizar os registros pertinentes a um dado evento de adaptação do sistema demandaria extensiva filtragem, determinar o contexto de execução em que esta ocorreu seria ainda mais complexo, e, finalmente, identificar qual alteração no contexto provocou a adaptação seria quase impraticável sem uma ferramenta especializada. Estes exemplos são particularmente interessantes quando se deseja identificar problemas no ciclo de adaptação.

Entre os possíveis problemas durante o ciclo de adaptação, está a realização de adaptações se e somente se forem necessárias. SA podem não realizar adaptações necessárias, gerando falsos negativos, ou realizar adaptações desnecessárias, gerando falsos positivos. Além disso, não é possível garantir que todos os problemas sejam detectados durante a fase de desenvolvimento. Isso motiva a utilização de mecanismos complementares de análise, como, por exemplo, suporte a monitoramento em tempo de execução dentro e fora do ambiente de desenvolvimento. Por fim, um mecanismo de auditoria ajudaria a detectar as razões que levaram o sistema a realizar adaptações incorretas, mesmo que tenham sido despercebidas

durante a fase de desenvolvimento ou no monitoramento. Durante estas atividades encontram-se muitos dos desafios de SA. Logo, mecanismos apropriados para as atividades de monitoramento e auditoria são de importância significativa para SA.

Realizar monitoramento significa acompanhar o comportamento exibido pelo sistema em condições de uso controladas ou não. Inúmeros aspectos de um sistema podem ser monitorados, variando desde elementos de baixo até alto nível. Por exemplo, no caso de SA, monitorar a evolução dos dados capturados por um sensor até monitorar os problemas encontrados pelo mecanismo de decisão em função desta evolução.

Com o objetivo de promover a verificação independente da conformidade de um software, atividades de auditoria realizam a avaliação do cumprimento de diferentes critérios no produto de software frente a especificações, padrões, regulamentações (IEEE, 1998). A auditoria é realizada por profissionais externos e independentes do produto auditado, e demandam acesso a evidências detalhadas para realizar suas tarefas. Um exemplo de evidências são os registros do comportamento de um SA ao longo se sua execução. Caso estes registros estejam na forma de logs textuais, os auditores passarão pelos mesmos desafios citados na sessão anterior.

2.7 TRABALHOS RELACIONADOS À EVOLUÇÃO DINÂMICA DE SISTEMAS AUTOADAPTÁVEIS

Inicialmente a busca por trabalhos relacionados procurou por pesquisas direcionadas ao controle de evolução em tempo de execução para SA. Entre os termos buscados estavam: *self-adaptive systems evolution control, self-adaptive systems configuration management*, *self-adaptive systems version control*. A busca retornou quatro resultados: ROSHANDEL *et al.* (2004), VAN DER HOEK, (2004), GEORGAS, VAN DER HOEK e TAYLOR (2005), e GEORGAS, VAN DER HOEK e TAYLOR (2009).

Posteriormente, a busca por trabalhos relacionados foi modificada, focando em trabalhos capazes de registrar a evolução em tempo de execução de SA. A busca por trabalhos relacionados foi composta de três etapas. Uma das etapas foi buscar por trabalhos relacionados por meio da ferramenta Google Scholar [9], que demonstrou ter os melhores resultados na fase preliminar, com as seguintes palavras chave: *runtime configuration management*, *runtime version control*, *runtime evolution control*, *runtime evolution management*. A segunda etapa utilizou artigos previamente encontrados para busca cruzada

[9] http://scholar.google.com.br

empregando-se suas referências (ROSHANDEL *et al.*, 2004; VAN DER HOEK, 2004; GEORGAS; VAN DER HOEK; TAYLOR, 2005, 2009). E a terceira foi uma pesquisa entre os trabalhos publicados pelos autores dos artigos anteriores.

Inicialmente, os artigos encontrados foram filtrados considerando-se dois critérios: registrar diferentes aspectos da evolução dinâmica de um SA e permitir análise cruzada dos aspectos registrados. Além disso, alguns trabalhos eram fruto da continuidade da pesquisa realizada anteriormente. Apesar dos poucos critérios de filtragem, foram identificados apenas dois trabalhos, a serem descritos nas subseções posteriores.

As seções seguintes fornecem mais detalhes sobre os trabalhos relacionados na seguinte ordem: seção 2.7.1 descreve os sistemas complementares Ménage e SelectorDriver, e a seção 2.7.2 aborda o sistema ARCM, considerado de maior similaridade com nossa proposta.

2.7.1 MÉNAGE E SELECTORDRIVER

Em VAN DER HOEK (2004), os autores identificaram a necessidade de aperfeiçoar os mecanismos de solução de variabilidades LPSD. A abordagem visa fornecer suporte a especificações de configurações arquiteturais de uma LPSD em qualquer momento do ciclo de vida de um sistema, a partir de uma solução integrada. Paralelamente, a abordagem realiza o controle da evolução das configurações arquiteturais selecionadas para as instâncias.

Para atingir estes objetivos, os autores deram continuidade ao trabalho realizado anteriormente (VAN DER HOEK *et al.*, 2001; ROSHANDEL *et al.*, 2004). Estes trabalhos descrevem a abordagem Mae, focada no controle da evolução de arquiteturas de software. Esta abordagem foi modificada no trabalho descrito em VAN DER HOEK (2004), onde possui novos objetivos.

Com o objetivo de suportar a definição de variabilidades em qualquer fase do ciclo de vida do software, são apontadas quatro funcionalidades obrigatórias a qualquer abordagem neste sentido. As funcionalidades são: um modelo para representação arquitetural de LPSD, uma ferramenta para modelagem de arquiteturas de LPSD, uma ferramenta capaz de auxiliar na definição de uma configuração válida da arquitetura e ferramentas capazes de aplicar configurações arquiteturais em qualquer fase do ciclo de vida do sistema. A abordagem dos autores utilizou as alternativas descritas a seguir para concretizar estas demandas.

O modelo de representação da arquitetura de um sistema pertencente a uma LPSD foi definido por meio da Linguagem de Descrição de Arquiteturas chamada xADL 2.0 (DASHOFY; VAN DER HOEK; TAYLOR, 2002). A linguagem xADL 2.0 é construída por

meio da combinação de esquemas XML, em um total de nove. Este fato confere propriedades importantes à xADL, como flexibilidade e extensibilidade, pois pode ser modificada quando necessário por meio de novas estruturas ou recombinações das mesmas.

A ferramenta para modelagem da arquitetura utilizada foi o Ménage (GARG *et al.*, 2003). O Ménage é capaz de definir variabilidades tanto em diversidade quanto em temporalidade, ou seja, componentes e suas versões compatíveis. Pontos de variabilidade são responsáveis pela determinação das possíveis configurações arquiteturais de uma linha, que pode facilmente chegar a números intratáveis manualmente. O Ménage também impõe uma política bloqueante de versionamento das arquiteturas. Por exemplo, arquitetos devem fazer *check-out* da arquitetura para realizarem modificações, e *check-in* do resultado para adicionar uma nova versão da arquitetura. Outros arquitetos não podem modificar a arquitetura enquanto este *check-in* não for realizado.

A ferramenta SelectorDriver realiza a criação de configurações arquiteturais com base no modelo em xADL criado na ferramenta Ménage. Para definir uma configuração arquitetural, primeiro o usuário escolhe uma série de pares nome e valor, que irão indicar quais qualidades ele deseja. Estas qualidades são utilizadas para definir os pontos de variação a partir dos componentes disponíveis na LPSD, uma forma de versionamento intencional (CONRADI; WESTFECHTEL, 1998). Entretanto, o resultado é variável devido ao algoritmo procurar por soluções de forma heurística. Além disso, é possível que variabilidades não sejam solucionadas pela ferramenta. Cabe ao usuário verificar se a configuração resultante está completa e válida.

A ferramenta responsável por aplicar uma configuração arquitetural selecionada chama-se Selector, e pode ser utilizada por meio da ferramenta SelectorDriver ou integrada a outras aplicações para adaptações automatizadas.

A avaliação da abordagem foi realizada considerando três cenários envolvendo fases distintas de aplicação das configurações arquiteturais. No primeiro cenário, a abordagem foi utilizada em fase de desenvolvimento para solucionar pontos de variabilidade em um sistema de entretenimento hipotético. No segundo cenário, a abordagem foi utilizada para aplicação da configuração arquitetural desejada no momento de inicialização da execução de um sistema capaz de realizar autocura, com algumas alterações na infraestrutura da abordagem. O terceiro cenário envolveu a aplicação da abordagem em tempo de execução, em um SA criado para este fim, onde a abordagem foi integrada.

A abordagem é capaz de realizar adaptações em sistemas pertencentes a uma LPSD em tempo de execução, caracterizando-os como SD. Entretanto, a abordagem controla apenas

a evolução das configurações arquiteturais. Não são registradas as motivações e condições que levaram as transições entre configurações arquiteturais.

2.7.2 ARCM

O trabalho realizado inicialmente em GEORGAS; VAN DER HOEK e TAYLOR (2005) e posteriormente modificado em GEORGAS; VAN DER HOEK e TAYLOR (2009), descreve a abordagem ARCM. Esta abordagem é voltada ao favorecimento da confiabilidade das adaptações realizadas por SA em tempo de execução. Para favorecer a confiabilidade das adaptações realizadas, a abordagem oferece três funcionalidades. Primeiro, permite o registro das modificações realizadas na configuração arquitetural. Segundo, suporta operações de reversão destas modificações. Terceiro, disponibiliza visualizações das informações históricas da evolução do sistema. A abordagem se caracteriza por ser integrada à infraestrutura responsável pelas adaptações (a abordagem KBAAM).

Uma representação em alto nível da arquitetura do ARCM pode ser vista na Figura 5. O elemento central na Figura 5 representa o modelo da configuração arquitetural de um SA. As adaptações do SA e consequentes atualizações do modelo são realizadas pelo componente chamado Gerenciador de adaptações. As adaptações são determinadas pelo componente chamado Gerenciador de evolução, por meio de políticas (aqui chamadas de contratos de adaptação, vide seção 2.3.2). Finalmente, o componente ARCM realiza as funcionalidades da abordagem, citadas anteriormente.

Em especial, os autores afirmam que SA podem realizar adaptações imprevistas em tempo de execução, possivelmente atingindo configurações arquiteturais indesejáveis. Para que seja possível ter conhecimento sobre tais eventos, ARCM monitora e registra as configurações arquiteturais atingidas pelo sistema alvo em tempo de execução. Além disso, disponibiliza operações de restauração de versões estáveis da configuração arquitetural e permite a visualização gráfica de representações das configurações arquiteturais registradas para análise.

A abordagem parte do princípio que a evolução das configurações arquiteturais apresentada por SA muito se assemelha ao cenário de desenvolvedores trabalhando em um mesmo conjunto de código fonte. Com isso, ARCM adota técnicas provenientes de GC para realizar seus objetivos, como registrar múltiplas versões das configurações arquiteturais.

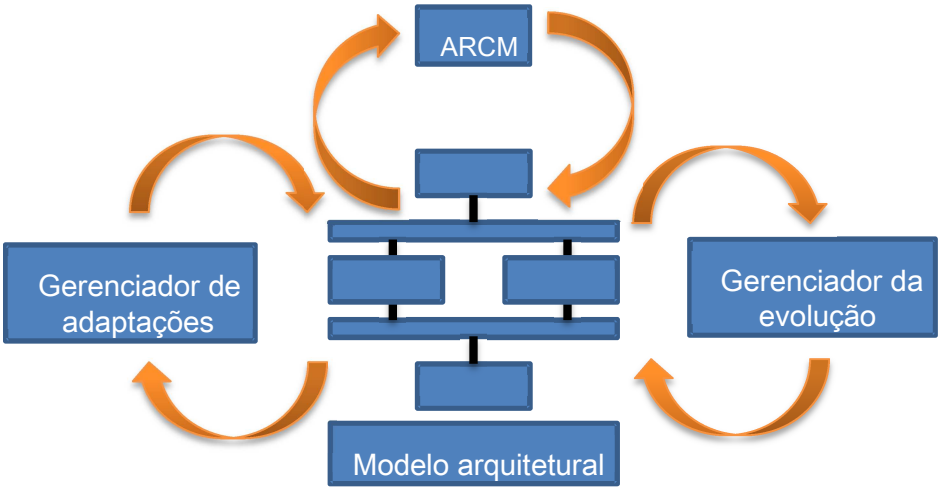


Figura 5 Arquitetura em alto nível da abordagem (retirada de GEORGAS, VAN DER HOEK e TAYLOR (2005))

Para realizar o registro das transições entre configurações arquiteturais, estas são representadas em xADL 2.0, assim como descrito na seção 2.7.1, e armazenadas em arquivos em formato XML. Além disso, são registrados os contratos de adaptação que motivaram a transição. ARCM não só registra informações como permite a interferência sobre o comportamento do sistema. Para isso disponibiliza operações de *rollback* e *rollforward*, respectivamente retornando o sistema a versões anteriores da configuração arquitetural ou avançando para uma versão posterior.

O ArchStudio 3 é composto por diversos componentes e o ARCM utiliza principalmente três deles: o *Architecture Evolution Manager* (AEM), o *ArchDiff* e o *ArchMerge*. O AEM concretiza o componente Gerenciador de adaptações, citado anteriormente. O *ArchDiff* fornece a funcionalidade de diferenciação entre duas configurações arquiteturais, gerando um arquivo contendo as modificações necessárias para realizar a transição entre estas configurações. Finalmente, o *ArchMerge* permite a realização da transição ao aplicar os passos descritos no arquivo gerado pelo *ArchDiff* por meio do AEM.

A captura das adaptações realizadas por um SA é feita pelo componente ARCM propriamente dito. Ele monitora eventos provenientes do AEM, tanto sinalizando a inicialização do sistema quanto sinalizando adaptações. Nestes casos, o componente ARCM registra o momento da transição, a configuração arquitetural inicial e a resultante, um link para o contrato de adaptação responsável pela transição e uma assinatura única para cada configuração arquitetural. Além disso, é incrementado um contador de utilização da mesma. Todas estas informações podem ser visualizadas por meio da ferramenta de visualização integrada ao ARCM, exibida na Figura 6. O ARCM também realiza o *rollback* e *rollforward* por meio do AEM.

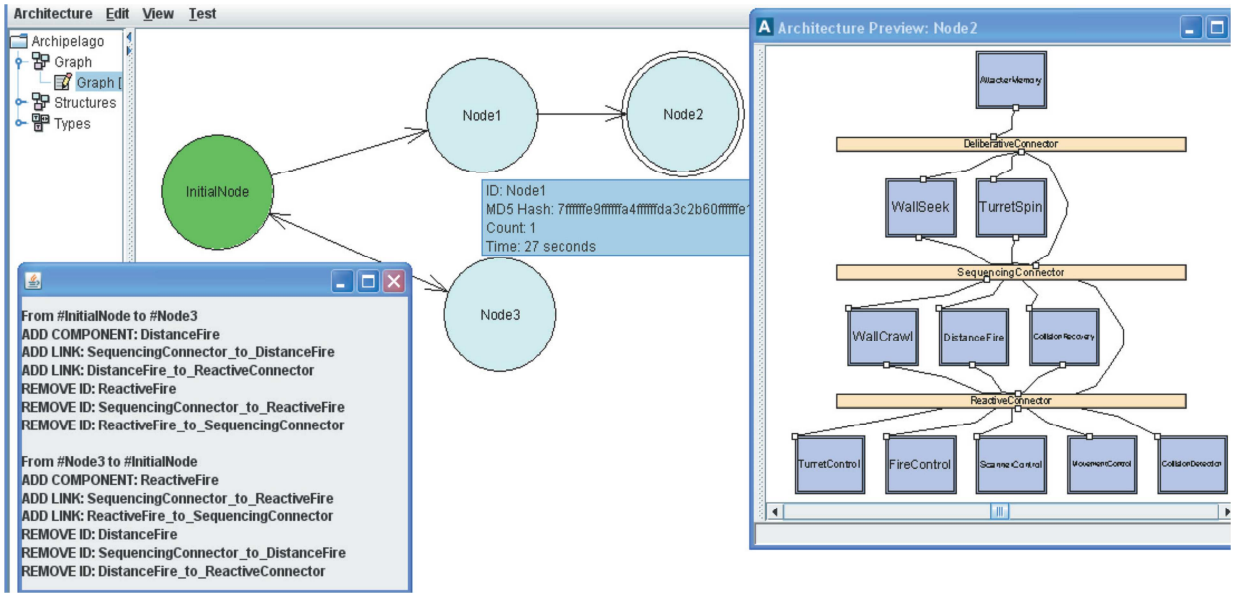


Figura 6 Ferramenta de visualização do ARCM (retirada de GEORGAS, VAN DER HOEK e TAYLOR (2005))

A avaliação da abordagem foi realizada em cenários distintos nos dois trabalhos, em vista do progresso da ferramenta. Em GEORGAS; VAN DER HOEK e TAYLOR (2005), a abordagem foi utilizada para registrar e exibir adaptações simples realizadas sobre um jogo chamado KLAX, ainda de forma bastante primária. Em GEORGAS; VAN DER HOEK e TAYLOR (2009) a ferramenta teve avanços significativos em relação à versão anterior. A avaliação consistiu em sua utilização para registrar e visualizar as adaptações realizadas por um simulador de jogos de guerra entre robôs autônomos, chamado Robocode [10]. Para permitir a avaliação dos benefícios da abordagem, o simulador foi integrado ao ArchStudio e foram construídos robôs capazes de se autoadaptar. A abordagem demonstrou ser capaz de realizar as tarefas esperadas.

Ao contrário das abordagens descritas nas sessões anteriores, ARCM apresenta clara relação com a aplicação de técnicas de GC a SA. A abordagem é a mais próxima dos objetivos deste trabalho. Entretanto, como dito na seção 2.4, o ArchStudio está integrado a IDE Eclipse, portanto a atuação do ARCM é restrita ao ambiente de desenvolvimento. Além disso, o ARCM não registra a evolução do contexto de execução do SA.

2.8 CONSIDERAÇÕES FINAIS

SA desempenham um papel importante no mundo de hoje, especialmente se considerarmos a demanda por estabilidade, eficiência e flexibilidade em dispositivos móveis

[10] Site do simulador: http://robocode.sourceforge.net

avançados. Desenvolver e manter SA é uma tarefa complexa, com significativos desafios em aberto, como aqueles descritos em CHENG *et al.* (2009) e SALEHIE e TAHVILDARI (2009). A busca por fornecimento de apoio adequado a estes sistemas tem gerado resultados significativos. Entretanto, a pesquisa realizada durante a revisão da literatura mostrou indícios de que os trabalhos atuais estão concentrados em desafios da fase de desenvolvimento de SA.

Desafios relacionados ao desenvolvimento de SA são significativos, vide a variedade de abordagens propostas para sua construção citadas anteriormente. Entretanto, desafios presentes durante a fase de operação e posteriores a ela são igualmente importantes. Negligenciá-los certamente afetaria a confiabilidade, eficiência e outras propriedades de interesse de SA. Atacar estes desafios demanda apoio adequado para a realização de monitoramento e auditoria das decisões tomadas por SA durante sua operação e posteriormente.

Os trabalhos relacionados descritos na seção anterior apontam alguns benefícios provenientes da aplicação de técnicas de GC no apoio ao monitoramento e auditoria da evolução de arquiteturas e configurações arquiteturais de SD. Entretanto, estes trabalhos deixam importantes lacunas, ao não se aprofundarem na discussão da motivação das adaptações realizadas. SA realizam suas adaptações na configuração arquitetural por razões potencialmente complexas, onde a simples comparação entre diferentes configurações arquiteturais ou a informação do contrato de adaptação utilizado sem seus valores, seriam incapazes de esclarecer suas reais motivações. Além disso, ao negligenciar o ambiente de operação, as abordagens disponíveis limitam suas contribuições. Incertezas enfrentadas por SA podem ocorrer no ambiente de operação, como, por exemplo, no contexto de execução (ESFAHANI; MALEK, 2012).

GC pode auxiliar o esclarecimento das motivações por trás das adaptações ao rastrear as condições sob as quais elas foram realizadas e os problemas percebidos pelo mecanismo de adaptação de forma mais abrangente. Ademais, a concentração destas informações em um repositório permite a realização do cruzamento de dados a fim de revelar outros fatos significativos sobre períodos de funcionamento particulares de um SA. No entanto GC é tradicionalmente aplicada em tempo de desenvolvimento, trazendo novos desafios para a sua adoção em tempo de execução.

No capítulo Capítulo 3 é descrito como propomos tratar destes desafios, por meio de uma abordagem baseada em técnicas de GC. No Capítulo 4 é descrito como implementamos a proposta.

CAPÍTULO 3 – CM@RT

3.1 INTRODUÇÃO

Como apresentado no Capítulo 2, trabalhos anteriores a este (VAN DER HOEK *et al.*, 2001; ROSHANDEL *et al.*, 2004; GEORGAS; VAN DER HOEK; TAYLOR, 2005, 2009) aplicaram técnicas de GC para registrar a evolução de SA durante sua execução, de forma restrita ao ambiente de desenvolvimento. No entanto, os mesmos se concentraram em registrar a evolução da configuração arquitetural. Desta forma, neste trabalho introduzimos a abordagem CM@RT (do inglês, *Configuration Management at Runtime*), que visa expandir as informações registradas. Esta expansão tem o objetivo de permitir o estudo da evolução dinâmica de um SA, guiado por meio de seis questões chave, denominadas 5W+1H (SALEHIE; TAHVILDARI, 2009): o que, quem, quando, como, onde e por que. Questões as quais poderão ser respondidas no futuro por mecanismos de visualização fornecidos pelo CM@RT.

A abordagem está dividida em duas partes, que podem ser realizadas concorrentemente: o registro de dados e a visualização de dados. Durante o registro de dados são registradas informações sobre a evolução dinâmica de um SA em um repositório. Entre as informações registradas estão atualizações no contexto de execução, *issues* e transições entre configurações arquiteturais. Para a visualização são geradas representações gráficas dos dados registrados no repositório, complementadas por funcionalidades de diferenciação entre contextos de execução e configurações arquiteturais. Além disso, ambas as partes podem ser utilizadas no ambiente de desenvolvimento e operação, uma vez que são independentes de ferramentas de desenvolvimento e integráveis ao SA.

Para concretizar a abordagem CM@RT, foi criada a arquitetura representada na Figura 7. É possível identificar por meio de cores dois grupos de módulos na Figura 7. O primeiro grupo, na cor azul, é formado pelos módulos pertencentes ao SA. O segundo grupo, na cor marrom, é composto pelos módulos da abordagem CM@RT. Também podem ser identificadas as principais fontes de dados sobre o comportamento do SA para a abordagem CM@RT, com fluxo representado por setas tracejadas. A troca de dados nestes fluxos é determinada por decisões de projeto tomadas pelos responsáveis por integrar os módulos da abordagem CM@RT ao ambiente do SA e seus módulos.

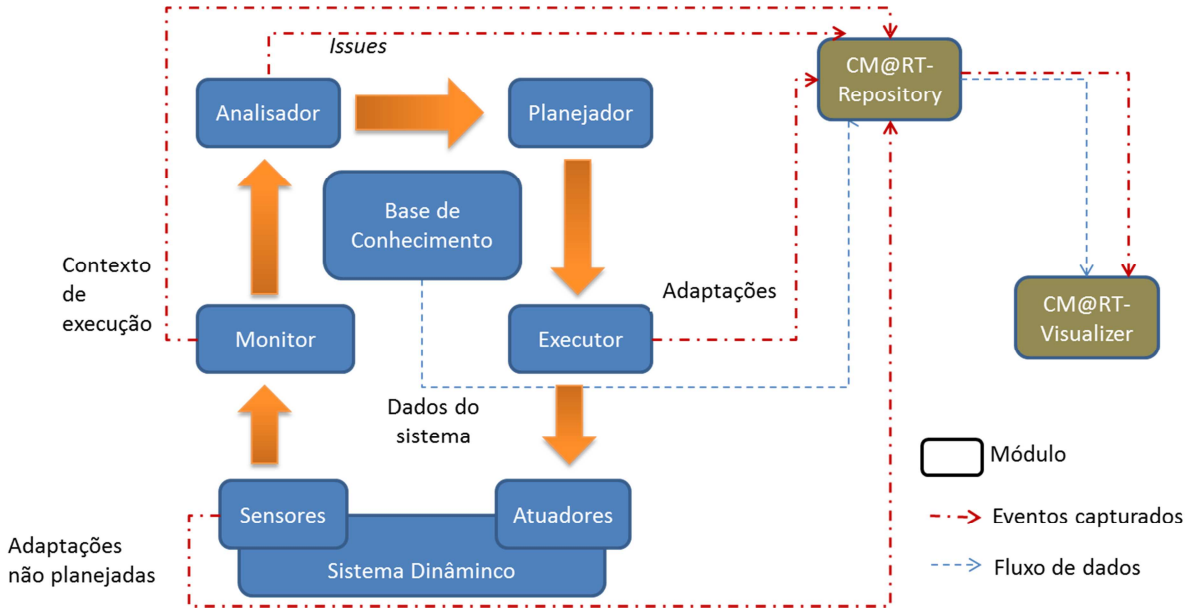


Figura 7 O ciclo MAPE-K e a abordagem CM@RT

O registro de dados é realizado por um módulo chamado CM@RT-Repository. Como visto no Capítulo 2, SA são implementados empregando-se várias tecnologias, incluindo diferentes linguagens de programação e infraestruturas de suporte. Em função destes fatos e do interesse por ampliar a aplicabilidade da abordagem, decidiu-se que CM@RT-Repository seria um módulo passível de integração em diferentes aplicações. A visualização dos dados é disponibilizada pelo módulo CM@RT-Visualizer. Este módulo utiliza as operações de *query* e *diff* disponibilizadas pela interface do CM@RT-Repository para exibir informações dos SA registrados. Além disso, observa eventos do CM@RT-Repository para fornecer funcionalidades de monitoramento.

Este capítulo está organizado em cinco seções além desta introdução. Na seção 3.2 são descritos os requisitos necessário a um SA para utilização da abordagem; Na seção 3.2 é fornecido um exemplo motivacional; Na seção 3.4 é explicado o registro de dados, incluindo seu metamodelo, a interface de serviços e como deve ser utilizada ao ser integrada; Na seção 3.5 é descrita a visualização de dados, incluindo sua arquitetura, recursos e seus propósitos; Na seção 3.6 são apresentadas as considerações finais.

3.2 REQUISITOS PARA UTILIZAÇÃO DO CM@RT

No Capítulo 2 foram descritas propriedades, características e outros aspectos que permeiam um SA. Entre estes elementos, foram selecionados os requisitos necessários a SA para utilização da abordagem. A seleção equacionou elementos entre os de maior frequência

em trabalhos relacionados e de interesses de pesquisa dos envolvidos neste trabalho. Estes elementos serão descritos a seguir.

Como dito no Capítulo 2, esta abordagem é voltada a SA com propriedades de **autoconfiguração** e **sensibilidade ao contexto de execução**. Entretanto, outras propriedades não impedem a utilização da abordagem. Por exemplo, SA **autoconscientes**, com **autocura** ou **autoaperfeiçoamento**. Neste sentindo, a abordagem demanda apenas que o SA seja capaz de informar o *issue* que motivou as adaptações e o contexto de execução.

A partir da taxonomia descrita no Capítulo 2, foi construída a Tabela 2. Esta tabela informa as características que impedem a utilização da abordagem CM@RT, se não satisfeitas. As demais características listadas na taxonomia do Capítulo 2, não afetam o uso da abordagem CM@RT.

A característica *artefato e granularidade* foi definida em função do foco em SA que realizem adaptações de *impacto e custo* forte. Uma vez que adaptações fortes modificam a configuração arquitetural do SA, tendem a representar maior risco para SA. Por exemplo, um componente adicionado pode falhar durante o processo de adaptação. No que se refere ao *tipo* nas questões de concepção, são requeridos SA baseados em modelos e fechados. SA baseados em modelos arquiteturais são o alvo deste trabalho, especialmente aqueles pertencentes a uma LPSD. As ações adaptativas suportadas são modificações na configuração arquitetural, seja por adição, remoção ou substituição de componentes. Em relação à *interoperabilidade*, não são suportados SA distribuídos neste estágio do trabalho, devido a sua complexidade.

Tabela 2 Requisitos da abordagem

Característica	Subgrupo	Requisito
Elemento adaptado	Artefato e granularidade	Componentes e conexões
	Impacto e custo	Forte
Questões de concepção	Tipo	Fechado
		Baseado em modelos
Interesses de interatividade	Interoperabilidade	SA não distribuídos

3.3 EXEMPLO MOTIVACIONAL

Para permitir a melhor compreensão das situações enfrentadas por SA, esta seção fornece um exemplo motivacional sintético. O SA utilizado é um protótipo destinado ao controle de um quarto. A aplicação é chamada JARBAS (do inglês, *Java Automated Residence Brazilian Adaptive System*) e está inserida em uma LPSD. Suas adaptações são

determinadas por contratos de adaptação arquitetural e efetivadas por um *framework* para SA. Os contratos de adaptação arquitetural utilizam variáveis de contexto para determinar ações adaptativas. O *framework* foi desenvolvido para a plataforma OSGi, baseado no ciclo MAPE-K, que é o ambiente de operação do JARBAS.

A arquitetura do JARBAS pode ser vista na Figura 8, onde são mostrados todos os componentes de software disponíveis para construção de configurações arquiteturais do SA, acrescidos da versão entre parênteses. O componente chamado JARBAS fornece funcionalidades de gerenciamento do SA. O componente chamado *Device Ctrl* realiza o gerenciamento dos dispositivos físicos conhecidos pelo SA, por meio de seus componentes de software. O componente chamado *Sensor Control* realiza o gerenciamento dos sensores físicos conhecidos pelo SA, por meio de seus componentes de software. O componente de software chamado *Window* é controla um dispositivo para abertura/fechamento de uma janela para uso de ventilação natural. O componente de software chamado *Fan* controla o funcionamento de um ventilador. O componente de software chamado *Air Conditioning* controla o funcionamento de um ar condicionador. O componente chamado *Sensor of Temperature* é um sensor capaz de fornecer a temperatura de um ambiente. O componente chamado *Sensor of Presence* é um sensor capaz de informar se existe alguém em um determinado cômodo. O componente chamado *Sensor of Room* é um sensor capaz de informar o estado da luz do quarto e de sua porta. Este componente possui duas versões disponíveis, para fim de exemplificação do uso de recursos de visualização a serem descritos posteriormente.

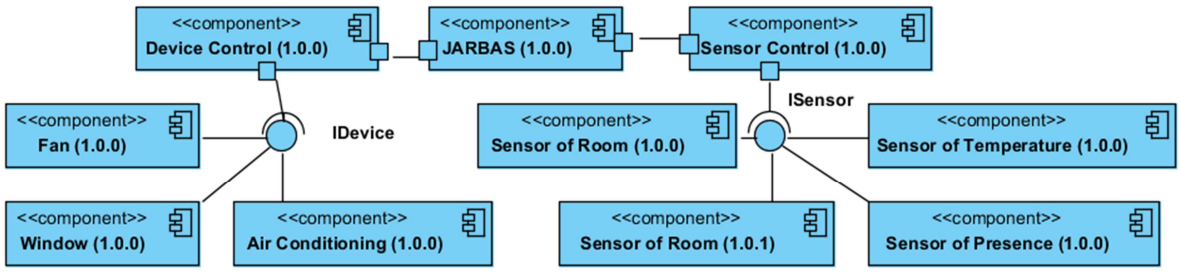


Figura 8 Arquitetura do JARBAS

As adaptações realizadas pelo JARBAS são regidas por um contrato de adaptação arquitetural chamado *Temperature Control*, descrito na Tabela 3. É importante dizer que os cenários a seguir poderiam ser tratados por parametrização. Entretanto, em situações onde haja limitação de recurso, como, por exemplo, sistemas embarcados, esta não é uma alternativa interessante uma vez que não os libera. O contrato possui quatro regras com precondições baseadas na temperatura do quarto e existência de movimentação. Quando a

precondição de uma regra é satisfeita, adaptações são realizadas como descrito a seguir. A regra nomeada *Energy Saving* determina que os dispositivos disponíveis devam ser desativados quando não é detectado movimento no ambiente. A regra nomeada *Activate Window* determina que apenas o componente *Window* deva estar ativado quando a temperatura está entre 18º C e 22º C inclusive. A regra nomeada *Activate Fan* determina que apenas o componente *Fan* deva estar ativado quando a temperatura está entre 23º C e 25º C. A regra nomeada *Activate Air Conditioning* determina que apenas o componente *Air Conditioning* deva estar ativado quando a temperatura é superior a 25º C. Estes dispositivos ajustam seu funcionamento automaticamente quando ativados, de forma a manter a temperatura do ambiente agradável. Por exemplo, o ar condicionado está configurado para manter a temperatura do ambiente em 26º C e a janela fica aberta para ventilação natural ou fechada para aquecimento.

Tabela 3 Contrato Temperature Control

Regra	Precondição	Ação
Energy Saving	Movimento não detectado	Desativar dispositivos
Activate Window	Movimento detectado	Ativar Janela
	Temperatura entre 18º C e 22º C	Desativar demais dispositivos
Activate Fan	Movimento detectado	Ativar Ventilador
	Temperatura entre 23º C e 25º C	Desativar demais dispositivos
Activate Air Conditioning	Movimento detectado	Ativar Ar Condicionado
	Temperatura acima de 25º C	Desativar demais dispositivos

As seções a seguir descrevem como o SA se comporta diante de alguns cenários planejados para ocorrer em sequência. Na seção 3.3.1 é descrito o comportamento do JARBAS ao ser inicializado. Na 3.3.2 é descrita a ativação da regra *Energy Saving*. Na seção 3.3.3 é descrita a ativação da regra *Activate Window*. Na seção 3.3.4 é descrita a ativação da regra *Activate Fan*.

3.3.1 CENÁRIO 1: JARBAS INICIADO

Ao ser inicializado, o SA JARBAS ativa a configuração arquitetural padrão da LPSD, ilustrada na Figura 9. A configuração arquitetural padrão inclui todos os componentes disponíveis. Entretanto, a versão mais recente do componente *Sensor of Room*, 1.0.1, não faz parte dela. Em seguida, conforme o contexto de execução é analisado pelo *framework* de autoadaptação, são realizadas as adaptações necessárias na configuração arquitetural.

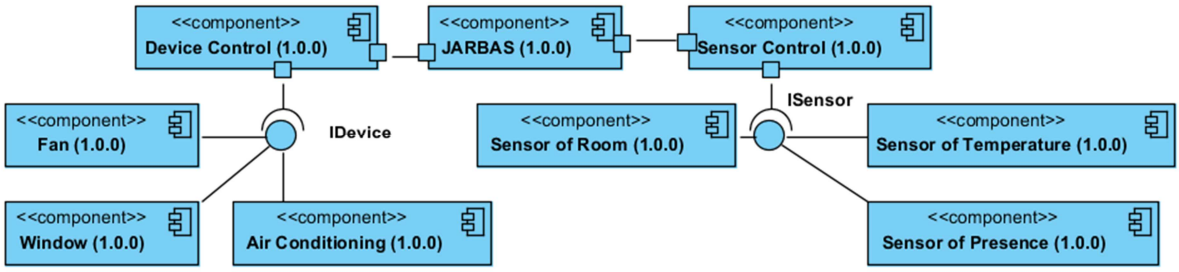


Figura 9 Configuração arquitetural padrão do JARBAS

A Tabela 4 ilustra o contexto de execução no momento de inicialização do JARBAS. Nesta tabela existem dois conjuntos de informações. O primeiro é o grupo das variáveis de contexto propriamente ditas, fornecidas pelos sensores disponíveis. O segundo grupo tem como fonte o contrato de adaptação. O valor da intepretação de cada uma das condições de suas regras foi adicionado ao contexto de execução com o objetivo de apresentar seus resultados ao usuário da abordagem.

Tabela 4 Contexto de execução inicial

Data	Fonte	Variável	Valor
2013-05-10T10:37:46	Sensor of Presence	Room.Presence	UNDETECTED
2013-05-10T10:44:29	Sensor of Room	Room.Light	OFF
2013-05-10T10:40:01	Sensor of Temperature	Room.Temperature	20°C
2013-05-10T10:44:29	Temperature Control	Activate Fan: (23°C <= Room.Temperature <= 25°C) and (Room.Presence = DETECTED)	false
2013-05-10T10:44:29	Temperature Control	Activate Air Conditioning: (Room.Presence = DETECTED) and (Room.Temperature > 25°C)	false
2013-05-10T10:44:29	Temperature Control	Activate Window: (18°C <= Room.Temperature <= 22°C) and (Room.Presence = DETECTED)	false
2013-05-10T10:44:29	Temperature Control	Energy Saving: (Room.Presence = UNDETECTED)	true

3.3.2 CENÁRIO 2: REGRA ENERGY SAVING SATISFEITA

No contexto de execução ilustrado na Tabela 4 é possível perceber que não foi detectado movimento no quarto. Logo, a regra *Energy Saving* do contrato de adaptação arquitetural *Temperature Control* foi satisfeita e deve ser efetivada. Como consequência, o SA JARBAS deve assumir a configuração arquitetural exibida na Figura 10. Nesta configuração, nenhum dispositivo relacionado à alteração ativa da temperatura do ambiente está em funcionamento.

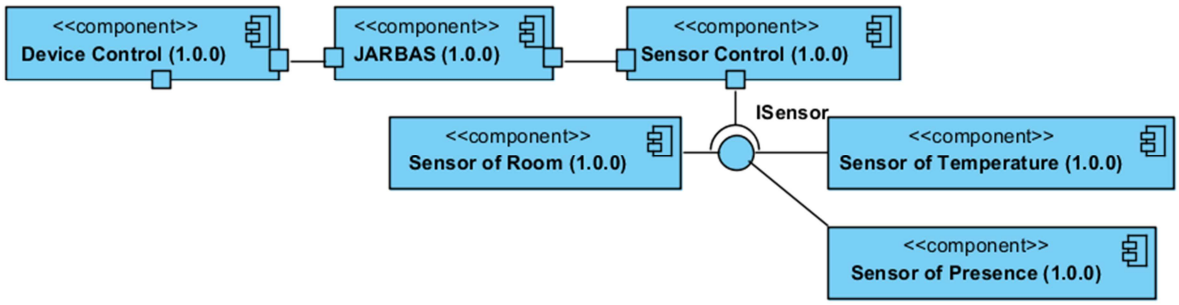


Figura 10 Configuração arquitetural para regra Energy Saving satisfeita

3.3.3 CENÁRIO 3: REGRA ACTIVATE WINDOW SATISFEITA

Alguns instantes após a realização da adaptação descrita na seção 3.3.2, o contexto de execução foi atualizado. O novo contexto de execução está reproduzido na Tabela 5. Entre as variáveis de contexto alteradas, foi detectado movimento no quarto. Logo, uma das regras que rege os dispositivos pode ser efetivada.

Neste caso a regra que deve ser efetivada é referente ao componente *Window*, em função da temperatura no quarto ser 19º C. A Figura 11 exibe a configuração arquitetural esperada como resultado da adaptação. É possível identificar na figura, que o componente *Window* foi adicionado à configuração arquitetural original, representada na Figura 10.

Tabela 5 Contexto de execução atualizado

Data	Fonte	Variável	Valor
2013-05-10T10:49:50	Sensor of Presence	Room.Presence	DETECTED
2013-05-10T11:04:36	Sensor of Room	Room.Door	CLOSED
2013-05-10T10:48:43	Sensor of Temperature	Room.Temperature	19°C
2013-05-10T11:04:36	Temperature Control	Activate Window: (18°C <= Room.Temperature <= 22°C) and (Room.Presence = DETECTED)	true
2013-05-10T11:04:36	Temperature Control	Activate Fan: (23°C <= Room.Temperature <= 25°C) and (Room.Presence = DETECTED)	false
2013-05-10T11:04:36	Temperature Control	Energy Saving: (Room.Presence = UNDETECTED)	false
2013-05-10T11:04:36	Temperature Control	Activate Air Conditioning: (Room.Presence = DETECTED) and (Room.Presence > 25°C)	false

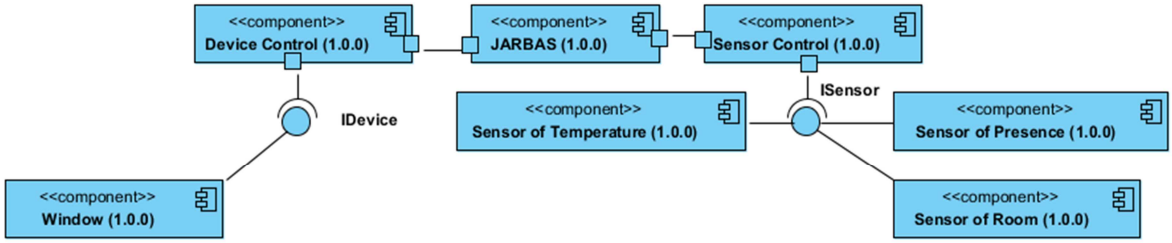


Figura 11 Configuração arquitetural para Regra Activate Window satisfeita

3.3.4 CENÁRIO 4: REGRA ACTIVATE FAN SATISFEITA

Alguns instantes após a realização da adaptação descrita na seção 3.3.3, o contexto de execução foi atualizado novamente. O novo contexto de execução está reproduzido na Tabela 6. Entre as variáveis de contexto alteradas, ocorreu aumento na temperatura do quarto. Logo, os dispositivos ativos podem ser alterados.

Tabela 6 Contexto de execução atualizado novamente

Data	Fonte	Variável	Valor
2013-05-10T11:28:59	Sensor of Presence	Room.Presence	DETECTED
2013-05-10T11:30:17	Sensor of Room	Room.Door	OPEN
2013-05-10T11:32:42	Sensor of Room	Room.Light	ON
2013-05-10T11:22:22	Sensor of Temperature	Room.Temperature	23°C
2013-05-10T11:32:42	Temperature Control	Activate Air Conditioning: (Room.Presence = DETECTED) and (Room.Temperature > 25°C)	false
2013-05-10T11:32:42	Temperature Control	Activate Window: (18°C <= Room.Temperature <= 22°C) and (Room.Presence = DETECTED)	false
2013-05-10T11:32:42	Temperature Control	Activate Fan: (23°C <= Room.Temperature <= 25°C) and (Room.Presence = DETECTED)	true

Neste caso a regra que deve ser ativada é referente ao componente *Fan*, em função da temperatura no quarto ser 23º C. A Figura 11 exibe a configuração arquitetural esperada como resultado da adaptação. Neste cenário, o componente *Sensor of Room* foi atualizado para a versão 1.0.1, apenas por razões ilustrativas e não é regida por contratos de adaptação.

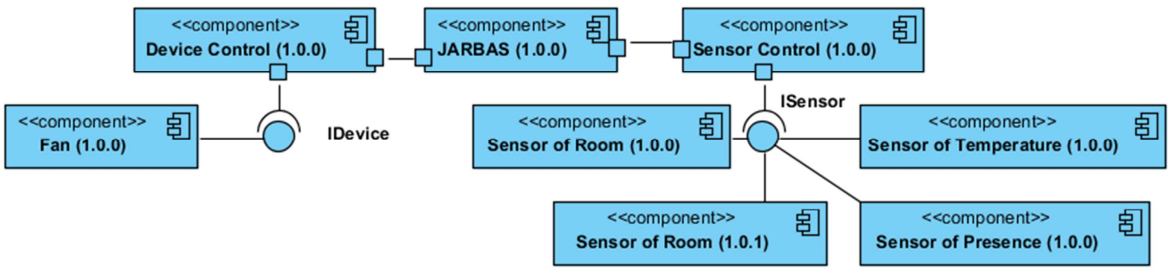


Figura 12 Configuração arquitetural para regra Activate Fan satisfeita

3.4 REGISTRO DE DADOS

As informações armazenadas são estruturadas de acordo com um metamodelo motivado pela relação entre SA e as 5W+1H. Este metamodelo é aplicado na arquitetura do CM@RT-Repository para o fornecimento das operações definidas em sua interface de serviços. Estas operações devem ser utilizadas em situações específicas durante a operação de um SA.

As subseções a seguir fornecem detalhes adicionais sobre o registro de dados. Na subseção 3.4.1 é explicado o metamodelo do CM@RT-Repository. Na seção 3.4.2 é descrita a arquitetura do CM@RT-Repository. Finalmente, a subseção 3.4.3 explica como pode ser utilizado o CM@RT-Repository para registro de dados do SA.

3.4.1 METAMODELO

O metamodelo da Figura 13 foi criado com dois objetivos: registrar a evolução e permitir o estudo do comportamento de SA por meio das informações 5W+1H. Para isso, o metamodelo combina elementos inspirados pelo ciclo de adaptação MAPE-K e conceitos vindos de GC. É importante destacar que este metamodelo não precisa ser seguido pelo SA alvo, sendo utilizado apenas pela abordagem para registro dos dados necessários. Um SA é representado pela classe *SASystem,* possuindo os atributos data de criação, nome, versão e uma chave *hash*. O par nome/versão identifica unicamente um SA no repositório. A chave *hash* também é uma alternativa de identificação, pois é gerada a partir dos atributos nome e versão. Além disso, um SA pode possuir vários contextos de execução e configurações arquiteturais.

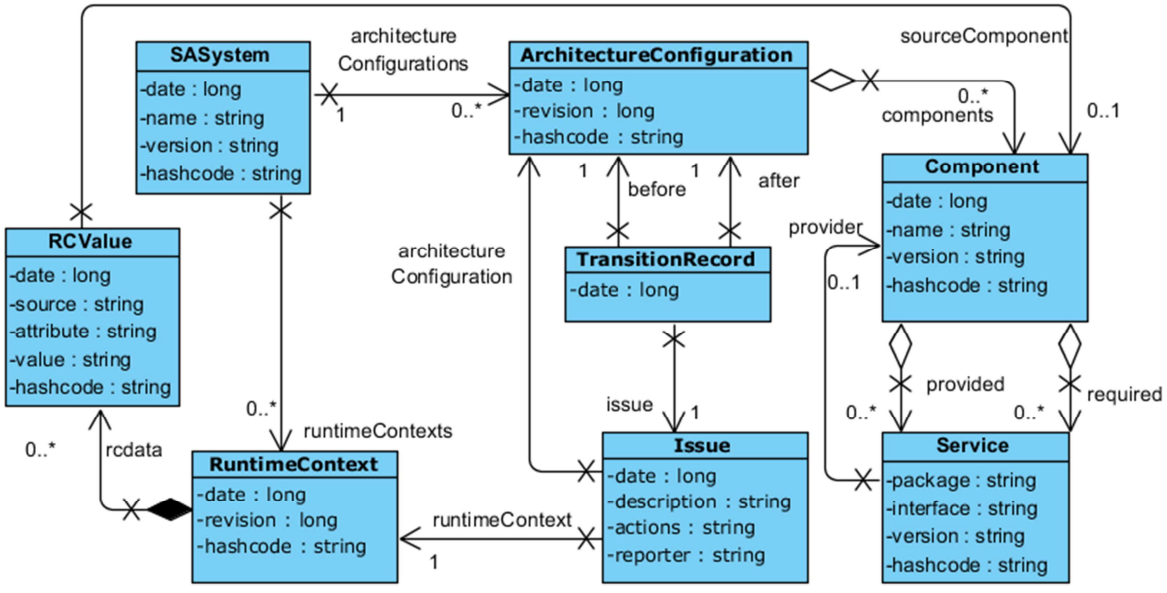


Figura 13 Metamodelo do CM@RT-Repository

O contexto de execução é representado pela classe *RuntimeContext*, possuindo o atributo data, revisão, chave *hash* e uma coleção de elementos da classe *RCValue*. O atributo data registra o instante em que foi considerado atualizado pelo SA. O atributo revisão reflete a evolução temporal do contexto de execução, sendo incrementado a cada novo registro de atualização. O atributo chave *hash* é utilizado para comparar instâncias de forma eficiente, sendo gerado a partir das chaves *hash* de cada *RCValue* contido na coleção. Cada *RCValue* armazena o valor de uma variável de contexto de execução. Uma variável de contexto de execução possui os atributos data de atualização, fonte da informação, nome da variável, valor e uma chave *hash*. A fonte da informação pode ser um componente da configuração arquitetural ou identificada pelo nome da fonte, uma vez que variáveis de contexto podem ser capturadas por componentes específicos. É importante destacar que variáveis de contexto podem ser atualizadas de forma assíncrona ou não; e que não necessariamente ocorre alteração em seu valor. Por exemplo, a temperatura de um cômodo terá pouca variação em períodos noturnos.

Mudanças na configuração arquitetural do SA ocorrem por diferentes *issues*, que são representadas pela classe *Issue*. A classe *Issue* possui os atributos data, relator, referência para um contexto de execução, descrição, referência para uma configuração arquitetural e ações de modificação. O atributo data registra o momento em que o *Issue* foi criado no SA. O atributo relator indica quem foi o responsável por informar a ocorrência do *issue*. A referência para um contexto de execução informa sob que condição estava o SA no momento da criação do *issue*. O atributo descrição serve para informar detalhes sobre a razão do *issue*. Por exemplo, o SCIADS é capaz de descrever a razão em função dos contratos de adaptação que a solicitaram. A referência para uma configuração arquitetural informa qual era o estado do SA no momento da criação do *issue*. O atributo ações permite informar as modificações desejadas no estado da configuração arquitetural. Por exemplo, o contrato descrito na Tabela 3 fornece diferentes ações possíveis.

A configuração arquitetural de um SA é representada pela classe *ArchitectureConfiguration*. A classe possui os atributos data, revisão, chave *hash* e uma coleção de componentes. O atributo data informa o momento em que a configuração arquitetural foi registrada. O atributo revisão reflete a evolução temporal da configuração arquitetural, sendo incrementado a cada novo registro realizado. O atributo chave *hash* identifica unicamente uma configuração arquitetural. Este atributo é gerado por meio da combinação ordenada das chaves *hash* da coleção de componentes. A coleção de componentes é composta por entidades da classe *Component*, descritas a seguir.

Um componente da configuração arquitetural é representado por meio da classe *Component.* A classe possui os atributos data, nome, versão, chave *hash*, uma coleção de serviços providos e outra de serviços requeridos. O atributo nome serve para compor a identificação do componente. O atributo versão é um campo livre também voltado à composição da identificação do componente. O par nome/versão identifica unicamente um componente, em relação a uma configuração arquitetural. Para identificar um componente entre todos os registrados deve ser utilizada sua chave *hash*. A chave *hash* é gerada utilizando a combinação entre as chaves *hash* do atributo nome, versão e chaves *hash* das coleções de serviços providos e requeridos. Esta distinção existe devido a um componente apresentar diferentes provedores de serviços requeridos em configurações arquiteturais distintas.

Um serviço é representado pela classe *Service*, que possui os atributos pacote, interface, versão, chave *hash* e referência para um componente provedor do serviço. Um serviço pode ser provido por vários componentes. Para indicar a topologia da configuração arquitetural existe a referência para o componente provedor do serviço. A tupla pacote/interface/versão são campos de texto livre, que identificam unicamente um serviço em uma configuração arquitetural. Para identificar um serviço entre todos os registrados deve ser utilizada sua chave *hash*. A chave *hash* é gerada combinando as chaves *hash* dos atributos pacote/interface/versão e dos atributos nome/versão do componente provedor do serviço, quando existir.

Transições entre duas configurações arquiteturais estão representadas por meio da classe *TransitionRecord*. A classe possui os atributos data, referência para o *issue* que a motivou, e referências para as configurações arquiteturais de antes e depois da transição. O atributo data se refere ao momento em que foi terminado o processo de transição entre as configurações arquiteturais no SA.

É importante destacar que os atributos *versão* não são gerados pelo CM@RT-Repository. O valor é fornecido durante o registro de dados pelo SA em que está integrado. Entretanto, o versionamento dos contextos de execução e das configurações arquiteturais é feito utilizando-se números de revisão, como mencionado anteriormente. No caso dos contextos de execução, cada *check-in* cria uma nova instância no repositório e gera incremento no numero de revisão referente. Configurações arquiteturais também são versionadas empregando-se números de revisão, porém apenas instâncias inéditas são armazenadas. Quando é realizado o *check-in* de uma configuração arquitetural, o CM@RT-Repository usa o *hash* da instância para localizar ocorrências anteriores, armazenando somente aquelas que não forem localizadas. O mesmo comportamento se estende para as

definições de serviços. Esta decisão visa evitar replicação no armazenamento das configurações. Estes mecanismos são possíveis graças à imutabilidade dos dados registrados, característica propiciada pelo versionamento. Ela garante que as chaves *hash* não sejam modificadas e permite sua utilização como indexador durante a realização das buscas e para comparação eficiente entre elementos.

3.4.2 ARQUITETURA

A integração do módulo do CM@RT-Repository ao SA demanda várias tarefas, dependentes da tecnologia empregada no ambiente de execução do sistema alvo. Independente da linguagem de programação e tecnologias adotadas no desenvolvimento da abordagem, sua integração demanda quantidade significativa de implementação por seus usuários. A abordagem requer a realização de cinco tarefas de implementação, listadas a seguir:

1. Capturar eventos e dados de interesse da abordagem.

   É realizada a critério do usuário, de acordo com seus objetivos para a abordagem. Por exemplo, o usuário pode capturar apenas atualizações do contexto de execução.

2. Transformar estes dados para o metamodelo da abordagem.

   Deve resultar em entidades construídas de acordo com a descrição do metamodelo, dada na seção 3.4.1. A transformação entre modelos em si não está no escopo desta abordagem.

3. Realizar *check-in* das entidades resultantes utilizando a interface de serviços do CM@RT-Repository, no momento oportuno definido pelos eventos capturados.

   Estão disponíveis *check-ins* para as entidades *SASystem*, *Issue*, *RuntimeContext, ArchitectureConfiguration* e *TransitionRecord*.

4. Lançar eventos próprios do CM@RT-Repository sobre *check-ins* realizados.

   Os eventos lançados são necessários ao monitoramento realizado pelo CM@RT-Visualizer.

5. Permitir acesso aos serviços de *query* e *diff* da interface do CM@RT-Repository.

   Os serviços de *query* permitem a recuperação de coleções das entidades *SASystem*, *Issue*, *RuntimeContext*, *TransitionRecord e ArchitectureConfiguration*. Os serviços

de *diff* existem para as entidades *RuntimeContext* e *ArchitectureConfiguration.*

Todos estes serviços são necessários ao funcionamento do CM@RT-Visualizer.

Para a realização das tarefas necessárias é sugerida a arquitetura representada na Figura 14. O componente chamado SA-Adapter seria responsável pela realização da primeira, segunda e quarta tarefas, enquanto que o componente CM@RT-Repository realiza a terceira e quinta tarefas. Estes dois componentes formariam o módulo do CM@RT-Repository. Este módulo teria ainda a responsabilidade de resolver quaisquer outras demandas da abordagem, como, por exemplo, bibliotecas e serviços requeridos.

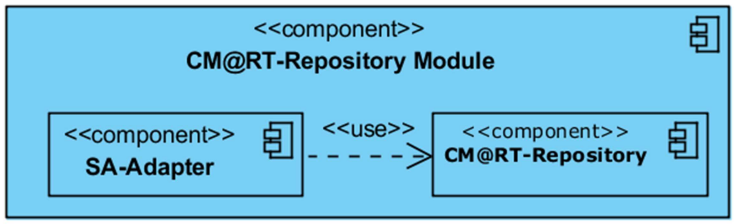


Figura 14 Arquitetura para integração do registro CM@RT-Repository

3.4.3 CENÁRIOS DE UTILIZAÇÃO

O CM@RT-Repository foi planejado para ser utilizado em diferentes cenários, baseando-se em situações possivelmente enfrentadas por SA durante sua operação. É importante dizer que as informações necessárias são fornecidas pelo mecanismo de integração construído para a abordagem. O diagrama de atividades representado na Figura 15 ilustra a utilização esperada do CM@RT-Repository para a realização das operações de *check-in* disponíveis. A seguir serão descritos estes cenários e o comportamento esperado.

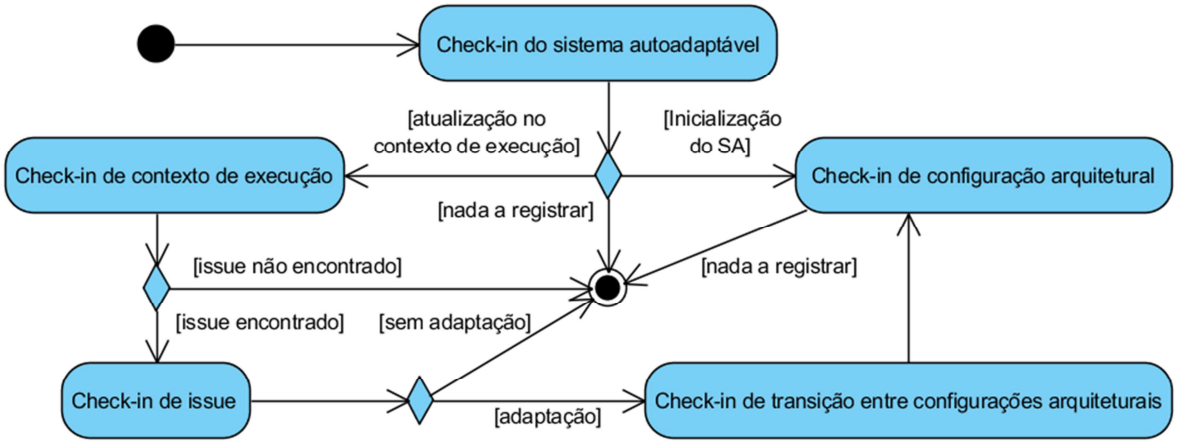


Figura 15 Diagrama de atividades do registro de comportamento de SA

Ao iniciar sua operação, o SA deve ser registrado no CM@RT-Repository. Para isso, o SA faz o *check-in* de uma instância de *SASystem*. Durante o processo de *check-in* são criados controles internos necessários ao registro da evolução. Por exemplo, é registrada a

primeira instância do sistema. Após *check-in* bem sucedido, o CM@RT-Repository está pronto para realizar *check-in* das demais entidades. A importação só é necessária uma vez e futuras inicializações do SA podem realizar operações de *check-in* diretamente.

Ao ser capturada alteração significativa no contexto de execução, deve ser feito *check-in* de uma instância de *RuntimeContext*. Para fazer o *check-in* é necessário que SA associado esteja previamente registrado. A instância de *RuntimeContext* deve conter todas as variáveis de contexto disponíveis, mesmo que seus valores não tenham sido modificados. É importante realizar *check-in* do maior número de atualizações possível, pois esta informação é uma das fontes de incertezas em SA citada anteriormente.

Quando for necessário o registro de uma configuração arquitetural deve ser feito o *check-in* de uma instância de *ArchitectureConfiguration*. Para fazer o *check-in*, é necessário que o SA associado esteja previamente registrado. O processo de *check-in* garante ainda a unicidade entre as configurações arquiteturais de um SA. Esta operação de *check-in* é utilizada atualmente apenas pelas operações de *check-in* de *issues* e transições entre configurações arquiteturais.

Ao ser capturado um *issue,* deve ser feito *check-in* de uma instância de *Issue.* Para fazer o *check-in* é necessário que o SA, o contexto de execução e a configuração arquitetural associados estejam previamente registrados. A instância de *Issue* pode ser fruto de *issues* criados por autoadaptação, intervenção humana ou falha interna de componente. Quando fruto de autoadaptação, o contexto de execução referenciado deve ser o que foi utilizado durante a determinação das adaptações necessárias. Nos demais casos, devem ser informados os contextos de execução mais recentes. É importante realizar o *check-in* destes últimos *issues*, especialmente por revelarem adaptações necessárias que não foram fruto do próprio SA. Além disso, estas informações seriam perdidas caso não houvesse o *check-in* dos *issues*. Existe ainda a possibilidade de que um *issue* encontrado não seja tratado pelo SA, ou seja, não terem ocorrido as adaptações solicitadas. Por exemplo, mecanismos de prioridade de execução podem alterar a ordem de atendimento dos *issues* e tornarem as adaptações solicitadas incompatíveis com a configuração arquitetural vigente, resultando no descarte do *issue* afetado.

Ao ser capturada uma transição entre configurações arquiteturais deve ser realizado *check-in* de uma instância de *TransitionRecord*. Para fazer o *check-in* é necessário que o SA, o *issue* e a configuração arquitetural resultantes da transição associados estejam previamente registrados. É importante realizar o *check-in* de transições entre configurações arquiteturais, pois estas registram a evolução do SA e apontam *issues* que foram efetivamente atendidos.

3.5 VISUALIZAÇÃO DOSDOS DADOS

Os dados provenientes do CM@RT-Repository permitem estudar a evolução de um SA durante sua execução. De acordo com o exposto no Capítulo 2, o estudo da evolução de um SA é necessária na fase de monitoramento e auditoria, devendo ser guiada pelas informações 5W+1H. Para atender a estas demandas foi desenvolvido o módulo CM@RT-Visualizer. As próximas seções descrevem o apoio oferecido pelo CM@RT-Visualizer para estudar os dados provenientes do CM@RT-Repository.

3.5.1 ARQUITETURA

A utilização do módulo CM@RT-Visualizer para a realização de monitoramento e auditoria requer que este tenha acesso a dados e eventos provenientes do módulo CM@RT-Repository. Por exemplo, para uso como ferramenta de monitoramento, o CM@RT-Visualizer observa em tempo real eventos de atualização gerados pelo CM@RT-Repository. Estes são capturados de acordo com os mecanismos disponíveis na infraestrutura utilizada pelo SA.

A Figura 16 ilustra uma arquitetura sugerida para concretizar a integração entre os módulos no ambiente do SA. Nesta figura é possível identificar o componente SA-Bridge, que realiza a ligação entre os módulos do CM@RT-Repository e do CM@RT-Visualizer. Ele realiza o *binding* entre a interface de serviços do CM@RT-Repository e a captura dos eventos lançados pelo SA-Adapter (descrito na seção 3.4.2). Para isso, pode ser usado o padrão *Observer* (GAMMA *et al.*, 1994) na implementação da integração.

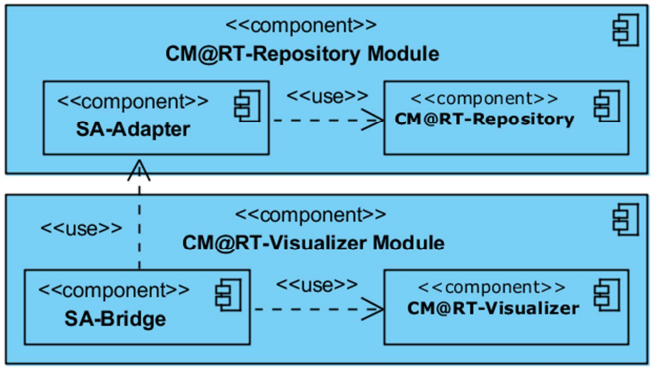


Figura 16 Arquitetura para integração da análise de dados

3.5.2 RECURSOS PARA VISUALIZAÇÃO

O CM@RT-Visualizer pode funcionar em modo de monitoramento ou para fim de auditoria. O que distingue ambos é a ativação dos recursos de atualização automática dos dados disponíveis para visualização. Uma vez selecionado o modo de utilização, o CM@RT-

Visualizer fornece um conjunto de visualizações voltadas à apresentação dos dados registrados pelo CM@RT-Repository. Estas visualizações foram concebidas com o objetivo de permitir a realização do monitoramento e auditoria do comportamento de SA. Ademais, foi considerada a semântica dos dados representados e relações entre eles ao se definir a aparência e funcionalidades das visualizações. Os dados disponíveis estão indexados em um painel chamado de índice de dados. Associado a ele existe outro painel que fornece informações adicionais sobre dados selecionados, o painel de informações. Em termos de apresentação dos dados registrados, existem três visualizações: contextos de execução, *issues* e configurações arquiteturais. Além destas, existem outras duas voltadas ao estudo comparativo entre contextos de execução e entre configurações arquiteturais. A aplicação possui ainda duas visualizações destinadas ao estudo das transições entre configurações arquiteturais, a visualização do histórico e a retrospectiva.

O **índice de dados** tem o propósito principal de organizar os dados disponíveis, de modo a favorecer a identificação do que se deseja estudar. Para isso, os dados são representados em uma estrutura hierárquica e subdividida em quatro grupos. Os grupos são chamados transições, configurações arquiteturais, contextos de execução e *issues*. No grupo transições são listadas as transições entre configurações arquiteturais. No grupo configurações arquiteturais são listadas as revisões existentes das entidades homônimas do metamodelo, assim como nos grupo contextos de execução e *issue*.

O **painel de informações** fornece descrições consolidadas sobre o elemento selecionado no índice de dados. As informações disponíveis foram planejadas em função da natureza de cada elemento disponível. Por exemplo, quando uma configuração arquitetural é selecionada, são exibidas métricas para análise de sua qualidade (VAN DER HOEK; DINCEL; MEDVIDOVIC, 2003). As métricas são PSU e RSU médias, PSU e RSU compostas.

A **visualização do contexto de execução** considera principalmente o potencialmente grande volume de variáveis de contexto a serem exibidas. Informações com estas características podem ser representadas de forma tabular, que é a representação escolhida para a abordagem, como descrito nos cenários do exemplo motivacional da seção 3.2.

A **visualização das diferenças entre contextos de execução** foi desenvolvida devido à importância de apoiar a comparação entre instâncias durante realização de estudos sobre sua evolução. O contexto de execução pode evoluir em duas dimensões: composição e valoração. O conjunto de variáveis presentes no contexto de execução pode ser modificado devido, por exemplo, a adição de um novo componente na configuração arquitetural, assim como o

inverso pode ocorrer. Enquanto que a evolução do valor de uma variável de contexto de execução é um fato conhecido, que pode ocorrer em conjunto com a situação anterior. Logo, foi desenvolvido um algoritmo direcional de diferenciação entre dois contextos de execução e uma visualização adequada à exibição dos resultados.

O **algoritmo** detecta quatro estados possíveis para uma variável de contexto: valor inalterado, valor alterado, variável removida e variável adicionada. Para isso, o algoritmo realiza quatro operações sobre conjuntos, ilustradas na Equação 1, onde o atributo data não é considerado. A operação (1) faz a interseção entre o contexto de execução de origem (CE_O) e o de destino (CE_D), obtendo um conjunto com as variáveis de contexto inalteradas (CE_{IN}). A operação (2) faz a diferença entre CE_D e CE_O, obtendo um conjunto com as variáveis de contexto adicionadas (CE_{AD}). A operação (3) faz a diferença entre CE_O e CE_D, obtendo um conjunto com as variáveis de contexto removidas (CE_{RE}). A operação (4) utiliza um operador de comparação sobrecarregado para realizar a interseção e obter o conjunto das variáveis de contexto alteradas (CE_{AL}). O operador sobrecarregado considera como elementos iguais, variáveis de contexto que possuam valores distintos e demais atributos iguais, com exceção da data.

$$1 \quad \cap \quad = \qquad\qquad 2 \quad - \quad =$$
$$3 \quad - \quad = \qquad\qquad 4 \quad \cap_{\star} \quad =$$

Equação 1 Diferenciação de contextos de execução

A **visualização dos resultados** da aplicação do algoritmo da Equação 1 sobre os contextos de execução da Tabela 4 e Tabela 5 é apresentado na Tabela 7. Para auxiliar a identificação visual dos estados encontrados, são utilizadas diferentes cores. Variáveis de contexto adicionadas estão na cor verde, removidas na cor vermelha, alteradas na cor azul e inalteradas na cor cinza.

Tabela 7 Resultado da comparação entre os contextos de execução

Estado	Fonte	Variável	Valor (antes)	Valor (depois)
Changed	Sensor of Presence	Room.Presence	UNDETECTED	DETECTED
Added	Sensor of Room	Room.Door		CLOSED
Removed	Sensor of Room	Room.Light	OFF	
Changed	Sensor of Temperature	Room.Temperature	20°C	19°C
Changed	Temperature Control	Energy Saving: (Room.Presence= UNDETECTED)	true	false
Changed	Temperature Control	Activate Window: (18°C <= Room.Temperature <= 22°C) and (Room.Presence = DETECTED)	false	true

Unchanged	Temperature Control	Activate Fan: (23°C <= Room.Temperature <= 25°C) and (Room.Presence = DETECTED)	false	false
Unchanged	Temperature Control	Activate Air Conditioning: (Room.Presence = DETECTED) and (Room.Temperature > 25°C)	false	false

A **visualização das configurações arquiteturais** é realizada empregando-se grafos direcionados, onde cada nó representa um componente e cada aresta um conector entre componentes. A Figura 17 ilustra o grafo referente à configuração arquitetural representada na Figura 10. Um componente é identificado no grafo por seu nome e versão. As arestas representam relações de dependência de serviço entre componente de origem (requerente) e destino (provedor). Por exemplo, na Figura 17 o componente Controlador de Sensores (1.0.0) provê serviços ao componente JARBAS (1.0.0).

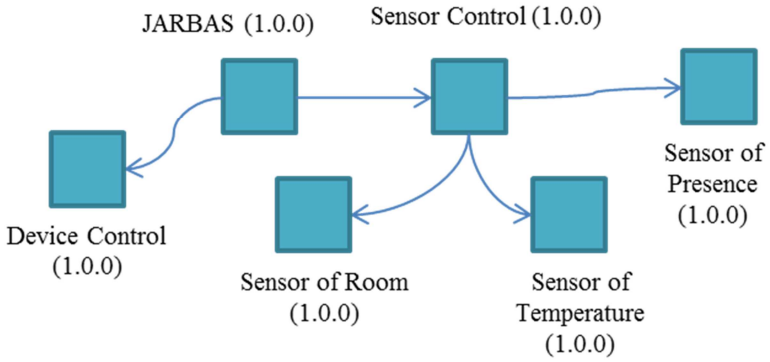


Figura 17 Grafo da configuração arquitetural para Economia de Energia

A **visualização das diferenças entre configurações arquiteturais**, assim como no caso de contextos de execução, é importante para realização de estudos sobre a evolução do SA. Logo, foi desenvolvido um algoritmo para diferenciação de configurações arquiteturais e uma representação gráfica adequada a visualização dos resultados.

O **algoritmo** detecta quatro estados possíveis para um componente: adicionado, removido, inalterado ou substituído. Para isso, o algoritmo realiza quatro operações sobre conjuntos, ilustradas na Equação 2. A operação (1) faz a intersecção entre a configuração arquitetural de origem (CA_O) e a de destino (CA_D), obtendo um conjunto com componentes inalterados (CA_{IN}). A operação (2) faz a diferença entre CA_D e CA_O, obtendo um conjunto com os componentes adicionados (CA_{AD}). A operação (3) faz a diferença entre CA_O e CA_D, obtendo um conjunto com os componentes removidos (CA_{RE}). A operação (4) utiliza um operador de comparação sobrecarregado para realizar a intersecção e obter o conjunto dos componentes alterados (CA_{AL}). O operador sobrecarregado considera como componentes iguais, os que possuem identificação idêntica e versões distintas. A detecção de componentes

substituídos é feita por meio de sua identificação e versão. Por exemplo, dois componentes com mesma identificação e versões distintas são considerados como substituição.

A **visualização do resultado** utiliza um grafo direcional, acrescido de notação em cores indicativa dos estados dos componentes. A notação tem o mesmo padrão usado na diferenciação de contextos de execução. A Figura 18 ilustra o resultado da diferenciação entre as configurações arquiteturais apresentadas nas seções 3.3.3 e 3.3.4, nesta ordem. É possível identificar na Figura 18, componentes adicionados, removidos, substituídos e inalterados. Para representar informações suprimidas no grafo, existe um formato textual que descreve as alterações encontradas na configuração arquitetural. Por exemplo, a informação que o componente *Sensor of Room* foi atualizado da versão 1.0.0 para a 1.0.1.

1 ∩ 2

3 4 ∩*

Equação 2 Diferenciação de configurações arquiteturais

A **visualização do issue** é composta da apresentação de seus atributos, informações adicionais e atalhos para visualizações pertinentes. Em um campo texto formatado são exibidos os atributos data, criador, descrição e ações. A Tabela 8 apresenta o *issue* gerado para o cenário da seção 3.3.4. Além disso, existe um atalho para visualização do contexto de execução relacionado. Nos casos onde o SA tenha executado as ações descritas, é possível visualizar seus efeitos de duas formas: descrição textual das ações realizadas, acrescida no final da transcrição dos atributos, e visualização de diferença entre configurações arquiteturais, para situações mais complexas.

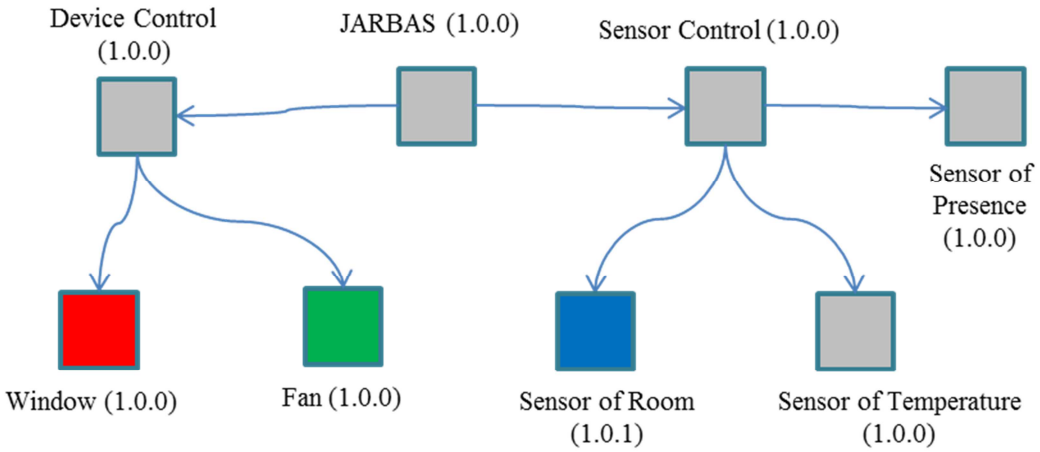


Figura 18 Resultado da comparação entre as configurações arquiteturais

A **visualização do histórico** visa permitir a análise do comportamento das transições, com foco nas configurações arquiteturais assumidas pelo SA. Para isso, foi utilizada novamente uma representação por meio de grafo direcional. Neste caso, cada nó representa

uma configuração arquitetural e cada aresta uma transição. Uma configuração arquitetural é identificada por sua revisão, enquanto que a aresta possui o número sequencial da transição como identificador. Na Figura 19 está ilustrada a visualização descrita, segundo a sequência dos cenários descritos na seção 3.2 e outras transições possíveis. Por exemplo, a transição 5 ilustra a formação de um ciclo de alternância entre configurações arquiteturais. A transição 5 ocorreu quando não houve movimento no quarto novamente e o sistema assumiu a revisão 2 da configuração arquitetural, ditada pela regra *Energy Saving*.

Tabela 8 Issue do cenário 4 da seção 3.3.4

Informação	Valor
Date	2013-05-10T11:32:47
Reporter	ADAPTATIONCONTRACT
Description	Cause (s): Contract "Temperature Control" internal conditions satisfied (23°C <= Room.Temperature <= 25°C) and (Room.Presence = DETECTED)
Actions	Remove: Window Device(1.0.0) Add: Fan Device(1.0.0)
Effects	Unchanged components: Sensor of Temperature(1.0.0) JARBAS(1.0.0) Sensor control(1.0.0) Sensor of Presence(1.0.0) Device control(1.0.0) Removed components: Window(1.0.0) Added components: Fan(1.0.0) Replaced components: Sensor of Room(1.0.0) -> Sensor of Room(1.0.1)

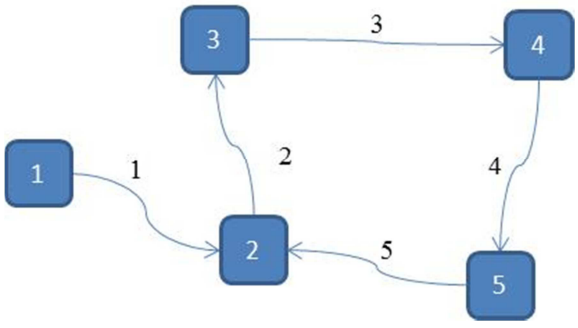
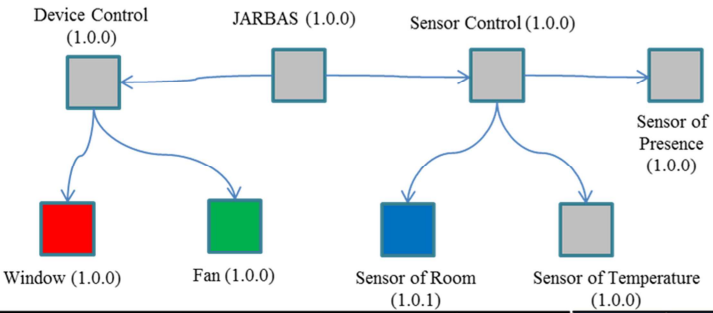


Figura 19 Histórico evolutivo das configurações arquiteturais

A **visualização da retrospectiva** das transições entre configurações arquiteturais se trata de uma animação da evolução do SA ao longo do tempo. Para cada transição registrada, são recuperados todos os dados registrados. Estes dados são representados na visualização e

atualizados passo a passo. Em cada passo, são exibidos os dados por meio da composição das visualizações pertinentes aos dados e descritas anteriormente nesta seção. Na Figura 20 é possível observar o esquema desta visualização, para a transição entre o cenário 1 e o cenário 2 do exemplo motivacional da seção 3.2. O topo da tela exibe as diferenças entre duas configurações arquiteturais, referente à Figura 18. No canto esquerdo inferior está o resultado da comparação entre contexto de execução que motivou a transição e seu antecessor, referente à Tabela 7. No canto direito inferior está o *issue*, referente à Tabela 8.



Estado	Fonte	Variável	Valor (antes)	Valor (depois)
Changed	Sensor of Presence	Room.Presence	UNDETECTED	DETECTED
Added	Sensor of Room	Room.Door		CLOSED
Removed	Sensor of Room	Room.Light	OFF	
Changed	Sensor of Temperature	Room.Temperature	20°C	19°C
Changed	Temperature Control	Energy Saving: (Room.Presence = UNDETECTED)	true	false
Changed	Temperature Control	Activate Window: (18°C <= Room.Temperature <= 22°C) and (Room.Presence = DETECTED)	false	true
Unchanged	Temperature Control	Activate Fan: (23°C <= Room.Temperature <= 25°C) and (Room.Presence = DETECTED)	false	false
Unchanged	Temperature Control	Activate Air Conditioning: (Room.Presence = DETECTED) and (Room.Temperature > 25°C)	false	false

Informação	Valor
Date	2013-05-10T11:32:47
Reporter	ADAPTATIONCONTRACT
Description	Cause (s): Contract "Temperature Control" internal conditions satisfied (23°C <= Room.Temperature <= 25°C) and (Room.Presence = DETECTED)
Actions	Remove: Window Device(1.0.0) Add: Fan Device(1.0.0)
Effects	Unchanged components: Sensor of Temperature(1.0.0) JARBAS(1.0.0) Sensor control(1.0.0) Sensor of Presence(1.0.0) Device control(1.0.0) Removed components: Window(1.0.0) Added components: Fan(1.0.0) Replaced components: Sensor of Room(1.0.0) -> Sensor of Room(1.0.1)

Figura 20 Visualização da retrospectiva

3.5.3 PROPÓSITO DOS RECURSOS DE VISUALIZAÇÃOVISUALIZAÇÃO

Utilizando os recursos descritos na seção 3.5.2, a abordagem permite responder a várias perguntas derivadas das 5W+1H, objetivando o estudo da evolução de um SA. A seguir são descritas as principais relações entre a natureza das perguntas e os recursos desenvolvidos.

Onde? Por meio da visualização da comparação entre configurações arquiteturais e da *descrição* do *issue*. O primeiro recurso fornece os elementos alterados e o segundo as razões da alteração, que podem incluir onde está o problema.

Quando? A visualização do contexto de execução fornece **quando** o *issue* surgiu efetivamente. Já a data do *issue* registrado indica **quando** ele foi percebido pelo SA.

O que? Podem ser utilizados os mesmos recursos destinados a responder **onde**.

Por quê? A visualização do *issue* fornece esta informação de forma direta em seu campo *descrição,* desde que seja fornecida pelo mecanismo de integração ao SA. Indiretamente, é possível utilizar as visualizações referentes ao contexto de execução para determinar o porquê de uma transição ter ocorrido ou não.

Quem? A visualização do *issue* fornece esta informação diretamente.

Como? As ações adaptativas registradas no *issue* fornece **como** a adaptação foi definida. A visualização da comparação entre configurações arquiteturais revela **como** ela efetivamente ocorreu.

É importante dizer que estas não são as únicas possibilidades de uso dos recursos, uma vez que podem ser combinados de diferentes formas. Por exemplo, entre as análises de escopo mais amplo temos a do comportamento das transições entre configurações arquiteturais, representado na visualização do histórico. Por meio dela tornam-se evidentes ocorrências de ciclos no fluxo de adaptações do SA. Em geral, ciclos curtos podem indicar problemas no mecanismo de adaptação, como, por exemplo, falha na modificação da configuração arquitetural. Além disso, o histórico favorece a percepção de padrões de comportamento no SA ao evidenciar suas transições em uma representação concisa.

3.6 CONSIDERAÇÕES FINAIS

O objetivo principal da abordagem CM@RT é permitir a realização de análises baseadas nas informações 5W+1H sobre o comportamento de SA, tanto durante o monitoramento quanto em auditorias. Para isso, a abordagem CM@RT propõe a utilização de dois módulos complementares. O primeiro emprega técnicas de GC para registrar as informações necessárias à realização de análises sobre o comportamento de SA. O segundo permite a realização da análise das informações registradas por meio de ferramentas especialmente criadas para este fim.

No Capítulo 4 é descrito como foram construídos estes módulos especializados no registro e visualização dos dados, respectivamente chamados de CM@RT-Repository e CM@RT-Visualizer. O capítulo descreve ainda a arquitetura dos módulos, seus funcionamento e utilização. Posteriormente, no Capítulo 5, são avaliados os benefícios gerados pela abordagem, especialmente em relação à obtenção de respostas para questões baseadas nas 5W+1H.

CAPÍTULO 4 – PROTÓTIPOS

4.1 INTRODUÇÃO

No Capítulo 3 foi explicada a abordagem CM@RT. Para concretizá-la, foram desenvolvidos dois protótipos: o módulo CM@RT-Repository e o módulo CM@RT-Visualizer. O CM@RT-Repository fornece as funções relativas ao registro de dados. O CM@RT-Visualizer fornece as funções relativas à visualização de dados. Para realizar suas funções, o CM@RT-Visualizer necessita de acesso a serviços prestados pelo CM@RT-Repository. Juntos, os protótipos permitem a realização de estudos sobre a evolução dinâmica de SA.

A realização destes estudos sobre um SA requer a integração dos protótipos ao sistema alvo. Uma vez integrados, o CM@RT-Repository passa a registrar os dados sobre a evolução do SA em um repositório. Paralelamente, os dados registrados são acessados pelo CM@RT-Visualizer para realização de monitoramento ou auditoria do SA. Ademais, o CM@RT-Repository é um módulo utilizado apenas pelo SA e pelo CM@RT-Visualizer. Por outro lado, o CM@RT-Visualizer é um módulo voltado à interação com usuários. Uma vez em execução, o usuário do CM@RT-Visualizer pode realizar monitoramento e auditoria da evolução dinâmica do SA, por meio de suas funcionalidades.

A variedade de soluções existentes para SA direcionou várias decisões de projeto, que permitiram a concretização da abordagem. Por exemplo, a demanda por portabilidade resultou na escolha da linguagem Java para o desenvolvimento dos protótipos. Esta decisão também levou em conta a considerável disponibilidade de ferramentas de apoio existentes para esta linguagem, como, por exemplo, para desenvolvimento e teste.

Esta seção descreve uma visão geral dos protótipos desenvolvidos. A seguir, a seção 4.2 descreve os requisitos de cada protótipo; a seção 4.3 descreve as arquiteturas dos protótipos; a seção 4.4 descreve o funcionamento do CM@RT-Repository; a seção 4.5 descreve a utilização do CM@RT-Visualizer. Finalmente, a seção 4.6 discute as considerações finais deste capítulo.

4.2 REQUISITOS

Como ambiente de execução do SA dos protótipos foi escolhida a plataforma OSGi, pelas características apontadas no Capítulo 2. Cada um dos protótipos que compõem a abordagem tem seus próprios requisitos funcionais e não funcionais. Os requisitos funcionais

foram definidos de acordo com as demandas do registro e da visualização de dados, permeando ambos os protótipos. Os requisitos não funcionais são semelhantes entre os protótipos, sendo motivados por interesses diversos, como favorecer a adoção da abordagem por terceiros.

Os requisitos funcionais do CM@RT-Repository estão listados na Tabela 9. Entre estes requisitos estão presentes demandas geradas pelo armazenamento e pela visualização dos dados. As demandas provenientes da visualização de dados estão identificadas por um asterisco.

Tabela 9 Requisitos funcionais do CM@RT-Repository

Grupo	Requisito
SA	Criar repositório exclusivo para diferentes SA
	Fornecer listagem dos SA com repositório*
Configurações arquiteturais	Registrar configuração arquitetural
	Fornecer listagem das configurações arquiteturais*
	Calcular diferenças entre configurações arquiteturais*
Contexto de execução	Registrar contextos de execução
	Fornecer listagem dos contextos de execução*
	Calcular diferenças entre contextos de execução*
Issues	Registrar *issues* para adaptação
	Fornecer listagem dos *issues* registrados*
Transições entre configurações arquiteturais	Registrar transições entre configurações arquiteturais
	Fornecer listagem das transições entre configurações arquiteturais registradas*

Os requisitos funcionais do CM@RT-Visualizer estão listados na Tabela 10. Estes requisitos atendem às demandas da realização do monitoramento e auditoria de um SA. Ademais, são definidas funcionalidades necessárias para a obtenção de respostas às perguntas provenientes do 5W+1H.

Tabela 10 Requisitos funcionais do CM@RT-Visualizer

Grupo	Requisito
Aplicação	Permitir atualizações automáticas para monitoramento
	Permitir atualizações manuais para auditoria
SA	Permitir seleção de SA para análise

	Exibir características de um SA
Configurações arquiteturais	Permitir seleção de configuração arquitetural para análise
	Exibir configuração arquitetural de forma gráfica
	Extrair métricas sobre configurações arquiteturais
	Exibir diferenças entre configurações arquiteturais graficamente
	Exibir diferenças entre configurações arquiteturais em formato textual
Contexto de execução	Permitir seleção de contexto de execução para análise
	Exibir contexto de execução em forma de tabela
	Exibir diferenças entre contextos de execução em forma de tabela
Issues	Permitir seleção de *issues* para análise
	Exibir *issues* em forma de texto
	Exibir características do *issue* selecionado
	Permitir acesso à configuração arquitetural após a adaptação
	Permitir acesso ao contexto de execução relacionado
Transições entre configurações arquiteturais	Exibir as transições entre configurações arquiteturais registradas
	Fornecer listagem das transições entre configurações arquiteturais registradas

Os requisitos não funcionais do CM@RT-Repository e do CM@RT-Visualizer estão listados na Tabela 11, juntamente com suas motivações. Estes requisitos têm suas motivações fundamentadas nas informações dadas no Capítulo 2.

Tabela 11 Requisitos não funcionais

Requisito	Motivação
Portabilidade	SA são executados em ambientes diversos
Confiabilidade	A abordagem fornece dados para auditoria
	SA incluem sistemas críticos de apoio à saúde
	O registro realizado é fonte de dados para localização de problemas no sistema alvo.
Reutilização	É desejável a continuidade de pesquisas sobre os protótipos desenvolvidos.
Flexibilidade	O protótipo é voltado à integração com outros sistemas

4.3 ARQUITETURA

O desenvolvimento do código fonte ocorreu na IDE Netbeans 7.2, por motivos de familiaridade e recursos de apoio disponíveis. Para gerenciar a construção dos protótipos foi

utilizado o sistema Maven (O'BRIEN *et al.*, 2008), presente em diversos ambientes de desenvolvimento na atualidade. Como SCV foi escolhido o Subversion (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2008), devido à disponibilidade de repositório no grupo de pesquisa. Para favorecer a qualidade e confiabilidade dos protótipos, foram criados testes unitários e de integração por meio do *framework* JUnit 4.8.1[11] e Pax-Exam 3.0.3[12].

As arquiteturas dos protótipos desenvolvidos foram concebidas a partir dos respectivos requisitos descritos na seção anterior. Como discutido anteriormente, a abordagem foi dividida em dois módulos principais: um para o CM@RT-Repository e outro para o CM@RT-Visualizer. No diagrama de pacotes exposto na Figura 21 está ilustrada a relação de dependência entre os módulos e seus componentes internos.

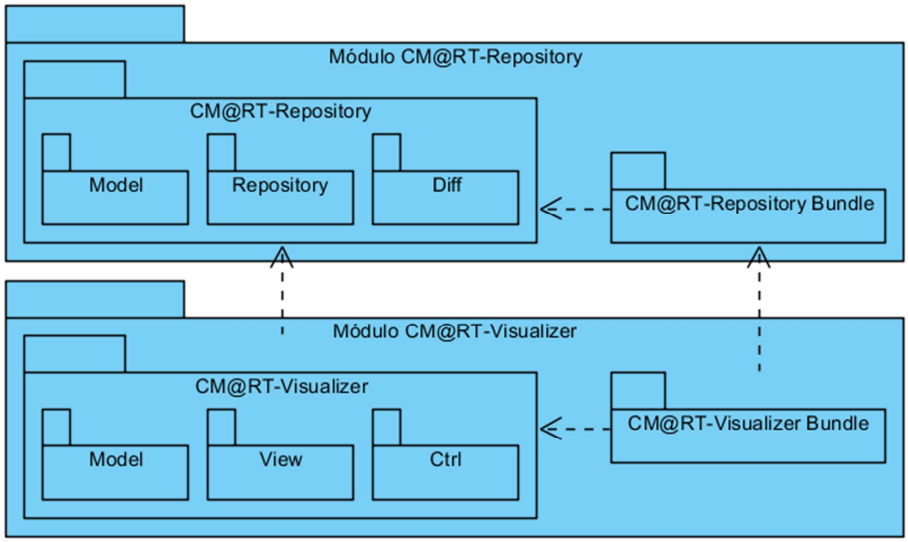


Figura 21 Módulos da abordagem CM@RT

Esta seção descreve como foi concebida a arquitetura da abordagem. Na seção 4.3.1 são fornecidas mais informações sobre a arquitetura do CM@RT-Repository. Na seção 4.3.2 são fornecidas mais informações sobre a arquitetura do CM@RT-Repository *bundle*. Na seção 4.3.3 são fornecidas mais informações sobre a arquitetura do CM@RT-Visualizer. Finalmente, na seção 4.3.4 são fornecidas mais informações sobre a arquitetura do CM@RT-Visualizer *bundle*.

4.3.1 CM@RT-REPOSITORY

O CM@RT-Repository possui um pacote principal público e quatro subpacotes privados, ilustrados na Figura 21. O pacote principal concentra os serviços públicos do

[11] Site do projeto: http://junit.org/
[12] Site do projeto: https://ops4j1.jira.com/wiki/display/PAXEXAM3/Pax+Exam

protótipo, definidos na interface *ICMatRTRepository*. O pacote *Model* concentra a implementação do metamodelo definido no Capítulo 2. O pacote *Diff* concentra as funcionalidades de comparação disponíveis. O pacote *Repository* cria uma camada de abstração sobre os serviços de armazenamento.

Como dito anteriormente na seção 3.4.2, o CM@RT-Repository define uma interface de serviços com operações de *check-in*, *query* e *diff* chamada *ICMatRTRepository*. A Figura 22 ilustra a relação destas operações. Além das operações já descritas, é possível identificar a classe *CMatRTException* voltada a descrição de problemas que surjam durante a realização das operações. As operações disponíveis na interface são resultado da composição de diversas interfaces pertencentes aos demais pacotes do CM@RT-Repository. A Figura 23 ilustra estas relações internas entre as interfaces. As demais interfaces serão descritas a seguir, agrupadas por pacote.

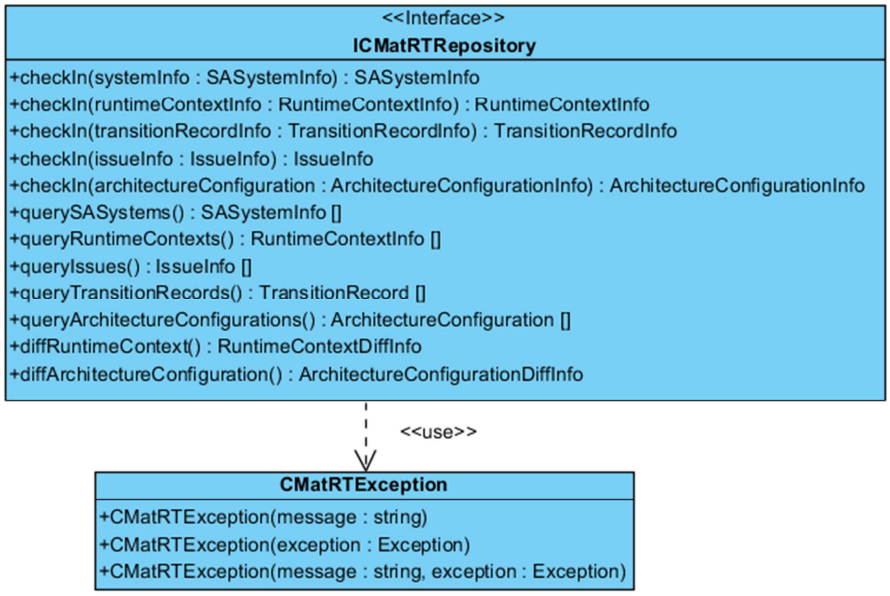


Figura 22 Operações da interface ICMatRTRepository

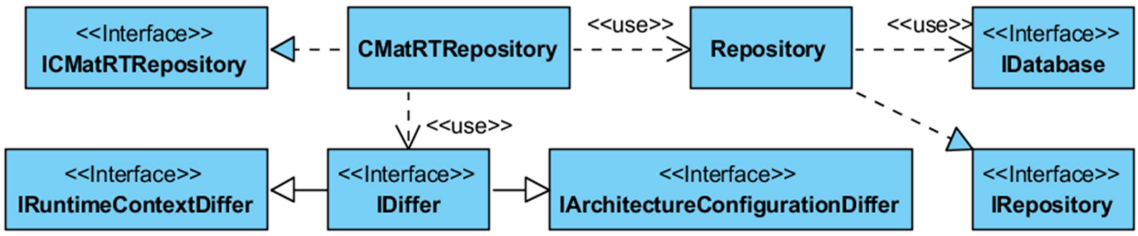


Figura 23 Diagrama de classes reduzido com as interfaces do CM@RT-Repository

O pacote *Model* detém as classes ilustradas na Figura 24. As classes implementam diferentes entidades do metamodelo original, mantendo o padrão de nomenclatura. Por exemplo, a classe *SASystemInfo* implementa a entidade *SASystem* e a classe *ComponentInfo*

implementa a entidade *Component*. O sufixo *Info* foi adicionado para facilitar a identificação de classes da abordagem e evitar conflitos com SA aos quais se integra. É importante destacar que as classes que possuem o atributo *hashcode* utilizam o algoritmo SHA-1 (STANDARD, 1993) em sua geração.

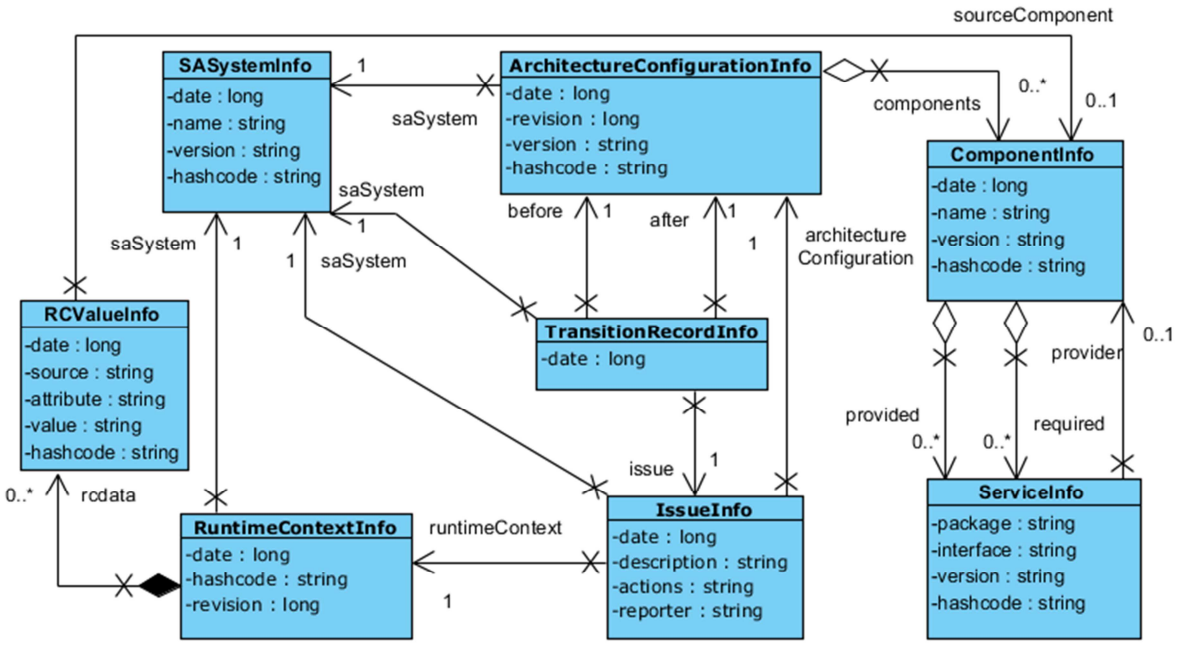


Figura 24 Implementação do metamodelo do CM@RT-Repository

A Figura 25 ilustra as interfaces do pacote *Diff*. A interface *IDiffer* concentra os serviços de comparação disponíveis. Para isso, a interface *IDiffer* estendendo serviços de comparação, atualmente estão disponíveis duas interfaces. A primeira é a interface *IRuntimeContextDiffer* que define os serviços de comparação entre contextos de execução. A segunda é a interface *IArchitectureConfigurationDiffer* para a comparação entre configurações arquiteturais. A Figura 25 ilustra como foi implementada a interface *IDiffer*, por meio da classe *Differ*. A classe *Differ* apenas delega às classes *ArchitectureConfigurationDiffer* e *RuntimeContextDiffer* a realização dos serviços. Para representar os resultados das diferenciações, são utilizadas as classes *ArchitectureConfigurationDiffInfo* e *RuntimeContextDiffInfo*. Estas classes registram dados usados para a comparação (configurações arquiteturais e contextos de execução, respectivamente), bem como os elementos adicionados, removidos, alterados e inalterados.

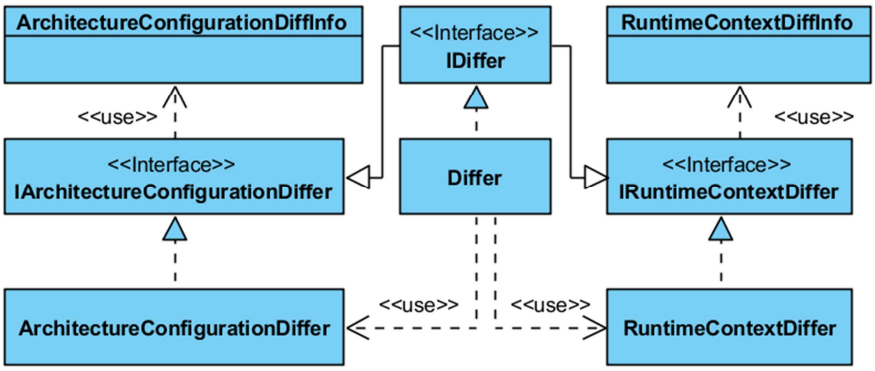


Figura 25 Implementação da interface IDiff

A Figura 26 ilustra as interfaces do pacote *Repository*. Os serviços de banco de dados estão definidos por meio da interface *IDatabase*, utilizada pela classe *Repository* para implementar a interface *IRepository*. A interface *IRepository* define os serviços necessários ao processo de registro e recuperação das informações no repositório. É responsabilidade do mecanismo de integração fornecer a implementação do serviço de banco de dados, representada na figura pela classe *Database*. Esta decisão foi tomada em função da variabilidade de recursos disponíveis em ambientes para SA. Por exemplo, a plataforma OSGi pode ser executada em ambientes embarcados ou servidores.

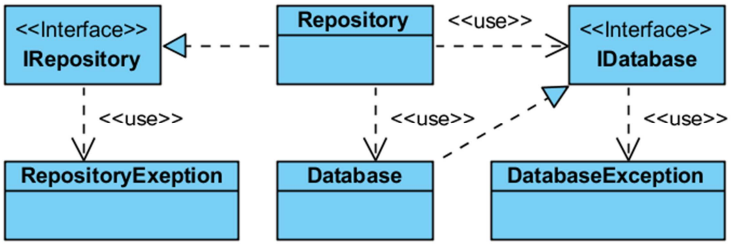


Figura 26 Implementação da interface IRepository

Atualmente o CM@RT-Repository é composto de 27 classes. Visando promover o requisito de confiabilidade, estas classes são submetidas a um total de 105 testes realizados com a da ferramenta JUnit 4.8.1. Os testes variam entre verificações simples e cenários de utilização complexos. Por exemplo, os serviços definidos na interface *ICMatRTRepository* são testados por meio de um cenário de operação simulado, incluindo geração de contextos de execução aleatórios.

4.3.2 CM@RT-REPOSITORY BUNDLE

Para atender ao requisito de execução na plataforma OSGi, o módulo CM@RT-Repository foi encapsulado em um *bundle*. Além disso, é por meio deste módulo que o CM@RT-Repository se integra ao SA alvo.

A Figura 27 ilustra um diagrama de pacotes com principais elementos da arquitetura do CM@RT-Repository *bundle*. O pacote CM@RT-Repository Bundle representa o *bundle* do CM@RT-Repository, que é testado pelas das ferramentas fornecidas pelos pacotes Pax-Exam 3.0.3 e JUnit 4.8.1. O pacote Pax-Exam 3.0.3 permite a realização de testes de integração entre o CM@RT-Repository *bundle* e a plataforma OSGi. O testes de integração são complementados por testes unitários via JUnit 4.8.1. Além destes pacotes, o pacote *Component* fornece um conjunto de classes de apoio a interação com a plataforma OSGi. Por exemplo, utilização de Serviços Declarativos fornecidos indiretamente pelo pacote Apache felix annotations 1.6.0. O pacote SA representa hipoteticamente o conjunto de entidades definidas pelo SA onde o CM@RT-Repository *bundle* está integrado.

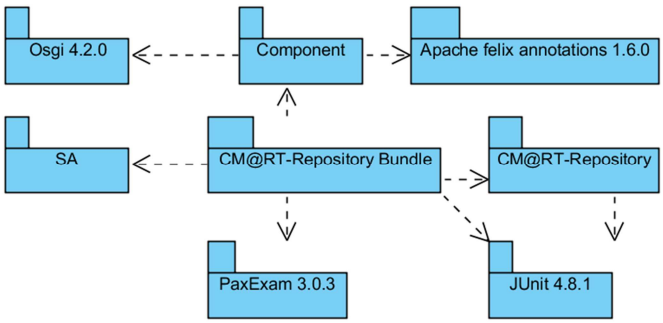


Figura 27 Arquitetura simplificada do bundle para o CM@RT-Repository

A integração entre o SA e o CM@RT-Repository *bundle* visa à realização de duas tarefas principais. A primeira é observar o SA e acionar o CM@RT-Repository para registrar a evolução no momento devido. Para isso, o CM@RT-Repository *bundle* realiza a segunda tarefa, que é recuperar e transformar as entidades existentes no SA para modelagem do CM@RT-Repository.

A observação do SA pelo CM@RT-Repository *bundle* ocorre por meio do monitoramento de eventos gerados pela infraestrutura do próprio SA e da plataforma OSGi. Por exemplo, a captura de transições entre configurações arquiteturais não planejadas é realizada por meio do CM@RT-Repository *bundle*. Para perceber a ocorrência destas transições são monitorados eventos de modificação no estado dos *bundles* residentes na plataforma OSGi, como, por exemplo, a desinstalação de um deles.

Quando eventos de interesse ocorrem, o CM@RT-Repository requer o fornecimento de dados específicos. Entretanto, SA possuem suas próprias entidades e estas provavelmente não serão equivalentes às existentes no CM@RT-Repository. Além disso, é possível que seja necessário acessar diferentes fontes de dados para compor as entidades requeridas à abordagem CM@RT-Repository. Por exemplo, obter a configuração arquitetural de um SA

pode ser feito de diferentes formas. Caso o SA possua a representação da configuração arquitetural, é necessário apenas sua transformação para a modelagem do CM@RT-Repository. Caso contrário, é necessário capturar esta configuração arquitetural do ambiente de execução do SA. Esta tarefa é realizada pelo CM@RT-Repository *bundle*, de acordo com o ambiente de execução do SA alvo. Exemplificando, na plataforma OSGi é possível obter os *bundles* residentes e suas conexões, sendo necessário filtrar aqueles que pertencem efetivamente ao SA.

4.3.3 CM@RT-VISUALIZER

A arquitetura do CM@RT-Visualizer é dividida em camadas. Estas camadas seguem o padrão de projeto *Model-View-Controller* (GAMMA *et al.*, 1994). Na Figura 21 é possível identificar os pacotes referentes a estas três camadas, a saber, *Model*, *View* e *Ctrl*.

O pacote *Model* concentra a modelagem dos dados recuperados do CM@RT-Repository. O CM@RT-Visualizer não utiliza a modelagem original do CM@RT-Repository, para diminuir o acoplamento entre os módulos. Além disso, permite a sobrecarga de métodos utilizados na representação gráfica das entidades e eficiência de recuperação dos dados. A Figura 28 ilustra o diagrama de classe do pacote *Model*. Além de entidades que refletem as definidas no metamodelo da seção 3.4.1, são mostradas as entidades que representam resultados de diferenciação entre configurações arquiteturais e contextos de execução. A classe *ArchitectureConfigurationDiff* concentra as configurações arquiteturais usadas na diferenciação e o resultado obtido em quatro coleções de componentes: adicionados, removidos, inalterados e substituídos. A classe *RuntimeContextDiff* concentra os contextos de execução usados na diferenciação e o resultado obtido em quatro coleções: adicionados, removidos, inalterados e substituídos.

O pacote *Ctrl* concentra as funcionalidades de controle por meio de um conjunto de classes *Singleton* (GAMMA *et al.*, 1994) representadas na Figura 29. Para realizar a comunicação entre o CM@RT-Repository e o CM@RT-Visualizer deve ser utilizada a classe CMatRTManager. A classe *QueryManager* utiliza a classe anterior para recuperação de dados do CM@RT-Repository, e a classe *ModelFactory* para transformá-los na modelagem do CM@RT-Visualizer por meio do padrão de projeto *Factory* (GAMMA *et al.*, 1994). A classe *Ctrl* é utilizada para registrar que dados estão sob estudo do usuário no momento. A classe *ConfigManager* concentra configurações da ferramenta CM@RT-Visualizer, como, por exemplo, se está ou não em modo de monitoramento.

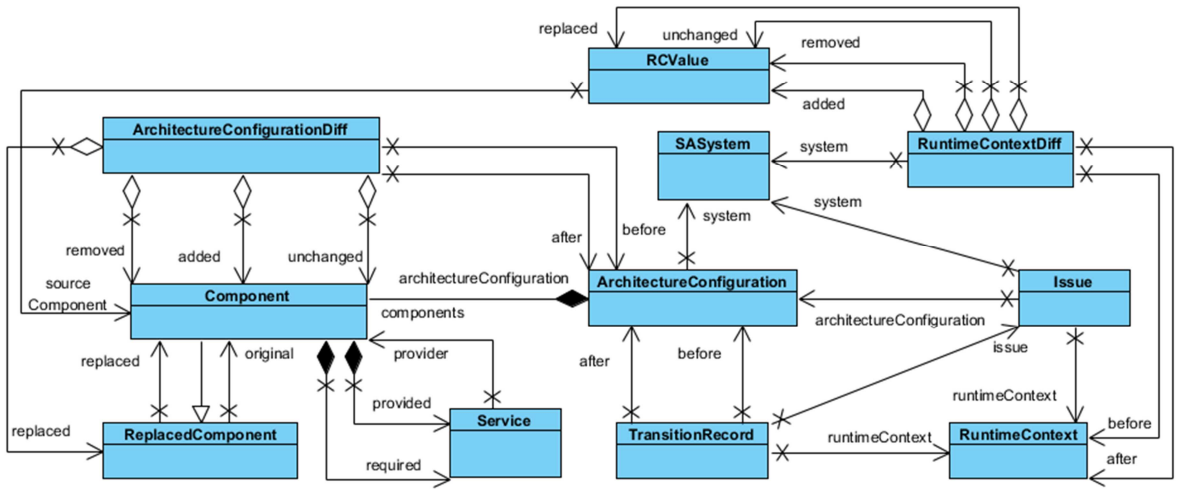


Figura 28 Diagrama das classes do pacote Model

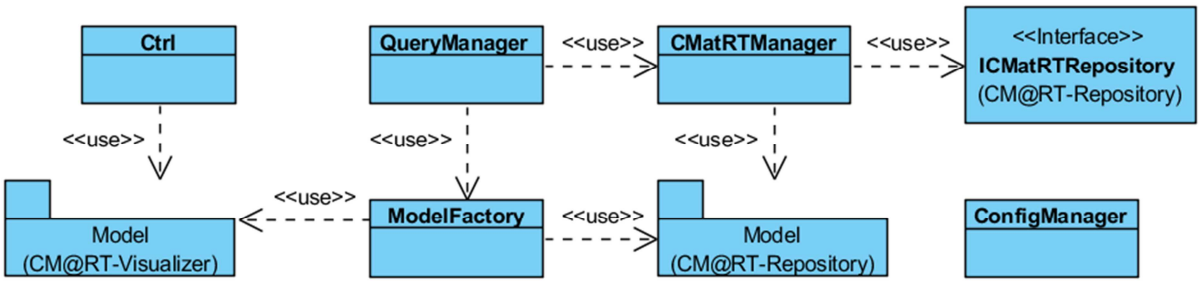


Figura 29 Diagrama de classes do pacote Ctrl

O pacote *View* detém a implementação das visualizações do CM@RT-Visualizer. A Figura 30 ilustra os subpacotes referentes as diferentes visualizações existentes no CM@RT-Visualizer. Estas visualizações utilizam sistematicamente o padrão de projeto *Observer* para atualizar seus conteúdos. Além disso, o padrão *Observer* é utilizado para a construção do mecanismo de monitoramento do CM@RT-Visualizer. Para isso, a aplicação observa eventos de *check-in* gerados pelo CM@RT-Repository. É importante dizer novamente, que a geração destes eventos depende do mecanismo de integração entre o CM@RT-Visualizer e o CM@RT-Repository, implementado pelo usuário da abordagem. Além destas características, algumas visualizações possuem grafos. Estes grafos são gerados com o auxílio da biblioteca Jung 2.0.1 [13] (O'MADADHAIN *et al.*, 2005), por meio de classes de apoio construídas no subpacote *graph* (Figura 30).

[13] http://jung.sourceforge.net/

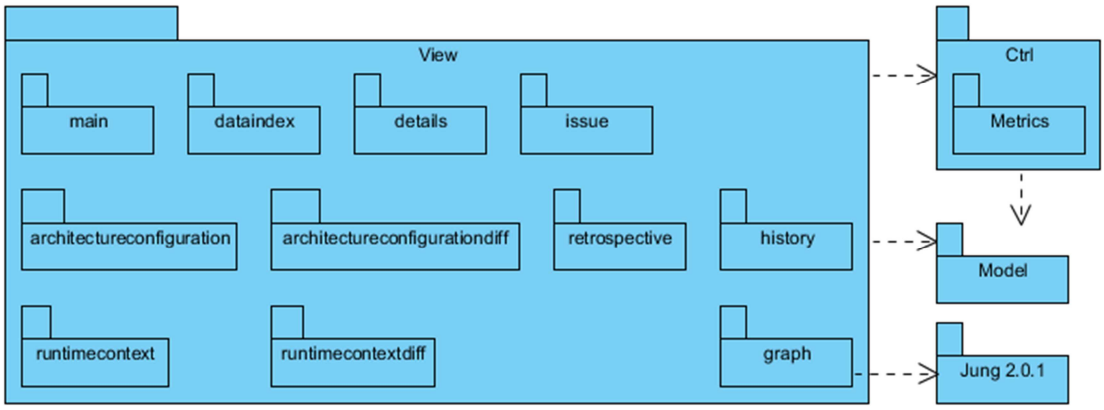


Figura 30 Diagrama de pacotes dos subpacotes de View

4.3.4 CM@RT-VISUALIZER BUNDLE

Assim como o módulo CM@RT-Repository *bundle*, o CM@RT-Visualizer *bundle* permite a integração do CM@RT-Visualizer à plataforma OSGi. Além disso, o CM@RT-Visualizer *bundle* realiza a comunicação e observação de eventos lançados pelo CM@RT-Repository, pelo do CM@RT-Repository *bundle*.

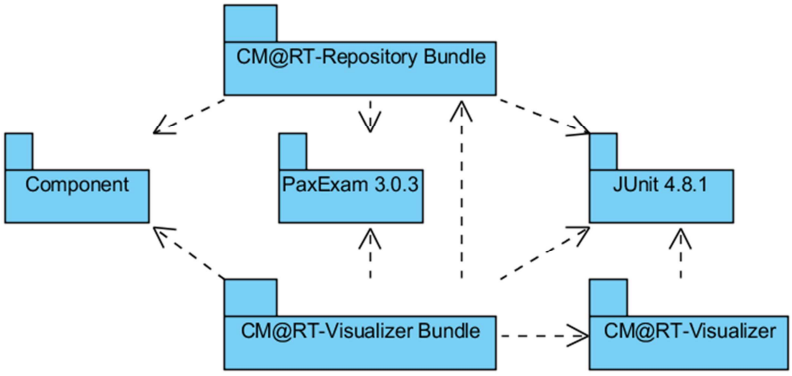


Figura 31 Arquitetura simplificada do bundle para o CM@RT-Visualizer

A comunicação entre o CM@RT-Visualizer *bundle* e o CM@RT-Repository *bundle* na plataforma OSGi ocorre por meio da interface *ICMatRTRepository*, definida na seção 4.3.1 e descrita anteriormente na seção 3.4.2. Serviços da plataforma OSGi realizam o *bindinging* dinâmico entre os *bundles*. Assim, quando se deseja utilizar o CM@RT-Visualizer *bundle* para análise, não é preciso realizar o *binding* explicitamente entre os módulos.

No momento em que o *binding* é efetivado, o CM@RT-Visualizer *bundle* também se torna observador dos eventos lançados pelo CM@RT-Repository *bundle*. Desta forma, o CM@RT-Visualizer é notificado sobre eventos de registro de SA, transições entre configurações arquiteturais, *issues* e contextos de execução.

4.4 FUNCIONAMENTO DO CM@RT-REPOSITORY

O CM@RT-Repository permite o *check-in*s simultâneo de diferentes SA inseridos em um mesmo ambiente de execução. No diagrama de atividades da Figura 15, está ilustrado o fluxo ideal das operações de *check-in*. Entretanto, existem fluxos alternativos possibilitados por um padrão recursivo implementado nas operações *check-in*. No início de cada operação de *check-in* é verificado se todas as entidades necessárias foram registradas anteriormente. Caso existam pendências, as operações necessárias são realizadas automaticamente. A seguir são descritas as verificações realizadas nas operações de *check-in* definidas na interface *ICMatRTRepository*, ilustradas anteriormente na Figura 22.

O *check-in* de uma instância de *SASystemInfo* verifica a unicidade da chave *hash* do SA junto ao repositório. Caso exista um SA com a mesma chave, o CM@RT-Repository assume que já houve o *check-in* e encerra o processo. Caso contrário, o SA é registrado.

O *check-in* de uma instância de *RuntimeContextInfo* verifica se o *SASystemInfo* associado está registrado. Se não estiver, é acionada a operação de *check-in* para *SASystemInfo*. Em seguida é verificado se a instância de *RuntimeContextInfo* possui ao menos uma variável de contexto. Em caso negativo é lançada uma exceção da classe *CM@RTException* (ver Figura 22). Em caso positivo, é realizado o registro da instância de *RuntimeContextInfo* e de seus dados associados. O padrão de salvamento se estende também aos dados associados, e foi suprimido por ser de importância menor.

O *check-in* de uma instância de *ArchitectureConfigurationInfo* verifica se o *SASystemInfo* associado está registrado. Se não estiver, é acionada a operação de *check-in* para *SASystemInfo*. Em seguida é verificada a unicidade da chave *hash* do *ArchitectureConfigurationInfo* junto ao repositório. Caso exista uma instância com a mesma chave, o CM@RT-Repository assume que já houve o *check-in* e encerra o processo. Caso contrário é realizado o registro da instancia de *ArchitectureConfigurationInfo* e de seus dados associados. O padrão de salvamento se estende também aos dados associados, e foi suprimido por ser de importância menor.

O *check-in* de uma instância de *IssueInfo* verifica se o *RuntimeContextInfo* associado está registrado. Se não estiver, é acionada a operação de *check-in* para *RuntimeContextInfo*. Em seguida é verificado se a instância de *ArchitectureConfigurationInfo* associada está registrada. Se não estiver, é acionada a operação de *check-in* para *ArchitectureConfigurationInfo.* Finalmente a instância de *IssueInfo* é registrada no repositório.

O *check-in* de uma instância de *TransitionRecordInfo* verifica primeiro se o *IssueInfo* associado esta registrado. Se não estiver, é acionada a operação de *check-in* para *IssueInfo*. Em seguida é verificado se a instância de *ArchitectureConfigurationInfo* apontada como resultante da transição entre configurações arquiteturais está registrada. Se não estiver, é acionada a operação de *check-in* para *ArchitectureConfigurationInfo*. Finalmente é registrada a instância de *TransitionRecordInfo* no repositório.

4.5 UTILIZAÇÃO DO CM@RT-VISUALIZER

O propósito do CM@RT-Visualizer é permitir a visualização dos dados registrados pelo CM@RT-Repository. Anteriormente, a seção 3.5.2 definiu os recursos disponíveis para utilização e a seção 3.5.3 o propósito principal de cada um deles. Esta seção apresenta as visualizações resultantes da implementação dos recursos citados.

A Figura 32 apresenta a tela inicial do sistema, ainda sem informações em exibição. As visualizações contidas na aplicação estão organizadas em dois grupos, em lados opostos. No lado esquerdo fica localizado o índice de dados, identificado pela letra A na Figura 32. Identificado pela letra B na Figura 32 está um painel destinado ao fornecimento de informações complementares. No lado direito estão agrupadas em um conjunto de abas as visualizações propriamente ditas da aplicação (identificado pela letra C na Figura 32). Para favorecer o espaço disponível a visualização dos dados em análise, todos os painéis podem ser redimensionados ou omitidos. As seções a seguir fornecem mais informações sobre as funcionalidades disponíveis e visualizações para realização de estudos.

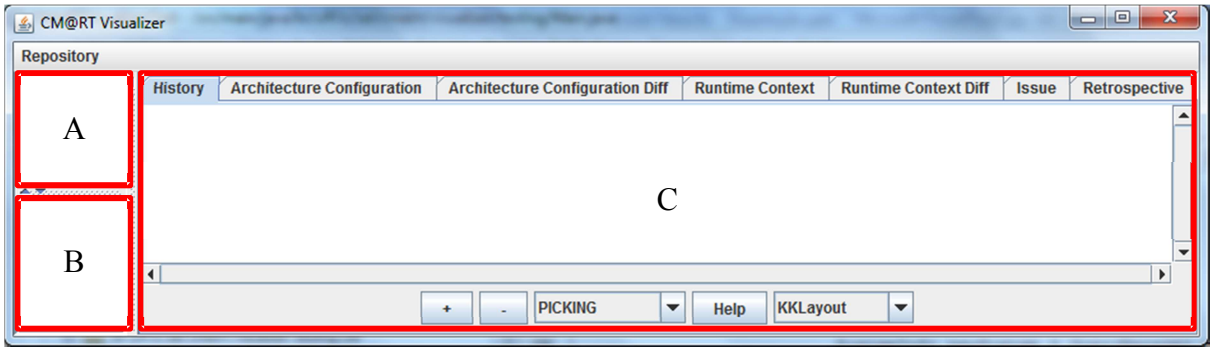


Figura 32 Tela inicial do CM@RT-Visualizer

4.5.1 SELEÇÃO DE DADOS PARA ANÁLISE

A seleção de um SA para estudo ocorre por meio do menu chamado *Repository*, mostrado na Figura 33. Uma vez acionado o menu, é exibido o item *Select system*. Quando este é acionado, é exibida a tela apresentada na Figura 34. A partir dela, um usuário pode escolher entre os SA disponíveis. Ao confirmar o SA desejado, a aplicação recupera as

informações necessárias do CM@RT-Repository e preenche as telas pertinentes do CM@RT-Visualizer. As telas preenchidas são o índice de dados e a tela de histórico, pois independem da seleção de dados específicos. As subseções seguintes fornecem descrições das telas em questão, bem como daquelas que necessitam de seleção.

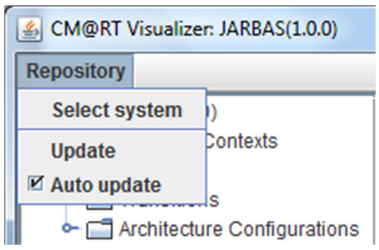


Figura 33 Menu Repository

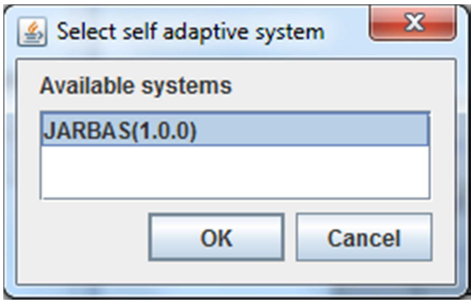


Figura 34 Tela de seleção de sistema para análise

A Figura 35 ilustra o resultado da seleção de um SA usado durante os testes da aplicação e descrito na seção 3.2. É nesta tela que são especificados os dados a serem visualizados no CM@RT-Visualizer. Os dados recuperados são organizados em uma hierarquia e representados em forma de árvore. O nó raiz representa o SA e possui quatro nós descendentes chamados: *Transitions*, *Architecture Configurations*, *Runtime Contexts* e *Issues*. Estes quatro grupos visam facilitar a localização da informação desejada para estudo.

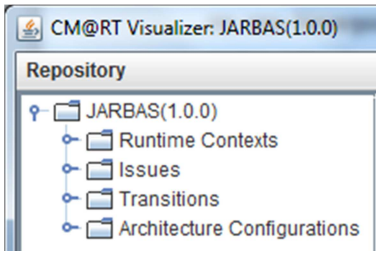


Figura 35 Índice de dados disponíveis

Ao expandir o nó *Transitions*, ficam visíveis as transições entre configurações arquiteturais registradas. Uma transição entre configurações arquiteturais é identificada por sua chave no repositório. Ao ser expandido o nó de uma transição entre configurações arquiteturais são exibidos quatro nós contendo: o contexto de execução, a razão para

adaptação, a configuração arquitetural de antes e a de depois da transição. A Figura 36 ilustra esta situação.

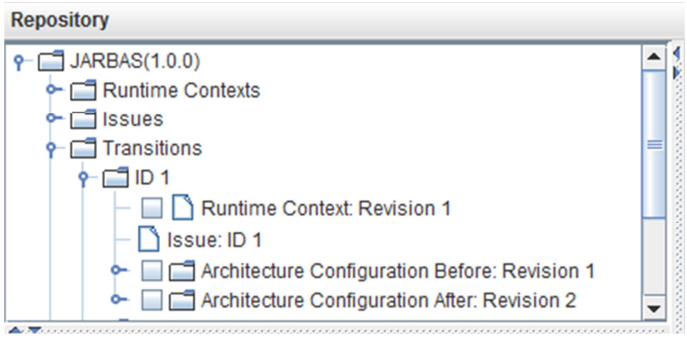


Figura 36 Nó contendo transições entre configurações arquiteturais

Ao expandir o nó *Architecture Configurations* ficam visíveis as configurações arquiteturais registradas. Uma configuração arquitetural é identificada por seu número de revisão. Ao ser expandido o nó de uma configuração arquitetural, em qualquer posição da árvore, é exibido seu conjunto de componentes. A Figura 37 ilustra esta situação para a primeira revisão do JARBAS, descrita na seção 3.3.1. O nó de um componente informa seu nome e sua versão entre parênteses. Ao ser expandido, o nó revela dois grupos referentes aos serviços: providos e requeridos. Cada serviço também é identificado por seu nome e versão entre parênteses.

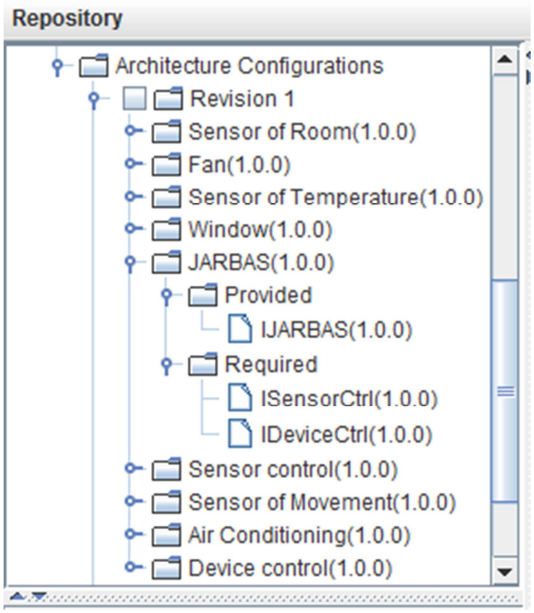


Figura 37 Nó contendo configurações arquiteturais

Ao expandir o nó *Runtime Contexts*, ficam visíveis os contextos de execução registrados. Um contexto de execução é identificado por seu número de revisão, como pode ser visto na Figura 38.

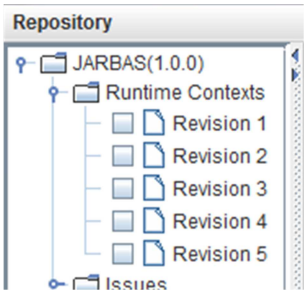


Figura 38 Nó contendo contextos de execução

Além do papel de organizador das informações, o índice de dados permite a seleção do elemento para análise usando outras visualizações da aplicação. As subseções 4.5.2, 4.5.3, 4.5.4 e 4.5.5 a seguir descrevem esta funcionalidade.

4.5.2 DESCRIÇÃO DE DADOS SELECIONADOS

Ao selecionar uma entidade no índice de dados é atualizado automaticamente o conteúdo do painel de informações, ilustrado no canto inferior da Figura 39. Este recurso permite ao usuário ter acesso a uma descrição da entidade selecionada, sem que sejam sobrecarregadas outras visualizações. Por exemplo, algumas métricas ou estatísticas podem ser fornecidas de acordo com a entidade em questão. A seguir serão descritas informações fornecidas para cada uma das entidades exibidas no índice de dados. As figuras fornecidas utilizam dados do exemplo motivacional da seção 3.2.

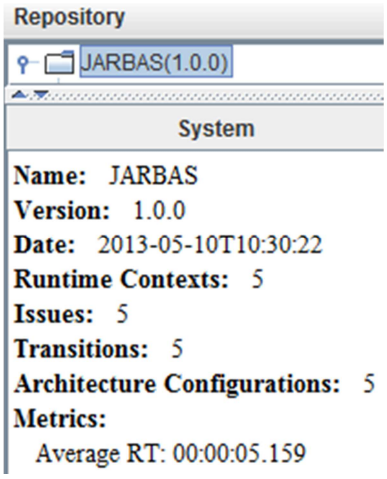


Figura 39 Descrição de um SA

Ao selecionar o nó referente ao SA, são exibidos os dados ilustrados na Figura 39. Além de seu nome e versão, são informadas a data de registro do SA no CM@RT-Repository, estatísticas e métricas sobre os dados armazenados. Por exemplo, é fornecida a métrica Tempo Médio de Resposta. Esta métrica calcula o tempo médio dos intervalos entre a data da atualização do contexto de execução e a transição arquitetural associada.

Ao selecionar um contexto de execução são exibidos os dados ilustrados na Figura 40. São fornecidas a data de registro do contexto de execução e a quantidade de variáveis de contexto de execução.

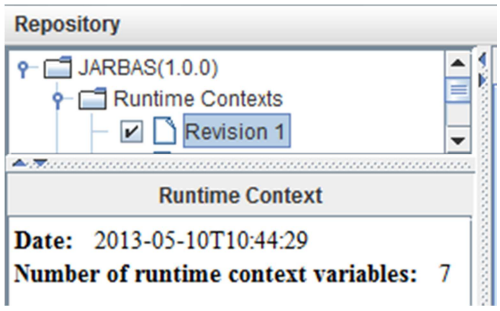


Figura 40 Descrição de um contexto de execução

Ao selecionar um *issue* são exibidas suas informações, como pode ser visto na Figura 41. A Figura 41 ilustra o *issue* do primeiro cenário do exemplo motivacional da seção 3.2, que adaptou o SA em vista da falta de movimento no ambiente. Esta decisão foi tomada para permitir o estudo destas informações juntamente com outras visualizações existentes. Por exemplo, é possível visualizar o *issue* e o contexto de execução de forma simultânea.

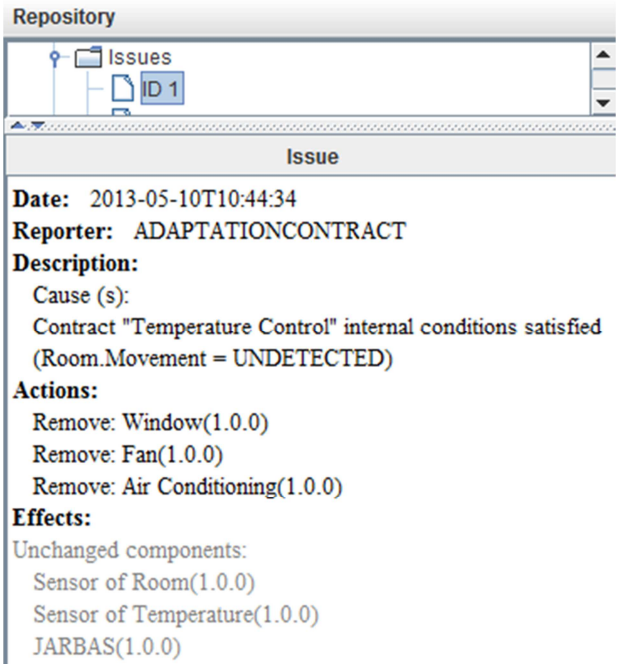


Figura 41 Descrição de um issue selecionado

Ao selecionar uma transição entre configurações arquiteturais são exibidas as informações ilustradas na Figura 42. Além da data de registro da transição, são fornecidos os identificadores das informações pertinentes a ela. Esta decisão foi tomada por não haver espaço suficiente ao fornecimento de informações mais complexas. Entretanto, os identificadores fornecidos permitem ao usuário localizar as informações que deseja estudar.

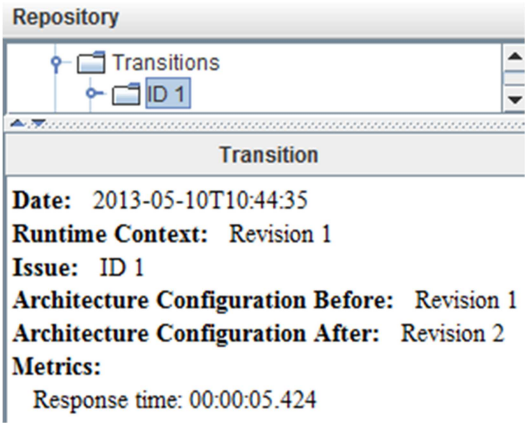


Figura 42 Descrição de uma transição entre configurações arquiteturais

Ao ser selecionada uma configuração arquitetural são fornecidos os dados exibidos na Figura 43. Além da data de registro da configuração arquitetural e seu número de revisão, são exibidas as métricas definidas anteriormente na seção 2.6.1.

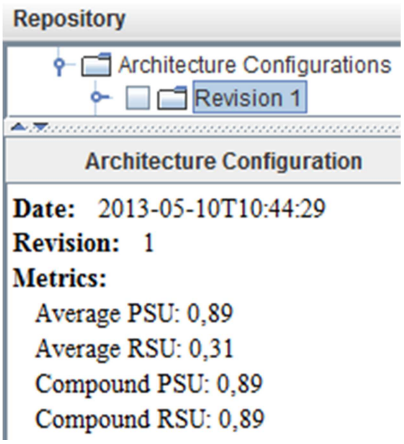


Figura 43 Descrição de uma configuração arquitetural

4.5.3 VISUALIZAÇÃO DE CONTEXTOS DE EXECUÇÃO

Existem duas alternativas de visualizações voltadas ao estudo do contexto de execução. A primeira exibe o contexto de execução propriamente dito, como ilustrado na Figura 44. Esta figura ilustra o contexto de execução exibido na Tabela 4. A segunda alternativa permite o estudo das diferenças existentes entre dois contextos de execução, como ilustrado na Figura 45. Esta figura ilustra o conteúdo da Tabela 7, resultante da comparação entre os dados da Tabela 4 e Tabela 5.

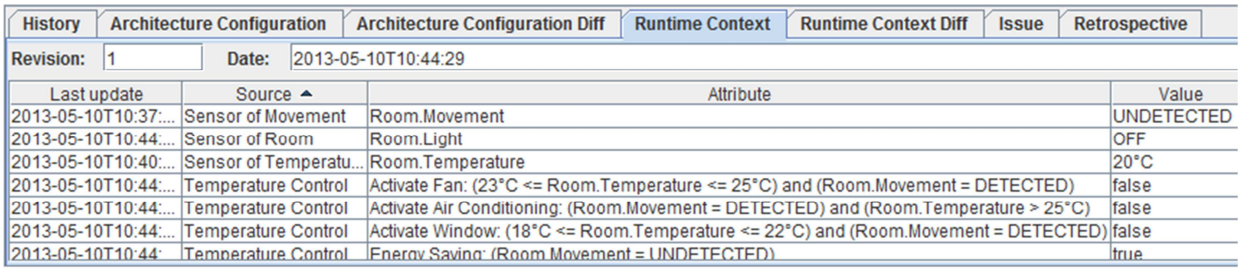


Figura 44 Visualização do contexto de execução

A visualização do contexto de execução é atualizada sempre que um deles é selecionado no índice de dados. Ela permite a identificação da data de registro da revisão do contexto de execução, assim como dos atributos de cada variável de contexto que o compõe. A visualização fornece ainda recursos de ordenação crescente ou decrescente por coluna e seleção para tornar o estudo dos dados mais eficiente e menos propensa a erros. O critério de ordenação depende do dado representado, podendo ser alfabética ou por data, por exemplo.

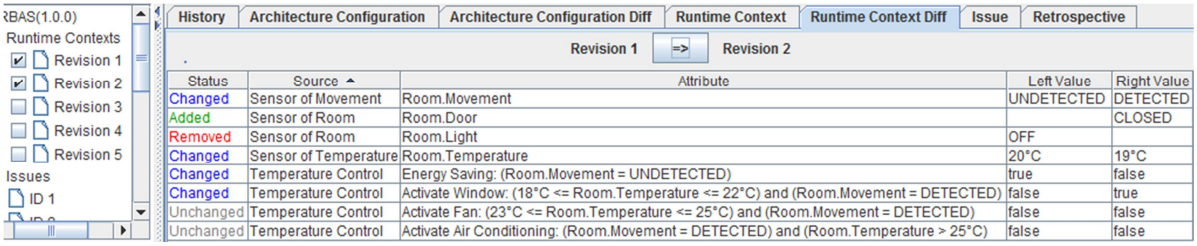


Figura 45 Visualização da comparação entre dois contextos de execução

A visualização de diferenças entre dois contextos de execução funciona de acordo com a descrição dada no capítulo anterior. Diferentes cores indicam eventuais modificações nas variáveis de contexto de execução. Quaisquer contextos de execução registrados podem ser selecionados para comparação empregando-se o índice de dados. A seleção é realizada por meio da marcação de dois contextos de execução, por meio de elementos de interface conhecidos como *checkbox*. A aplicação garante que apenas dois contextos de execução estejam marcados, removendo a marcação do mais antigo ao atingir a contagem de três elementos. Além disso, replicações do contexto de execução no índice de dados também são mantidas consistentes com as marcações. Ao serem marcados dois contextos de execução, o sistema atualiza a visualização para o novo resultado da comparação entre os mesmos. A visualização também permite a inversão no sentido de comparação, acionando o botão mostrado ao centro da parte superior da Figura 45, semelhante a uma seta para a direita. Este recurso visa facilitar a identificação das modificações existentes entre as revisões do contexto de execução.

4.5.4 VISUALIZAÇÃO DE ISSUES

A visualização de um *issue* exibe seus dados de forma semelhante a um relatório, como pode ser visto na Figura 46. A figura ilustra o *issue* representado anteriormente na Tabela 8. Neste relatório podem ser identificados: data, responsável, descrição e ações corretivas. Além disso, nos casos onde as ações corretivas foram realizadas pelo SA, seus efeitos sobre a configuração arquitetural são adicionados. Existem ainda dois botões que exibem a visualização do contexto de execução e a comparação entre a configuração arquitetural antes e após a adaptação.

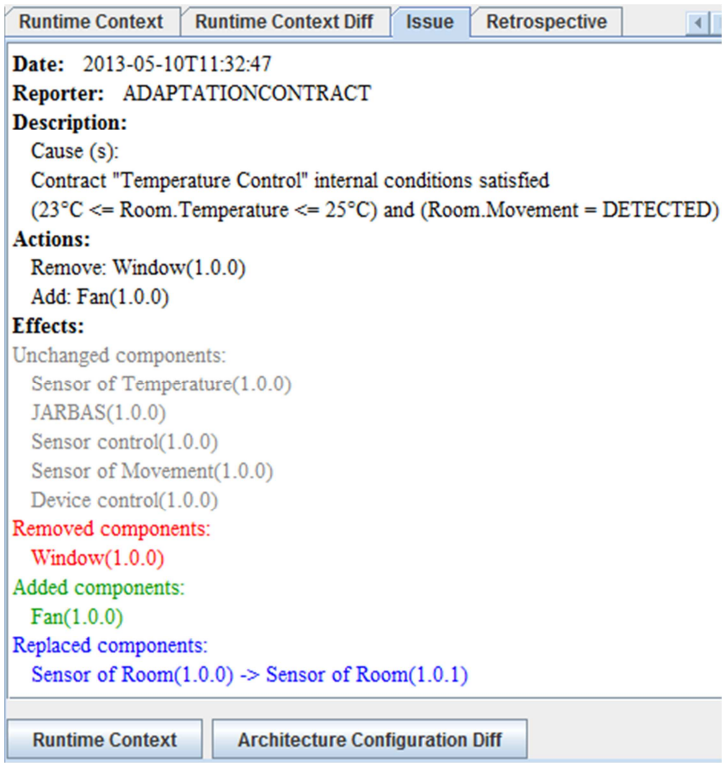


Figura 46 Visualização de um issue

4.5.5 CONFIGURAÇÃO ARQUITETURAL

Assim como no caso dos contextos de execução, existem duas alternativas de visualização voltadas ao estudo de configurações arquiteturais. A primeira exibe a configuração arquitetural, como ilustrado na Figura 47. Esta figura exibe a configuração arquitetural referente à Figura 17, descrita anteriormente por meio da mesma notação gráfica. A segunda permite a análise das diferenças existentes entre duas configurações arquiteturais, como ilustrado na Figura 48. A figura exibe a visualização referente à Figura 18, descrita anteriormente.

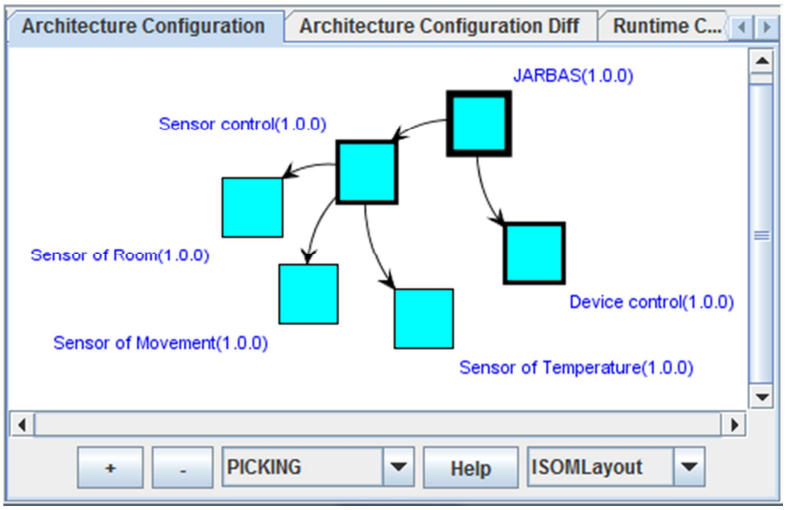


Figura 47 Visualização de uma configuração arquitetural

A visualização da configuração arquitetural é atualizada sempre que uma delas é selecionada. Ela permite a visualização da topologia e composição da configuração arquitetural por meio de um grafo. Para diminuir a poluição visual, os componentes são identificados por seu nome e versão entre parênteses. A topologia da configuração arquitetural é representada empregando-se arestas direcionadas, no sentido provedor do serviço requerido pelo componente de origem. A visualização fornece ainda recursos de manipulação do grafo, como reposicionamento manual ou por meio de algoritmos da biblioteca Jung.

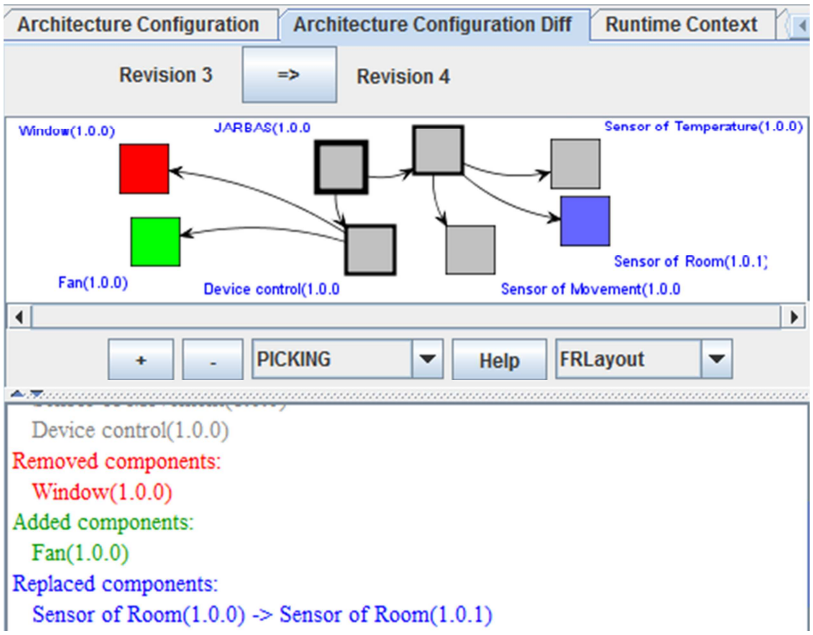


Figura 48 Visualização da comparação entre configurações arquiteturais

A visualização de diferenças entre duas configurações arquiteturais funciona de acordo com a descrição dada no capítulo anterior. Para visualizar as diferenças é preciso marcar duas configurações arquiteturais por meio do índice de dados. Ao serem marcadas duas configurações arquiteturais, suas diferenças são representadas por meio de um grafo na visualização. Cores são utilizadas para sinalizar as diferenças entre as configurações arquiteturais. Além disso, uma descrição textual das diferenças está disponível na parte inferior da interface de visualização. A visualização também permite a inversão no sentido de comparação, por meio do botão semelhante a uma seta para a direita.

4.5.6 TRANSIÇÕES ENTRE CONFIGURAÇÕES ARQUITETURAIS

A partir das transições entre configurações arquiteturais foram desenvolvidas duas visualizações. A primeira é chamada de Histórico e é exibida na Figura 49. Esta figura é referente à Figura 19, descrita anteriormente. Ela representa de forma estática a evolução do SA em termos das revisões de configurações arquiteturais assumidas ao longo do tempo. A segunda é chamada de Retrospectiva e é exibida na Figura 50. Esta figura é referente à Figura 20, descrita anteriormente. Ela representa de forma dinâmica a evolução das modificações no SA ao longo do tempo.

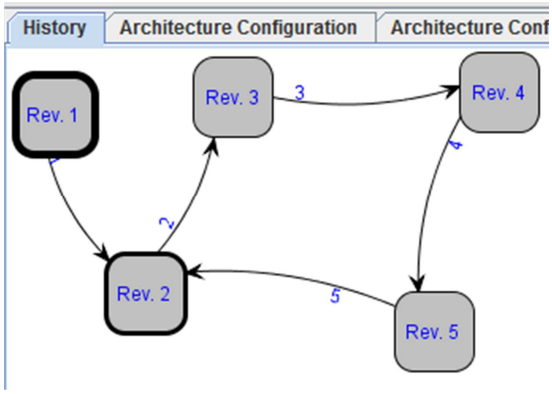


Figura 49 Visualização do histórico de transições arquiteturais

O grafo do histórico possui vértices que representam a revisões de configurações arquiteturais e arestas que representam transições entre elas. A sequência das transições entre configurações arquiteturais está representada por meio de índices numéricos juntos às arestas. Esta visualização permite a realização de estudos sobre o comportamento do SA em um nível de abstração mais elevado que o da visualização Retrospectiva. Por exemplo, é possível identificar a formação de ciclos de adaptação, conforme representado entre a revisão 2 e 3 da Figura 49.

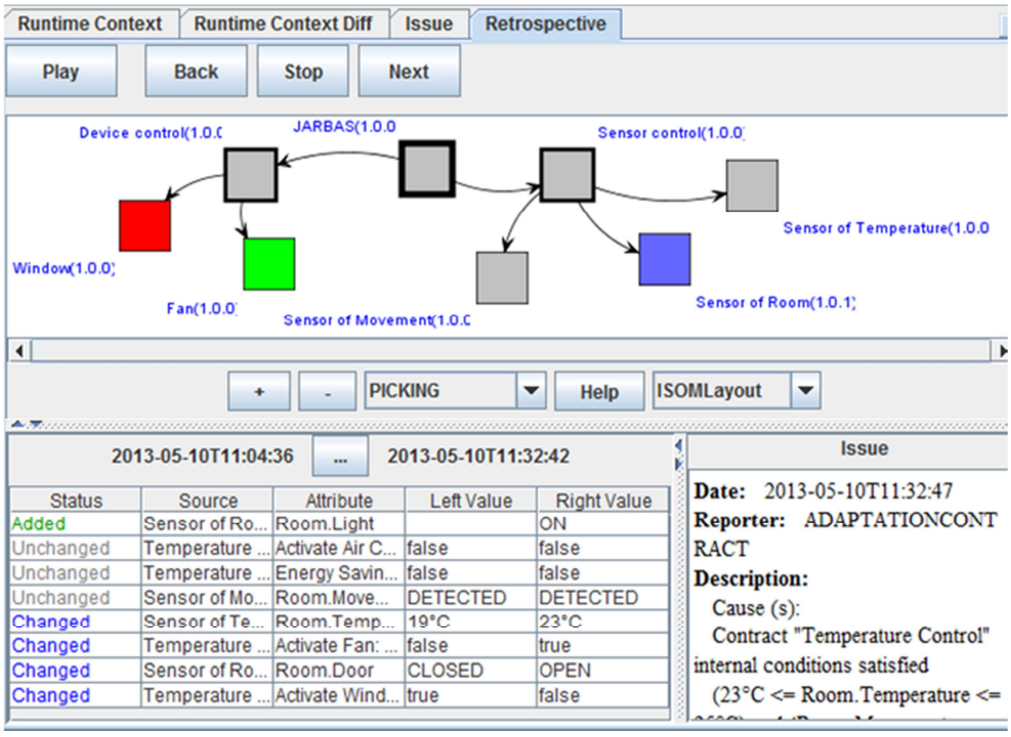


Figura 50 Visualização da retrospectiva das modificações na configuração arquitetural

A visualização denominada Retrospectiva gera uma animação, onde podem ser vistas as modificações realizadas na configuração arquitetural e no contexto de execução. Além disso, são exibidos os *issues* que motivaram as modificações realizadas. A animação pode ser

realizada em passos ou continuamente, utilizando-se controles dispostos na parte superior da visualização. A qualquer instante é possível paralisar a animação e continuar.

4.6 CONSIDERAÇÕES FINAIS

O protótipo construído para o registro de dados permite o armazenamento das informações necessárias à realização de estudos sobre a evolução de um SA. Além disso, o protótipo CM@RT-Repository possui características que favorecem sua confiabilidade e flexibilidade. O protótipo construído para a visualização de dados, o CM@RT-Visualizer, fornece uma gama variada de ferramentas. Estas ferramentas visam possibilitar a interpretação dos dados provenientes do CM@RT-Repository com o auxílio de visualizações que consideram a semântica da informação representada.

No entanto, os protótipos são um esforço inicial em face das possibilidades criadas pela abordagem. Estes protótipos não exploram questões avançadas como escalabilidade das visualizações e recursos de análise automatizada dos dados registrados. Estes e outros aspectos serão abordados em futuras extensões deste trabalho.

O Capítulo 5 descreve como foi realizada a avaliação desta abordagem, a fim de determinar seus benefícios. Neste capítulo é avaliada a capacidade do conjunto de protótipos construídos em prover respostas às questões de competência de SA, descritas no Capítulo 2. No capítulo seguinte, Capítulo 6, são expostas as conclusões sobre o trabalho realizado, suas limitações, contribuições e trabalhos futuros.

CAPÍTULO 5 – AVALIAÇÃO

5.1 INTRODUÇÃO

A abordagem CM@RT tem como objetivo principal permitir o estudo da evolução dinâmica de SA, fornecendo respostas a perguntas fundamentais nas informações 5W+1H. Acredita-se que estes estudos contribuirão para a solução de diversos desafios da área de SA, reduzindo a complexidade de sua manutenção e aumentando sua confiabilidade. O capítulo corrente avalia como a abordagem pode ser utilizada para estudo da evolução dinâmica de um SA, respondendo a questões de interesse baseadas em diferentes artigos da área.

O SA alvo da avaliação denomina-se SCIADS (CARVALHO; COPETTI; LOQUES, 2011; CARVALHO; MURTA; LOQUES, 2012). O SCIADS é um sistema crítico voltado à prestação de serviços de apoio a saúde em ambiente residencial. Entre suas características está o fato de ser construído sob o paradigma de LPSD e ter suas adaptações definidas por contratos de adaptação. Além disso, o SCIADS é um sistema acadêmico, logo em constante estado de evolução. Entre as motivações para sua evolução podem ser incluídos novos interesses de pesquisa e exploração de novas tecnologias.

Recentemente, os benefícios da plataforma OSGi motivaram o desenvolvimento de uma versão compatível do SCIADS, chamada SA-SCIADS (do inglês, *Self-Adaptive* SCIADS). A arquitetura do SA-SCIADS é constituída por dois grupos de componentes. O primeiro forma um framework para construção de SA na plataforma OSGi, chamado SAF (do inglês, *Self-Adaptive Framework*). O segundo constitui o SCIADS propriamente dito, um SA guiado por contratos de adaptação, descritos em CARVALHO, MURTA e LOQUES (2011). Para a realização da avaliação, foram conectados a esta arquitetura dois componentes responsáveis por integrar o CM@RT-Repository e o CM@RT-Visualizer à plataforma OSGi.

Com o auxílio destes protótipos, os benefícios da abordagem são demonstrados mediante sua aplicação em três cenários de adaptação extraídos de CARVALHO, LOQUES e MURTA (2011). Para cada um destes cenários são descritos três aspectos: a porção da configuração arquitetural do SA-SCIADS relevante ao cenário, questões de interesse a serem respondidas e como obter estas respostas utilizando-se a abordagem.

Além desta seção introdutória, este capítulo possui outras cinco subseções. A subseção 5.2 descreve o sistema SCIADS, SA alvo dos estudos realizados. A subseção 5.3 descreve os protótipos utilizados para a avaliação; a subseção 5.4 descreve a avaliação da abordagem. A subseção 5.5 aponta ameaças ao estudo e a subseção 5.6 discute as considerações finais.

5.2 SCIADS

O SCIADS é um sistema ubíquo pervasivo de apoio à saúde no ambiente residencial, capaz de interligar o paciente aos provedores de serviços médicos que lhe monitoram remotamente. Seus objetivos principais são o monitoramento do paciente em suas atividades diárias e de seus dados fisiológicos. Com base nestas informações, o sistema realiza atividades que visam promover a aderência do paciente a seu tratamento e garantir atendimento adequado em caso de situações de emergência. A promoção da aderência ocorre principalmente por um recurso de notificações sobre atividades do tratamento em seus respectivos horários. Enquanto que a prestação de atendimento de emergência provém do suporte a detecção de alterações significativas nos dados fisiológicos, que resultam em notificação aos serviços médicos adequados.

A Figura 51 mostra uma visão geral da arquitetura do SCIADS. O SCIADS é composto de duas aplicações distintas: a Central de Supervisão Médica (CSM) e a Central de Saúde Residencial (CSR). A CSM é utilizada pelos prestadores de serviço médico para atendimento remoto às necessidades do paciente, como, por exemplo, realizar uma teleconferência entre paciente e médico para fim de consulta de acompanhamento. A CSR é responsável pela interação com o paciente, bem como por seu monitoramento e da residência. A interação com o paciente ocorre por meio de diversos dispositivos, como, por exemplo, TV digital (ou DiTV, por simplicidade), celular e monitores dedicados. O monitoramento do paciente é realizado por outra gama de dispositivos como, por exemplo, sensores fisiológicos, de movimento e de queda (MARELI, 2011). O monitoramento da residência é realizado utilizando-se sensores de ambiente, como, por exemplo, temperatura, incêndio e presença.

Os dados frutos do monitoramento são enviados ao módulo de Análise e Decisão, que determina o quadro sintomático do paciente continuamente. Este módulo seria o responsável por detectar situações de emergência e realizar a ações necessárias no sistema. O Plano de Cuidados é responsável pelo controle do tratamento do paciente. É o Plano de Cuidados que gera as notificações ao paciente sobre sua medicação e realização de medições fisiológicas.

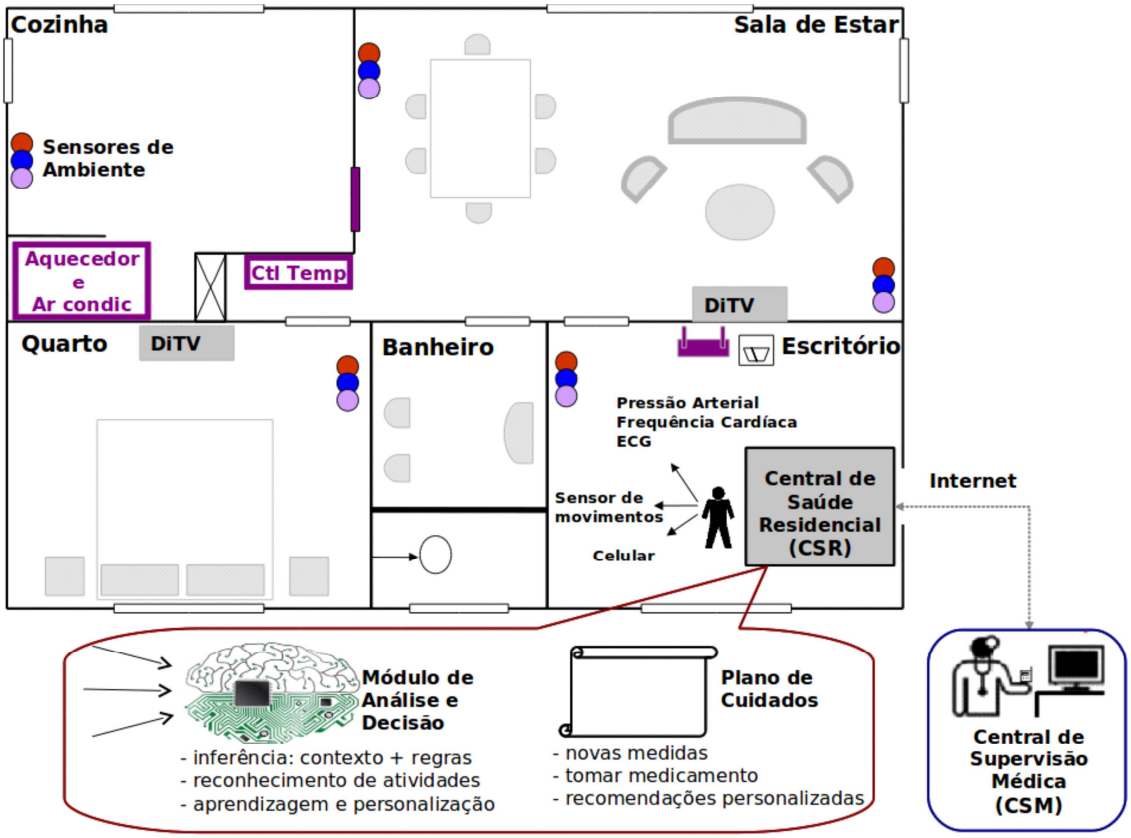


Figura 51 Visão geral da arquitetura do SCIADS (retirada de CARVALHO et al. (2011))

Além destas características, o SCIADS é construído segundo o paradigma de LPSD. Entre outras vantagens, esta característica favorece o suporte à evolução do quadro clínico de seus usuários (CARVALHO; MURTA; LOQUES, 2012). Por exemplo, um paciente hipertenso necessita obrigatoriamente de um medidor de pressão disponível, o que pode ser especificado por meio de regras de composição da LPSD. Um conjunto de regras de composição forma um perfil, que pode refletir a necessidade de um determinado quadro clínico. Além disso, uma LPSD dá suporte à evolução do quadro clínico dos pacientes mesmo após a implantação do sistema em suas residências. No caso de um paciente evoluir de um quadro hipertensivo para insuficiência cardíaca (CARVALHO; MURTA; LOQUES, 2012), o novo perfil reflete em um novo produto a ser derivado da LPSD. Este processo de derivação deve ocorrer em tempo de execução, considerando inclusive os dados fisiológicos do usuário antes de realizar as alterações necessárias no produto.

O LPSD é desenvolvido pelo laboratório Tempo [14], que trabalha continuamente na evolução do sistema. Entre as motivações desta evolução contínua está o desenvolvimento de pesquisas e o emprego de novas tecnologias que favoreçam a utilização da aplicação. Estão

[14] http://www.tempo.uff.br/

em desenvolvimento dois novos protótipos do SCIADS, sendo uma versão voltada ao ambiente Android e outra para a plataforma OSGi. Esta última se chama SA-SCIADS [15] e foi utilizada para a realização da avaliação da abordagem CM@RT.

A arquitetura do SCIADS foi apresentada em trabalhos anteriores (CARVALHO; COPETTI; LOQUES, 2011; CARVALHO; MURTA; LOQUES, 2012). A Figura 52 mostra esta arquitetura, onde podem ser identificados três componentes: o Configurador, o Gerenciador de Contratos e o Serviço de Contexto. O Configurador é responsável por modificar a configuração arquitetural do produto. O Gerenciador de Contratos interpreta os contratos de adaptação e o Serviço de Contexto fornece o contexto de execução ao Gerenciador de Contratos. Estes três componentes formam o mecanismo para autoadaptação do SCIADS, que trabalha em tempo de execução seguindo o fluxo de processamento definido pelas setas.

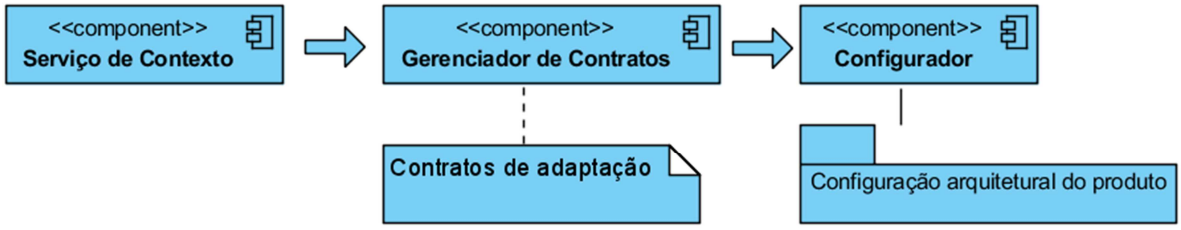


Figura 52 Arquitetura do mecanismo de autoadaptação do SCIADS (adaptada de CARVALHO, MURTA e LOQUES (2011))

As adaptações realizadas pelo mecanismo de autoadaptação no SCIADS ocorrem de acordo com contratos de adaptação. A Figura 53 ilustra o metamodelo mais recente dos contratos de adaptação, descrito em CARVALHO, MURTA e LOQUES (2012). Não é objetivo deste trabalho se aprofundar em contratos de adaptação, principalmente por sua complexidade. Ademais, em termos gerais, um contrato define um conjunto de ações de adaptação a serem realizadas quando regras baseadas no contexto de execução assim as determinam. Estas ações de adaptação atuam no nível de componentes da LPSD, sobre pontos de variabilidade previstos na linha. A verificação dos contratos de adaptação ocorre em função do monitoramento do contexto de execução. Estes contratos podem ser ativados ou desativados pelos arquitetos de software, de modo a atender a diferentes produtos da LPSD. Além disso, os contratos de adaptação podem ser utilizados para diferentes fins. Por exemplo, para garantia da Qualidade de Serviço (do inglês, *Quality of* Service (QoS)), personalização

[15] O autor deste trabalho é participante do desenvolvimento do SA-SCIADS desde seu início.

do comportamento do sistema e adequação a evolução do quadro clínico do paciente. Exemplos de contratos de adaptação serão fornecidos na seção 5.4.

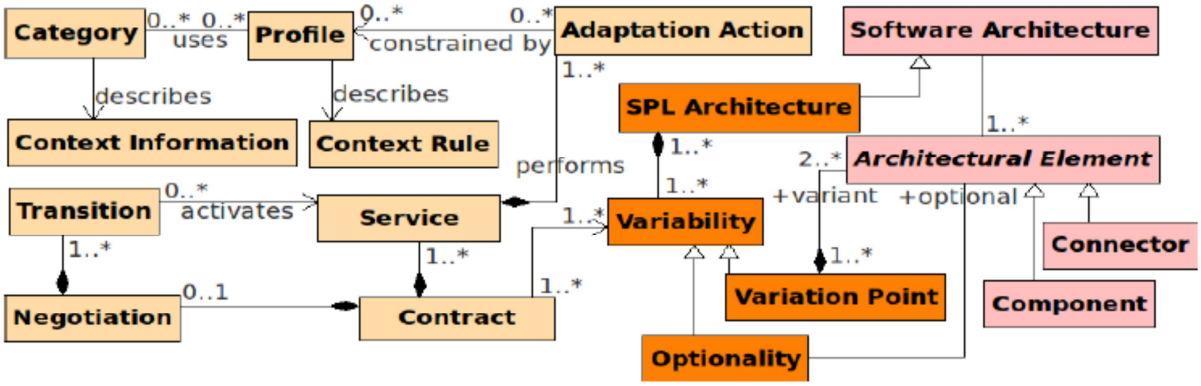


Figura 53 Metamodelo de um contrato de adaptação (retirado de CARVALHO, MURTA e LOQUES (2012))

5.3 PROTÓTIPOS UTILIZADOS

Os protótipos descritos a seguir foram desenvolvidos para operar na plataforma OSGi, que favorece a construção de aplicações dinâmicas em linguagem Java. Os protótipos estão organizados em dois grupos. O primeiro grupo está relacionado ao SA-SCIADS e o segundo à abordagem CM@RT.

As subseções a seguir descrevem estes dois grupos. A subseção 5.3.1 descreve a infraestrutura de suporte ao SA-SCIADS. A subseção 5.3.2 descreve os protótipos para a integração da abordagem CM@RT à plataforma OSGi e ao SA-SCIADS.

5.3.1 SA-SCIADS

Assim como o SCIADS, o SA-SCIADS é adaptado por um mecanismo externo fornecido por um *framework*. Este *framework* é destinado à construção de SA em linguagem Java e também foi desenvolvido pelo laboratório Tempo. Ele se chama *Self-Adapter Framework* (SAF), tendo entre suas características o uso do ciclo MAPE-K como modelo de adaptação e por deixar em aberto o ambiente de execução em que opera. Para isso, o SAF define um conjunto de interfaces que refletem as fases do ciclo MAPE-K, classes que modelam resultados gerados por cada uma destas fases e um conjunto de recursos de apoio. Para utilizar o SAF, um SA deve se registrar na base de conhecimento e fornecer uma série de dados a seu respeito. Entre estes dados estão o conjunto de contratos de adaptação, identificadores de seus sensores e um filtro que permite identificar seus componentes entre os demais no ambiente de operação. Com estas informações, o SAF inicia o monitoramento dos

sensores e analisa mudanças significativas no contexto de execução frente aos contratos de adaptação fornecidos.

Para a realização da avaliação do CM@RT, o SAF foi implementado para a plataforma OSGi. A Figura 54 ilustra a arquitetura implementada do SAF, onde as setas pontilhadas indicam relação de dependência direta. Nesta figura podem ser identificados seis componentes. O componente principal é chamado SAF, que é um *bundle* OSGi com objetivo de fornecer aos demais as interfaces e modelos públicos do *framework*. O componente *Monitor* realiza o monitoramento dos sensores cadastrados e gera eventos em caso de mudanças significativas. O componente *Knowledge* captura estes eventos e lança eventos de atualização no contexto de execução em caso de alteração entre os valores capturados e aqueles armazenados. O componente *Analyzer* captura os eventos de atualização no contexto e verifica os contratos de adaptação. Em caso de necessidade de adaptação é gerado um evento de solicitação de adaptação. O componente *Planner* captura este evento, processa e agenda sua execução em momento oportuno. Quando uma adaptação puder ser realizada, o *Planner* utiliza o componente *Executer* para efetivá-la. No final de uma adaptação, é lançado evento de conclusão do processo.

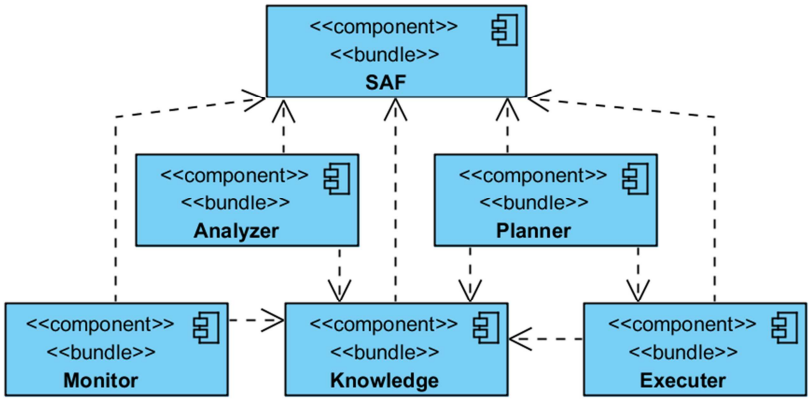


Figura 54 Arquitetura do SAF

Além de armazenar o contexto de execução e os contratos de um SA, a base de conhecimento também mantém uma representação atualizada da configuração arquitetural do sistema. Quando o *Analyzer* avalia os contratos de adaptação, esta representação é necessária ao processo. O *Executer* também a utiliza para verificar se a adaptação pode ser realizada sobre a configuração arquitetural vigente. Para manter a representação da configuração arquitetural sempre atualizada, o *Knowledge* monitora eventos de falha de componentes e de conclusão de adaptações.

A arquitetura do SA-SCIADS foi baseada nas apresentadas em trabalhos anteriores (CARVALHO; COPETTI; LOQUES, 2011; CARVALHO; MURTA; LOQUES, 2012). A

Figura 55 ilustra a arquitetura completa do SCIADS. A descrição do papel de cada componente é fornecida na seção 5.4. É importante dizer que a implementação desta arquitetura adicionou novos componentes ao SCIADS. Por exemplo, foi criado um componente responsável pelo gerenciamento de dispositivos integrados ao sistema. Esta e outras modificações não alteram o comportamento do SCIADS e suas motivações são descritas também na seção 5.4.

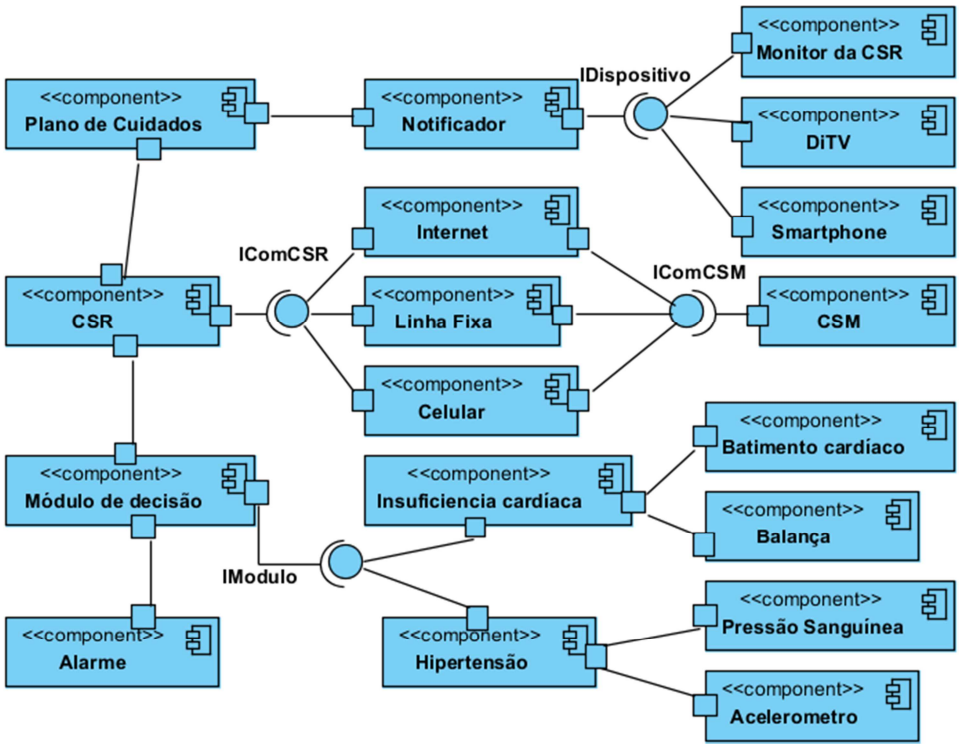


Figura 55 Arquitetura de LPSD completa do SCIADS (adaptada de CARVALHO, MURTA e LOQUES (2011))

5.3.2 INTEGRAÇÃO DO CM@RT

A abordagem CM@RT foi integrada ao SAF para permitir sua avaliação. Para isso, o CM@RT-Repository e o CM@RT-Visualizer foram encapsulados em *bundles* OSGi. A Figura 56 ilustra como foi feita esta integração.

É possível identificar na Figura 56 o *bundle* do CM@RT-Repository. Ele monitora eventos de diferentes componentes para registrar a evolução do SA. Por exemplo, eventos de registro de SA resultam na criação de repositórios para no CM@RT-Repository. Eventos de solicitação de adaptação são registrados como *issues*. Eventos de adaptação são registrados como transições entre configurações arquiteturais. Também são registradas atualizações no contexto de execução e transições arquiteturais por falha de componentes. Além destas tarefas, o *bundle* CM@RT-Repository lança seus próprios eventos e publica seus serviços.

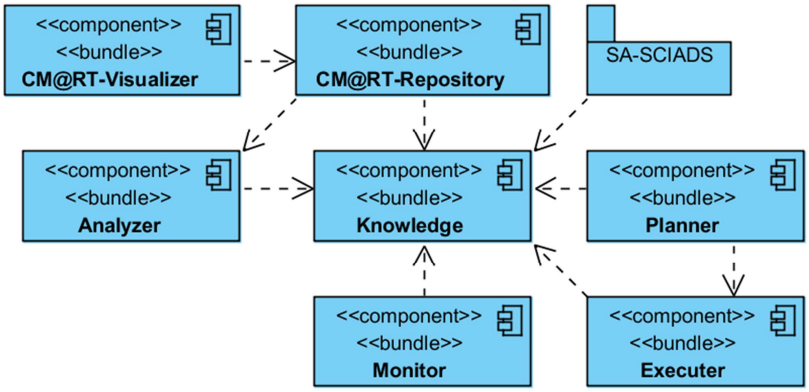


Figura 56 Integração entre SAF e CM@RT

O *bundle* do CM@RT-Visualizer, por sua vez, monitora os eventos do CM@RT-Repository e utiliza seus serviços para realizar suas operações. Utilizando eventos como *check-in* de SA e *check-in* de contexto de execução, o CM@RT-Visualizer atualiza suas visualizações. Os serviços de consulta disponibilizados pelo CM@RT-Repository são utilizados para popular as visualizações do CM@RT-Visualizer.

5.4 UTILIZAÇÃO DO CM@RT NO SA-SCIADS

A avaliação dos benefícios provenientes da abordagem CM@RT foi realizada por meio de sua utilização para estudo da evolução do SCIADS em três cenários sintéticos originalmente descritos em CARVALHO, MURTA e LOQUES (2011). Cada cenário envolve elementos específicos da configuração arquitetural sendo modificados por contratos de adaptação distintos. Para cada um dos cenários são propostas questões de interesse com origem nas informações 5W+1H, tanto genéricas quanto específicas ao SCIADS. As respostas a estas questões permitem a realização de estudos sobre a evolução dinâmica do SA (SALEHIE; TAHVILDARI, 2009). Estes estudos servem tanto às atividades de monitoramento da operação, como para posterior auditoria das decisões do SA.

Cada um dos cenários tem descrita sua arquitetura de LPSD e a configuração arquitetural que a implementa na plataforma OSGi. Todos os componentes das configurações arquiteturais descritos foram construídos como *bundles* OSGi. Para simplificar os diagramas necessários, foram omitidos componentes não relevantes aos cenários e o estereótipo <<*bundle*>>.

As subseções a seguir descrevem a utilização do CM@RT em três cenários. A subseção 5.4.1 descreve um cenário focado no serviço de comunicação. A subseção 5.4.2 descreve um cenário voltado à realização de notificações ao paciente. A subseção 5.4.3 descreve um cenário onde ocorre evolução no quadro clínico do paciente.

5.4.1 CENÁRIO 1: SERVIÇO DE COMUNICAÇÃO

O serviço de comunicação entre a CSR e CSM é de importância significativa para o SCIADS. Por exemplo, em caso de uma situação de emergência envolvendo o paciente, este serviço é necessário para comunicar aos médicos responsáveis o ocorrido. Para favorecer a tolerância a falhas, a arquitetura da LPSD do SCIADS prevê três alternativas de serviços de comunicação. A Figura 57 ilustra o trecho da arquitetura em questão e permite identificar os três serviços, a saber, Internet, Linha Fixa e Celular.

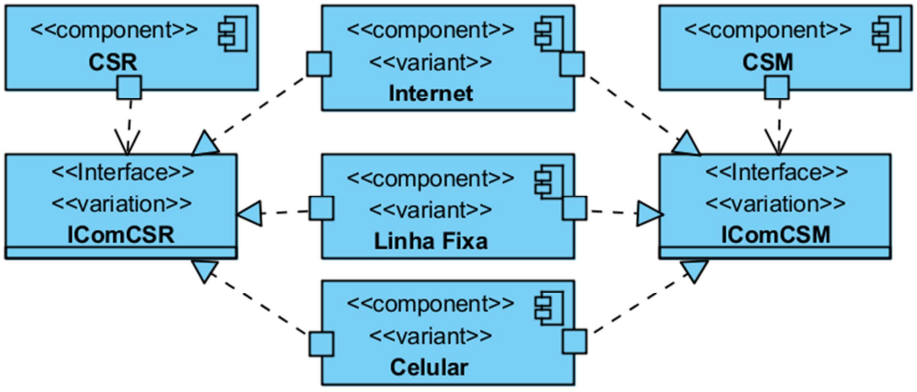


Figura 57 Arquitetura da LPSD para comunicação (adaptada de CARVALHO, MURTA e LOQUES (2011))

Em caso de falha do serviço em uso, um contrato de adaptação prevê qual será a alternativa utilizada pelo SCIADS. O contrato define a seguinte ordem de preferência: Internet > Linha Fixa > Celular. Assim, o SCIADS utiliza a Internet preferencialmente e alterna para Linha Fixa em caso de falha no primeiro. Em caso de falha na linha fixa, o SCIADS alterna para Celular. Finalmente, em caso de falha no Celular, o SCIADS fica incomunicável e o paciente em risco de morte.

A subseção 5.4.1.1 descreve a configuração arquitetural que implementa a arquitetura definida na Figura 57 e o contrato de adaptação relacionado. A subseção 5.4.1.2 descreve as perguntas de interesse motivadas por este cenário e a subseção 5.4.1.3 descreve como o CM@RT é capaz de responder a estas perguntas.

5.4.1.1 ARQUITETURA E CONTRATO DE ADAPTAÇÃO

A Figura 58 ilustra os componentes que concretizam a arquitetura definida na seção anterior para a comunicação entre CSR e CSM. Nela é possível identificar 7 componentes envolvidos diretamente na realização de comunicação entre as duas aplicações. O componente chamado Serviço de Comunicação é responsável por utilizar um dos três serviços de transmissão disponíveis. Estes serviços de transmissão são fornecidos pelos componentes

chamados Internet, Linha Fixa e Celular. Cada um deles é responsável por adequar as necessidades de comunicação do Serviço de Comunicação às limitações de seus provedores. Os componentes chamados de Provedor Internet, Provedor Linha Fixa e Provedor Celular, são responsáveis por serviços de baixo nível para a transmissão de dados.

Para controlar a disponibilidade de serviços de comunicação adequados, o contrato de adaptação mencionado no início da seção 5.4.1 foi implementado de forma que, de acordo com o estado da conexão de cada um dos provedores de serviço no contexto de execução, o contrato determina qual componente de transmissão estará instalado na plataforma OSGi. O estado da conexão é atualizado no contexto de execução pelos componentes provedores, por isso os mesmos não são retirados da plataforma.

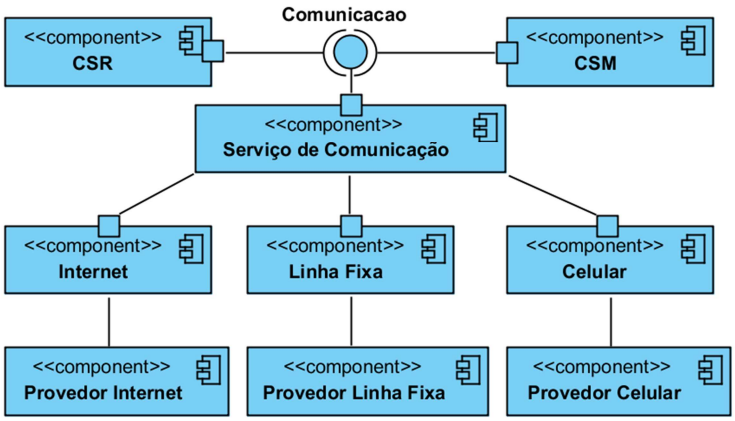


Figura 58 Implementação da arquitetura de LPSD do cenário 1

5.4.1.2 QUESTÕES DE INTERESSE

As perguntas aqui descritas demonstram como observar a confiabilidade de componentes do SA e de seu comportamento. Estas perguntas foram motivadas pelo trabalho de GEORGAS, VAN DER HOEK e TAYLOR (2005).

1) **O sistema está sujeito a que frequência de adaptações por falha na conexão?**

Acredita-se que esta resposta varie significativamente entre residências de pacientes. Esta questão é uma fonte de incerteza para o SA, que pode afetar significativamente seu desempenho. Responder a ela permite aos desenvolvedores melhorar o desempenho da aplicação. Arquitetos de software podem identificar pontos de correção nos contratos que regem as adaptações. Além disso, auditores podem determinar o real desempenho do sistema e dos serviços de que dependem.

Para responder a esta pergunta, é preciso fornecer meios de contabilizar as ocorrências de falha na conexão dos provedores disponíveis e medir o intervalo de tempo entre elas. Entretanto, nem toda falha de conexão leva o sistema a se adaptar. Por exemplo, o contrato dá

prioridade ao uso de internet, em relação às demais opções. Logo, se esta permanecer *online*, mudanças no estado das demais não tem consequências no sistema.

2) Qual é o tempo de resposta médio do SA?

Neste contexto, tempo de resposta é o intervalo entre a atualização no contexto de execução e o fim do processo de adaptação. Este tempo de resposta é fundamental para qualquer sistema crítico, pois impacta diretamente no atendimento adequado das necessidades de seu usuário. Por exemplo, um paciente não pode ficar sem comunicação com seus médicos quando está em estado de emergência.

Para responder a esta pergunta é necessário calcular o intervalo entre a atualização do contexto de execução e o fim da transição entre configurações arquiteturais.

5.4.1.3 ESTUDO DA EVOLUÇAO DINÂMICA

Nesta subseção são demonstradas alternativas para obter as respostas às perguntas postuladas na subseção anterior, segundo as necessidades descritas.

- **O sistema está sujeito a que frequência de adaptações por falha na conexão?**

Existem três alternativas no CM@RT-Visualizer para responder esta pergunta com base nos registros feitos pelo CM@RT-Repository. A primeira utiliza-se das revisões do contexto de execução. A segunda baseia-se na comparação entre revisões do contexto de execução. Finalmente, a terceira emprega os *issues* detectados pelo sistema. Todas as alternativas são acessíveis por meio do índice de dados.

Nas revisões do contexto de execução está registrada a evolução do estado das conexões dos provedores. Logo, é possível identificar dentre os registros quais apresentam falhas na conexão. A Figura 59 ilustra um contexto de execução onde pode ser identificado que o estado da conexão via Internet é *offline*, o que provocaria adaptação para conexão via Linha fixa. Identificar outras ocorrências de mudança no estado da conexão possibilita estimar a frequência de adaptações necessárias ao sistema. Entretanto, a mudança no estado da conexão não significa que a adaptação necessária tenha ocorrido.

Ao utilizar a ferramenta de comparação entre contextos de execução, a busca por mudanças no estado da conexão fica mais simples. Como pode ser visto na Figura 60, as variáveis de contexto alteradas entre as duas revisões selecionadas são marcadas pela cor azul. Além disso, a ordenação das linhas agrupa modificações semelhantes, favorecendo a identificação. Comparar os demais contextos de execução é possível utilizando-se de marcações no índice de dados. Esta alternativa é mais eficiente em relação a anterior no que

se refere à identificação da variável de contexto que se deseja estudar. Entretanto, mantém a limitação de não revelar se o sistema realizou ou não a adaptação quando necessário.

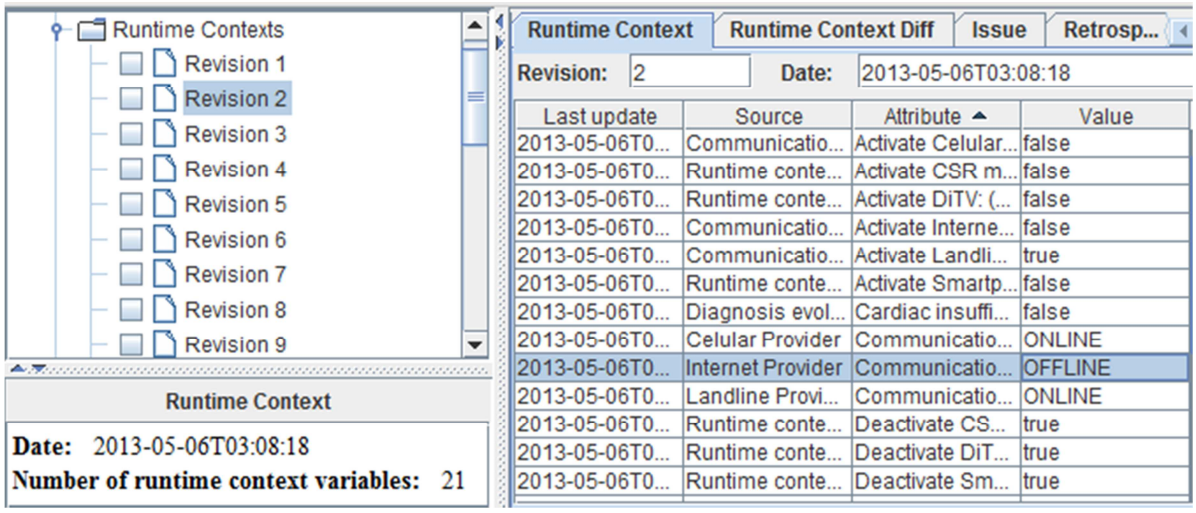


Figura 59 Contexto de execução com estado de conexão alterado

No *issue* deve ser observada sua descrição, que pode ser identificada na Figura 61. Nesta figura está representado o *issue* detectado em função da mudança no contexto de execução relatada no parágrafo anterior. Em função dela, foi solicitada adaptação na configuração arquitetural do sistema. Na descrição do *issue* estão representados os efeitos sobre a configuração arquitetural desta adaptação solicitada. Isto indica que houve a adaptação. Nesta alternativa para obtenção da resposta, a limitação de identificar se houve ou não a adaptação necessária não existe. Porém, apenas para os casos onde o sistema detectou a necessidade de adaptação são gerados *issues*.

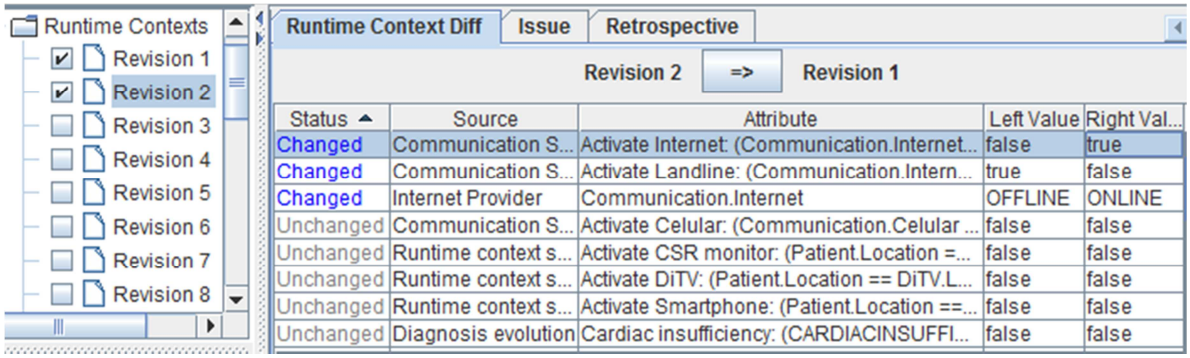


Figura 60 Diff entre revisões de contextos de execução

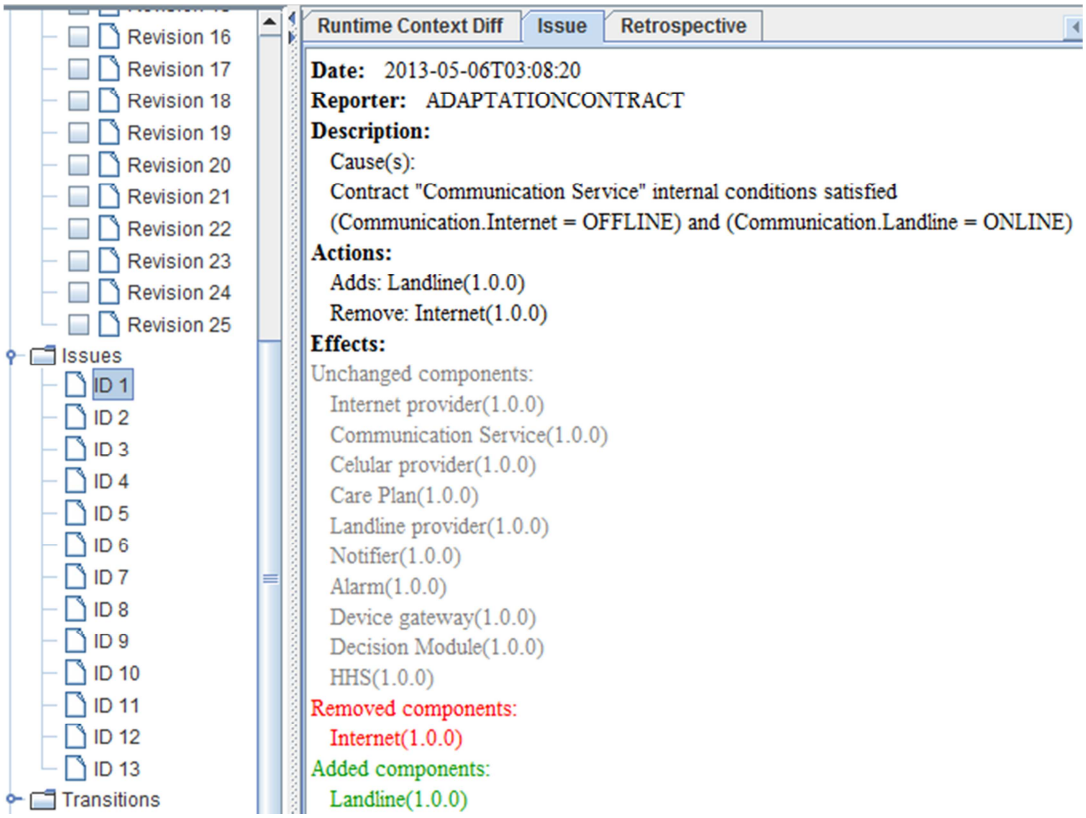


Figura 61 Issue motivado por mudança no estado da conexão

Finalmente, existem duas respostas a esta pergunta. A primeira é a demanda efetiva por adaptação, encontrada por meio do contexto de execução. A segunda é a percepção do SA sobre esta demanda, que está expressa nos *issues* registrados.

- **Qual foi o tempo de resposta médio do SA?**

O CM@RT-Repository registra o instante em que ocorreram a atualização do contexto de execução e da transição entre configurações arquiteturais. Logo, é possível determinar o tempo de resposta do sistema pela diferença entre estes instantes. Consequentemente, o tempo médio do sistema também é determinável.

Em função da importância relatada anteriormente para esta informação, o CM@RT-Visualizer disponibiliza o tempo de resposta por transição e médio do SA em sua interface. Na Figura 62 é possível identificar o tempo de resposta da transição arquitetural selecionada no índice de dados no canto inferior esquerdo da imagem. Na mesma figura é exibido o contexto de execução que motivou a transição, onde é possível identificar o instante em que foi atualizado por meio do campo *Date*. É importante destacar que o tempo de resposta é calculado com maior precisão que a exibida no campo *Date*.

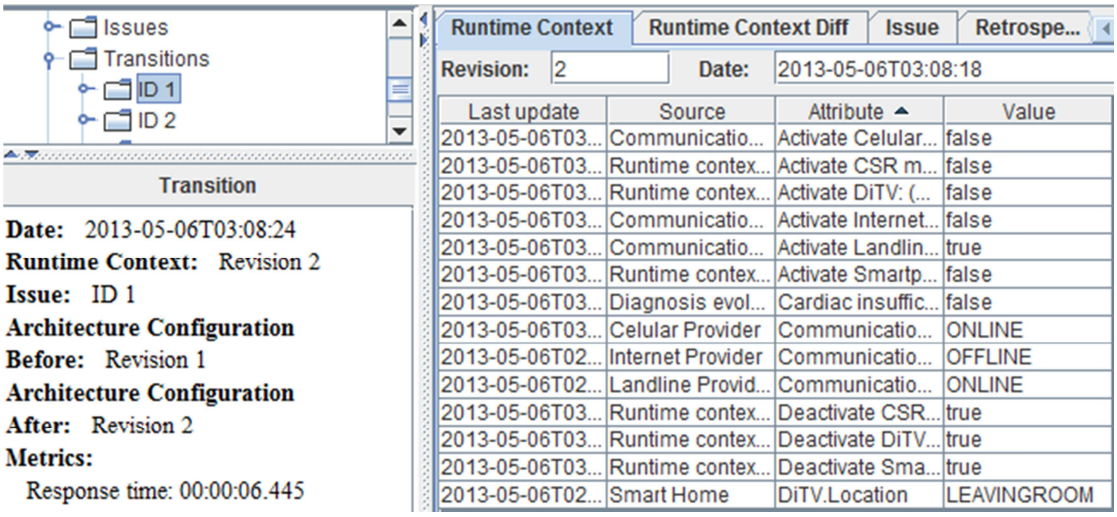


Figura 62 Tempo de resposta de uma transição entre configurações arquiteturais.

O tempo médio de resposta em relação a todas as transições entre configurações arquiteturais registradas pode ser obtido ao selecionar o SA no índice de dados. A Figura 63 ilustra esta situação, onde o tempo de resposta médio é identificado por *Average RT* e exibido com precisão de milésimos de segundo. Este valor é obtido por meio da média aritmética dos tempos de resposta de todas as transições entre configurações arquiteturais registradas. Neste caso, o valor obtido foi de 4.589 segundos.

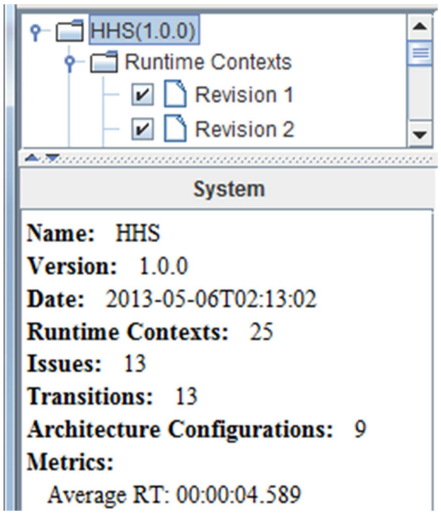


Figura 63 Tempo de resposta médio do SA

5.4.2 CENÁRIO 2: NOTIFICAÇÕES SENSÍVEIS AO CONTEXTO DE EXECUÇÃO

O SCIADS possui a funcionalidade de notificar o paciente utilizando o dispositivo de sua preferência que esteja mais próximo. A Figura 64 ilustra a porção da arquitetura de LPSD que permite identificar três dispositivos capazes de notificar o paciente: o monitor da CSR, a TV digital e o *Smartphone*. É possível ainda identificar na Figura 64 o componente do Plano de Cuidados, responsável por notificar o paciente em momentos oportunos a respeito de

atividades definidas pelo médico. A notificação propriamente dita é solicitada ao componente chamado Notificador, também ilustrado na Figura 64. Ele é responsável por selecionar o dispositivo preferencial do paciente para a notificação.

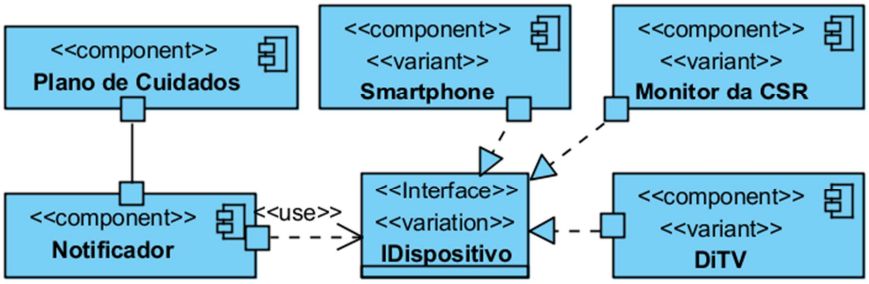


Figura 64 Arquitetura da LPSD para notificações (adaptada de CARVALHO, MURTA e LOQUES (2011))

A seleção do dispositivo é realizada por um contrato específico, gerado a partir das preferências do paciente e dispositivos disponíveis na residência. De forma semelhante ao contrato da seção anterior, é definida uma fila de prioridade entre os dispositivos disponíveis na arquitetura da CSR. Em função da proximidade entre o paciente e os dispositivos disponíveis, o contrato determina o dispositivo que enviará a notificação.

A subseção 5.4.2.1 descreve a configuração arquitetural que implementa a arquitetura definida na Figura 64 e o contrato de adaptação relacionado. A subseção 5.4.2.2 descreve as perguntas de interesse motivadas por este cenário e a subseção 5.4.2.3 descreve como responder estas perguntas com a abordagem.

5.4.2.1 ARQUITETURA E CONTRATO DE ADAPTAÇÃO

A Figura 65 ilustra a configuração arquitetural que concretiza a arquitetura LPSD definida para este cenário. Além dos componentes previstos na arquitetura de LPSD, foi inserido um componente chamado Gerenciador de Dispositivos. Este componente é responsável por intermediar a comunicação entre o componente Notificador e os dispositivos presentes no sistema. Os componentes Plano de Cuidados e Notificador desempenham as funções descritas anteriormente na seção 5.4.2. Os componentes Monitor da CSR, DiTV e *Smartphone* realizam a integração dos respectivos dispositivos ao sistema. Estes componentes tornam disponíveis serviços pelas diferentes interfaces, daí a necessidade do Gerenciador de Dispositivos. Ele concentra o conhecimento destas interfaces e as torna transparentes para o restante do sistema pela interface *IDispositivo*.

Para selecionar o dispositivo preferencial de notificação, o contrato de adaptação descrito na seção 5.4.2 foi implementado da seguinte forma: de acordo com a localização do

paciente na residência, os dispositivos que não estejam no mesmo ambiente da residência são removidos. Caso mais de um dispositivo exista no mesmo ambiente, o Gerenciador de Dispositivos replica a notificação em todos. Se não houver dispositivo disponível, a notificação não é realizada.

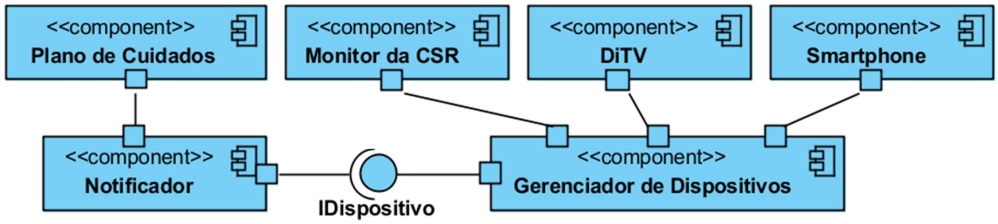


Figura 65 Implementação da arquitetura de LPSD do cenário 2

5.4.2.2 QUESTÕES DE INTERESSE

Em ESFAHANI e MALEK (2012) são discutidas fontes de incertezas enfrentadas por SA, que podem ser internas ou externas. Incertezas internas dizem respeito ao impacto de adaptações sobre a qualidade dos serviços. Incertezas externas dizem respeito ao ambiente de execução e ao domínio da aplicação. A seguir são postuladas duas questões de interesse relacionadas, respectivamente, a incertezas internas e externas.

1) O Notificador possui estabilidade adequada?

O recurso de notificação sensível ao contexto de execução é importante, pois aciona a opção mais adequada à localização do paciente. Este recurso favorece a aderência do paciente a seu tratamento, pois o incentiva a realizar a tarefa notificada. Porém, é possível a ocorrência de falhas nos dispositivos de notificação. Logo, se faz necessária a capacidade de determinar a estabilidade dos dispositivos de notificação. Para isso, é preciso saber se os dispositivos apresentaram falhas no ambiente de execução da residência do paciente.

Para responder a esta pergunta é necessário ter acesso ao comportamento do SA no momento em que foi necessário ativar o dispositivo. Analisando este momento, é possível determinar se ocorre um número significativo de falhas na adição do dispositivo à configuração arquitetural.

2) Qual o comportamento do paciente em sua rotina diária?

Este pode ser considerado um caso de incerteza para os desenvolvedores e arquitetos do SCIADS. Determinar qual a rotina diária do paciente é importante para a definição de seu dispositivo preferencial de notificação. Por exemplo, o dispositivo configurado como preferencial pode nunca ser usado por estar em local não utilizado pelo paciente.

Para responder a esta pergunta é necessário ter acesso ao histórico da movimentação do paciente. A partir dele pode ser analisado se a localização dos dispositivos na residência é

adequada. Entretanto, o dispositivo efetivamente mais utilizado só pode ser determinado em casos onde não exista mais de um dispositivo por cômodo na residência. Caso contrário, não há como determinar qual dispositivo efetivamente está sendo usado pelo paciente para receber as notificações nesta versão do SCIADS.

5.4.2.3 ESTUDO DA EVOLUÇAO DINÂMICA

Nesta subseção são demonstradas alternativas para obter respostas às perguntas postuladas na subseção anterior, segundo as necessidades descritas.

1) O Notificador possui estabilidade adequada?

Para responder a esta pergunta o usuário deve primeiro identificar uma transição entre configurações arquiteturais que seja motivada por mudança no dispositivo de notificação. Navegando pelo índice de dados é possível localizar *issues* com a motivação desejada, o que é ilustrado na Figura 66. O *issue* descreve a adição do dispositivo TV digital, por estar na mesma localização do paciente. Este *issue* foi gerado pelo contrato de adaptação de notificações sensíveis ao contexto de execução. Fato que pode ser identificado pelo atributo *Reporter* ou pelo texto que descreve a motivação do *issue*.

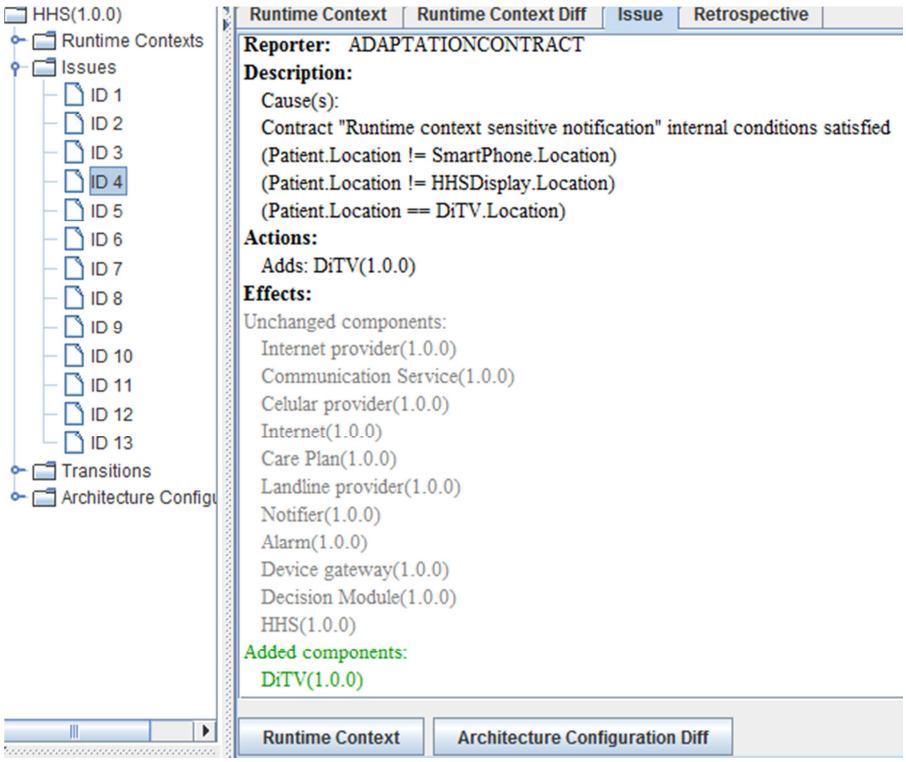


Figura 66 Issue motivado por notificação sensível ao contexto

No entanto, ao analisar um *issue* posterior, ilustrado na Figura 67, é possível perceber que o dispositivo *Smartphone* falhou. A falha ocorreu logo após sua adição, registrada no *issue* precedente [16] (identificado pelo ID 7). O fato foi detectado pelo SAF, que registrou o evento como uma transição arquitetural por falha, ilustrada na Figura 68. A Figura 68 exibe ainda a representação gráfica das modificações realizadas na configuração arquitetural, que estavam descritas textualmente na Figura 67. Por ela é possível identificar que o componente Gerenciador de Dispositivos ficou sem alternativas para realizar notificações solicitadas pelo componente Notificador.

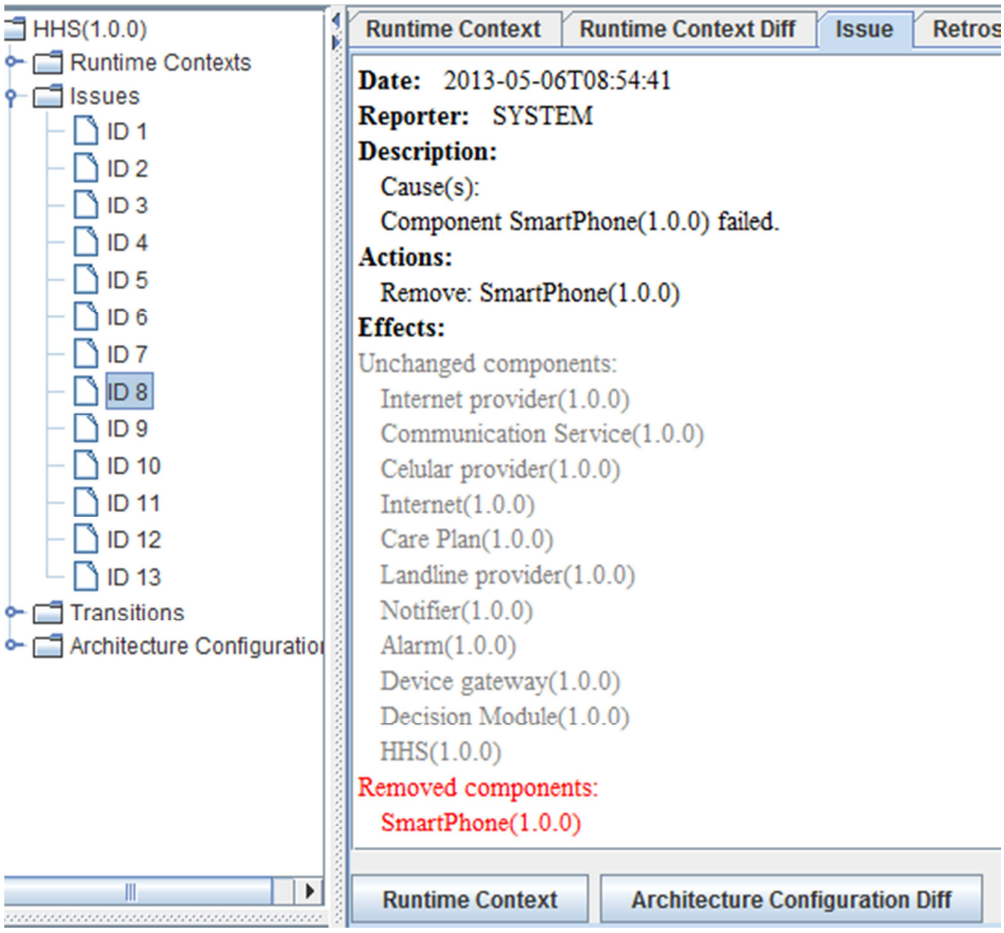


Figura 67 Issue motivado por falha em dispositivo

[16] Omitido por questões de espaço e por não acrescentar informações significativas.

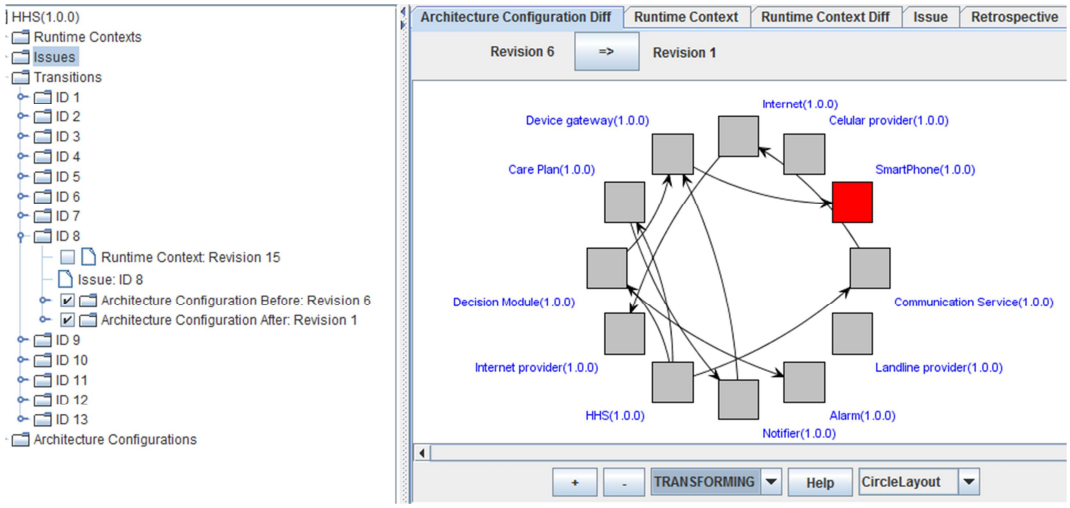


Figura 68 Transição arquitetural por falha em dispositivo

O próximo *issue,* ilustrado na Figura 69, exibe a reação do SAF à falha do dispositivo. Foi gerado um *issue* que solicita nova tentativa de adição do dispositivo. Este mecanismo faz parte da política de tolerância a falhas. Entretanto, em caso de nova falha, não é feita outra tentativa de adição.

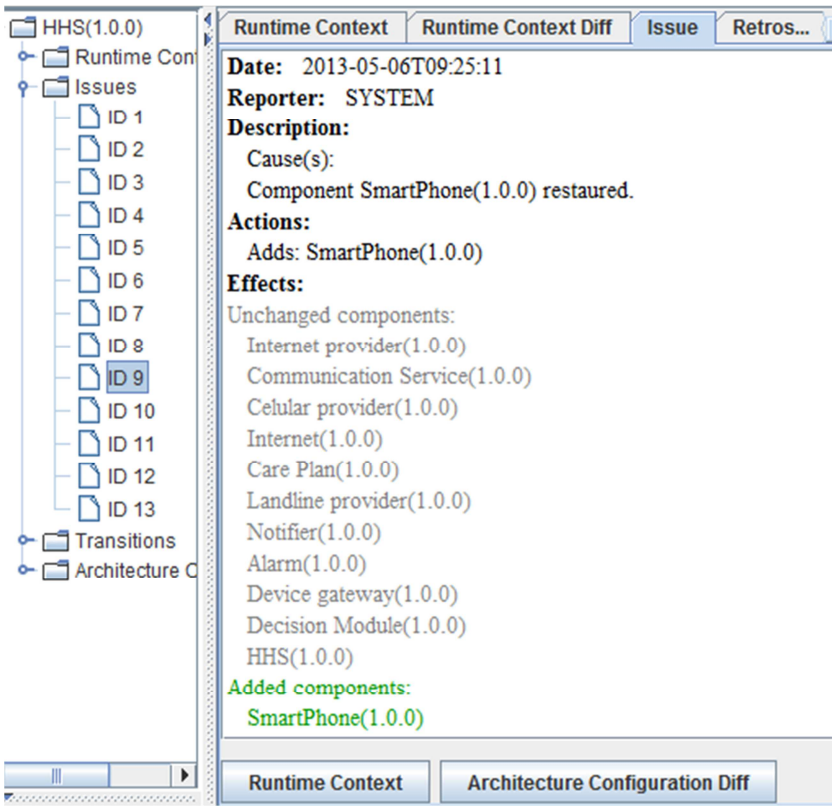


Figura 69 Issue motivado por adição de dispositivo que apresentou falha.

2) Qual o comportamento do paciente em sua rotina diária?

Para acompanhar a evolução da localização do paciente em sua residência, pode ser utilizada a ferramenta de retrospectiva. A Figura 70 ilustra um instante particular da animação

gerada pela ferramenta de retrospectiva. No canto inferior esquerdo desta figura é possível identificar que o paciente se deslocou do banheiro para o quarto, por meio da alteração de valores da variável de contexto *Pacient.Location*. Pela ferramenta de retrospectiva é possível ainda estudar as mudanças ocorridas na configuração arquitetural, que indicam os dispositivos disponíveis para notificação. Por exemplo, neste caso, o dispositivo DiTV foi adicionado à configuração arquitetural do SCIADS, como pode ser visto na representação gráfica das modificações realizadas.

Avançando na animação, é possível fazer outras análises sobre a rotina diária do paciente e o comportamento do SCIADS. Por exemplo, revendo o mapa da residência e a Figura 71, é possível perceber que o paciente se deslocou pela sala, para sair do quarto para o escritório. Entretanto, este fato não foi registrado pelo CM@RT. Em função desta ocorrência, é possível inferir que há problema no sensor de movimento da sala. Além disso, é um caso importante de ser percebido durante eventual auditoria do sistema. Uma situação como esta revela falhas no sistema, que poderiam impactar em seu funcionamento.

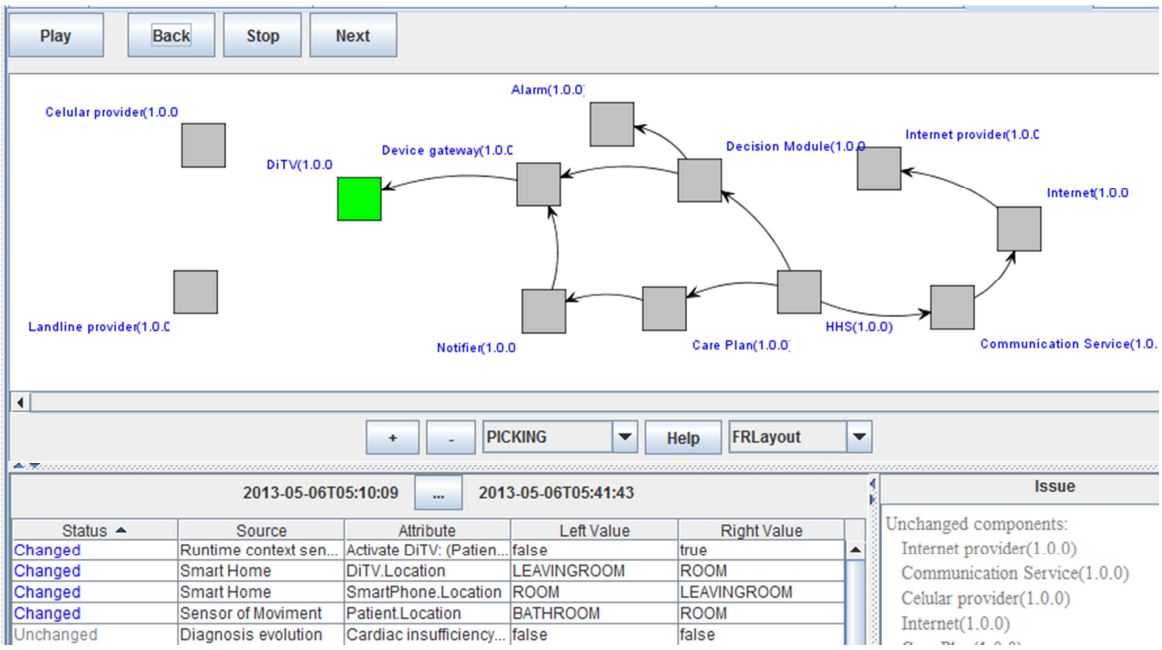


Figura 70 Adição de dispositivo por deslocamento do paciente

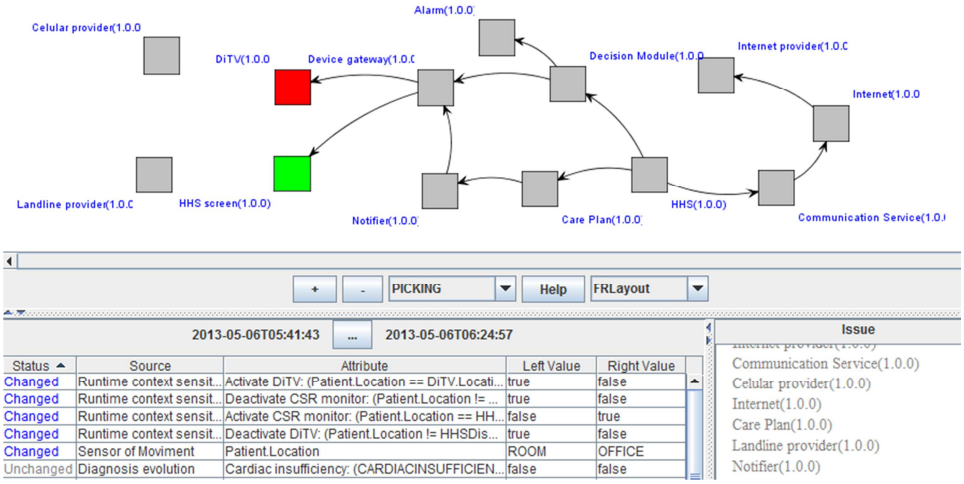


Figura 71 Paciente se desloca pela sala sem disparar detector de movimento

Outra forma de estudar o comportamento do paciente em sua rotina diária é monitorá-lo. O CM@RT-Visualizer é capaz de atualizar suas informações automaticamente. Por exemplo, atualizações no contexto de execução são registradas e disponibilizadas conforme o paciente se desloca pela residência. Para que este recurso esteja habilitado, basta acionar a opção '*Auto update*', exibida na Figura 72.

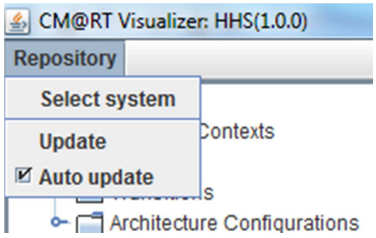


Figura 72 Acionamento da atualização automática do CM@RT-Visualizer

Para comparar os contextos de execução registrados, deve ser usada a ferramenta de comparação exibida na Figura 73, por meio da marcação de duas revisões. A comparação ilustrada mostra duas diferenças entre as quatro possíveis. Foram encontradas variáveis de contexto modificadas e não modificadas. Como dito anteriormente, a ferramenta é capaz de detectar ainda adições e remoções de variáveis de contexto.

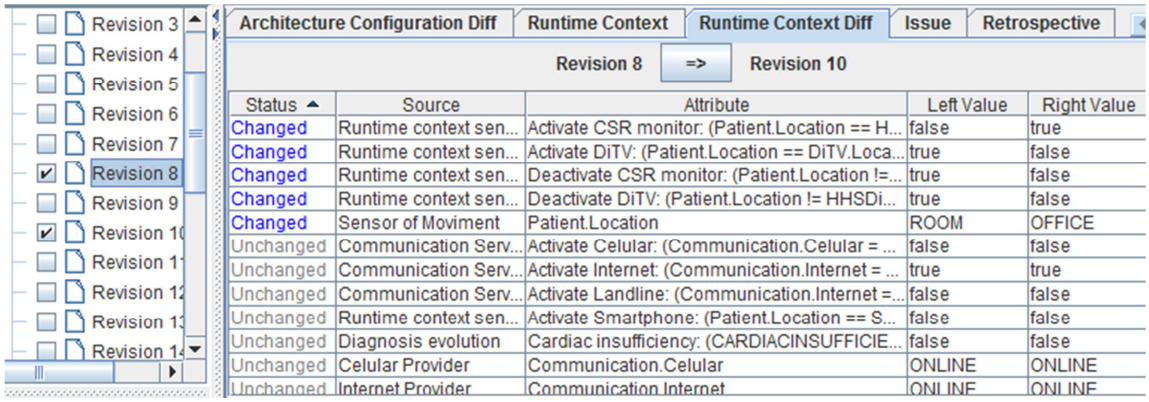


Figura 73 Comparação entre duas revisões do contexto de execução

5.4.3 CENÁRIO 3: EVOLUÇÃO NO QUADRO CLÍNICO DO PACIENTE

Como dito anteriormente, o SCIADS suporta evolução no quadro clínico do paciente, adaptando sua configuração arquitetural em acordo com as demandas. Na Figura 74 está ilustrado um trecho da arquitetura da LPSD, que exemplifica um destes casos. O componente chamado Módulo de Decisão é responsável por analisar os sintomas fisiológicos do paciente, empregando regras específicas por diagnóstico médico. O componente chamado Alarme tem a responsabilidade de enviar alertas a CSM em caso de risco de vida para o paciente. O componente chamado Hipertensão fornece regras para o componente Módulo de Decisão, relativas a pacientes com diagnostico de hipertensão arterial. Além disso, o componente Hipertensão demanda dois sensores, o acelerômetro e o de pressão sanguínea. O componente chamado Insuficiência Cardíaca fornece regras para o componente Módulo de Decisão relativas a pacientes com diagnóstico de insuficiência cardíaca. Além disso, o componente Insuficiência Cardíaca demanda dois sensores, uma balança e medidor de batimento cardíaco.

Neste cenário é importante perceber que um paciente pode ter diferentes diagnósticos ao mesmo tempo. Logo, os contratos que gerem as modificações no sistema devem ser planejados com maior atenção. Em outras palavras, conflitos entre contratos não devem existir, sob hipótese de risco para o paciente. O contrato de adaptação relacionado a este cenário define que para cada diagnóstico médico do paciente, seja adicionado o respectivo componente. Por exemplo, se o paciente apresenta hipertensão, o componente chamado Hipertensão deve ser adicionado à configuração arquitetural.

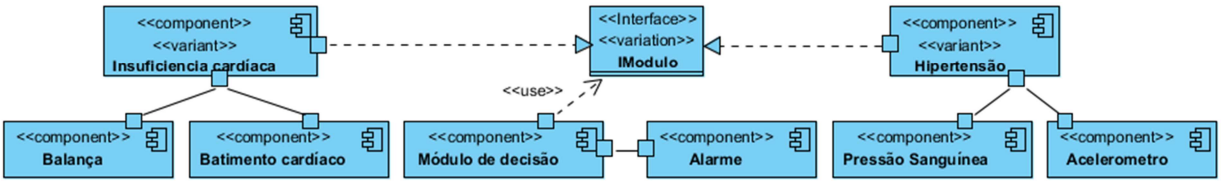


Figura 74 Arquitetura da LPSD para evolução do quadro clínico (adaptada de CARVALHO, MURTA e LOQUES (2012))

A subseção 5.4.3.1 descreve a configuração arquitetural que implementa a arquitetura definida na Figura 74 e o contrato de adaptação relacionado. A subseção 5.4.3.2 descreve as perguntas de interesse motivadas por este cenário e a subseção 5.4.3.3 descreve como responder estas perguntas com a abordagem.

5.4.3.1 ARQUITETURA E CONTRATO DE ADAPTAÇÃO

Diferentemente dos cenários anteriores, não foram necessários novos componentes para concretizar a arquitetura planejada. Consequentemente o diagrama de componentes é igual ao ilustrado na Figura 74, assim como as funções dos componentes.

O contrato de adaptação definido para este cenário foi implementado como descrito na seção 5.4.3. Neste caso também não foram necessárias adaptações em função das limitações do protótipo do SA-SCIADS.

5.4.3.2 QUESTÕES DE INTERESSE

As duas perguntas aqui descritas são motivadas pelos trabalhos CHENG et al. (2009) e VAN DER HOEK, DINCEL e MEDIDOVIC (2003), respectivamente. O trabalho de CHENG et al. (2009) aponta a importância do fornecimento de evidencias sobre adaptações em SA, especialmente para sistemas críticos. Já o trabalho de VAN DER HOEK, DINCEL e MEDIDOVIC (2003) aponta a importância de avaliar a qualidade da configuração arquitetural ao longo da evolução natural de uma LPS.

1) **O que mudou quando o paciente teve evolução no quadro clínico?**

Este cenário apresenta uma característica distinta dos anteriores. Quando um componente fornecedor de regras é adicionado, ele necessita que suas dependências também sejam adicionadas. Nos cenários anteriores, os componentes adicionados não possuíam dependências obrigatórias como neste cenário. Neste caso, sem os sensores presentes o componente não consegue desempenhar suas funções e pode causar risco para o paciente. Logo, é importante verificar se estas dependências são solucionadas adequadamente.

Para responder a esta pergunta é interessante uma ferramenta capaz de representar as diferenças entre as configurações arquiteturais, antes e depois de uma adaptação.

2) Qual a qualidade das configurações arquiteturais do SCIADS?

Com a evolução do quadro clínico, novos módulos são integrados ao sistema, resultando em diferentes produtos da LPSD. Métricas para LPSD permitem avaliar a qualidade das configurações (VAN DER HOEK; DINCEL; MEDVIDOVIC, 2003; NAKAMURA; BASILI, 2005). Além disso, comparar produtos semelhantes pode revelar diferenças significativas na qualidade da configuração.

Para responder a esta pergunta é necessário acesso a métricas de qualidade e a sua evolução ao longo da operação do sistema.

5.4.3.3 ESTUDO DA EVOLUÇAO DINÂMICA

Nesta subseção são demonstradas alternativas para obter respostas às perguntas postuladas na subseção anterior, segundo as necessidades descritas.

1) O que mudou quando o paciente teve evolução no quadro clínico?

O CM@RT-Visualizer disponibiliza uma ferramenta de comparação entre configurações arquiteturais, que foi motivada por demandas como esta. Para analisar o que mudou quando o paciente foi diagnosticado com uma nova doença, primeiro é preciso localizar uma transição entre configurações arquiteturais com esta motivação. Navegando no índice de dados é possível localizar uma transição com estas características de três formas.

A primeira é pelo grupo que indexa os *issues*, analisando suas descrições na visualização dos mesmos. A Figura 75 ilustra a localização de um *issue* por meio do índice de dados. A Figura 75 ainda mostra o botão que ativa a exibição das diferenças entre as configurações arquiteturais. Isto acontece, pois o sistema efetivamente realizou as adaptações solicitadas.

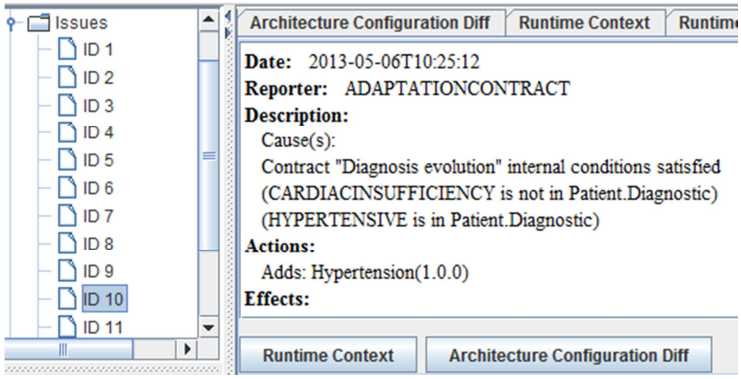


Figura 75 Issue motivado por evolução no quadro clínico

A segunda alternativa é navegar no grupo de revisões da configuração arquitetural. O grupo está ilustrado na Figura 76. Ao selecionar duas configurações arquiteturais, o sistema

atualiza a visualização das diferenças existentes entre as mesmas. Este processo serve tanto para responder a esta pergunta, como a outras semelhantes.

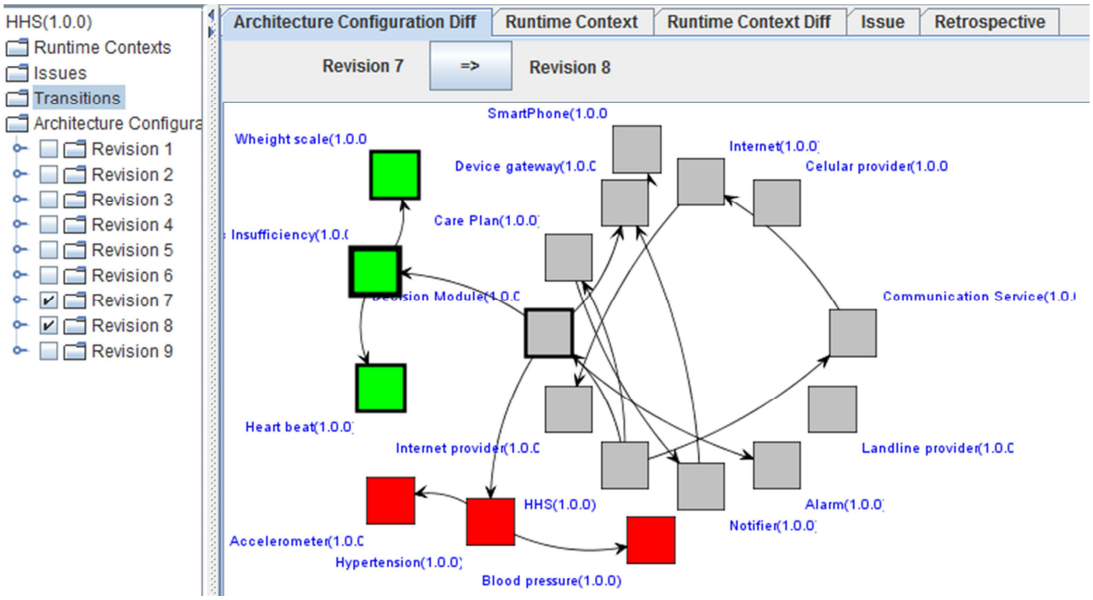


Figura 76 Diferenças entre configurações arquiteturais com alteração no quadro clínico

A terceira alternativa é navegar pelo grupo de transições arquiteturais. A Figura 77 ilustra o grupo em questão. Cada item do grupo aponta o contexto de execução, o *issue* e as configurações arquiteturais antes e após a transição. Logo, permite que a busca seja feita tanto na primeira quanto na segunda alternativa.

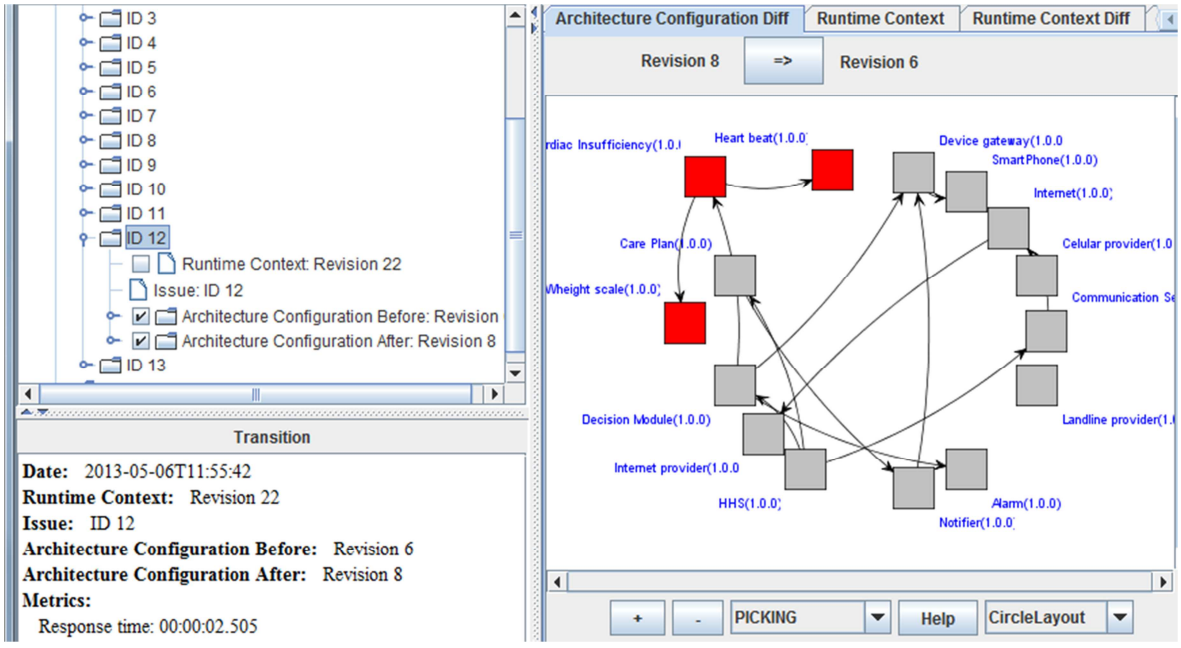


Figura 77 Navegação no grupo de transições arquiteturais

2) Qual a qualidade das configurações arquiteturais do SCIADS?

Como descrito anteriormente, o CM@RT-Visualizer disponibiliza dois grupos de métricas de qualidade para configurações arquiteturais inseridas em uma LPSD. O primeiro é relativo a configuração arquitetural como um todo, e o segundo é calculado por componente. Para identificar os valores das métricas relativas a configuração arquitetural, basta selecionar uma no índice de dados. A Figura 78 ilustra este caso. A Figura 79 ilustra o caso onde o usuário selecionou um componente desta mesma revisão. É possível identificar em ambas as figuras, a exibição das métricas na parte inferior da imagem. Estas métricas foram explicadas no Capítulo 2, seção 2.6.1.

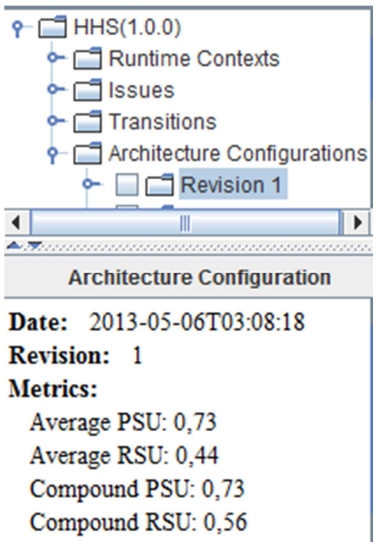


Figura 78 Métricas para uma configuração arquitetural selecionada

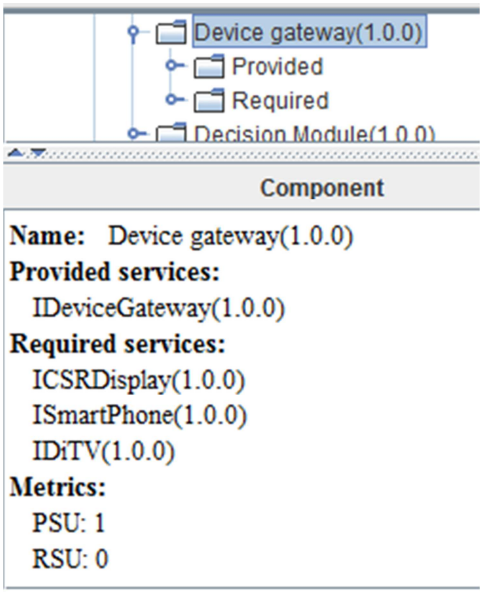


Figura 79 Métricas de um componente selecionado

5.5 AMEAÇAS AO ESTUDO

A avaliação realizada permite identificar indícios dos benefícios gerados pela abordagem CM@RT na realização de análises sobre o comportamento de SA. Os dados registrados pelo CM@RT-Repository e representados no CM@RT-Visualizer fornecem informações suficientes à obtenção de respostas a perguntas motivadas pelas 5W+1H. Entretanto, existem ameaças ao estudo que serão descritas a seguir.

Amostragem – apenas um SA foi utilizado na avaliação. É preciso realizar a avaliação em uma amostragem superior para obter indícios mais fortes dos benefícios esperados. Entretanto, o SA alvo é um sistema crítico, alvo de diversas pesquisas (CARVALHO; COPETTI; LOQUES, 2010, 2011; CARVALHO; MURTA; LOQUES, 2011, 2012) e sob evolução constante. Logo, suas demandas fornecem indícios sobre os benefícios da abordagem.

Avaliação – foi realizada uma demonstração do uso da abordagem. Demonstrações não fornecem meios para comprovar os benefícios pretendidos. São necessários experimentos adicionais para comprovação dos benefícios da abordagem. No entanto, trata-se de uma abordagem voltada à realização de análises humanas do comportamento de SA. O fornecimento de ferramentas de visualização sobre dados históricos deste comportamento favorecem sua compreensão (GEORGAS; VAN DER HOEK; TAYLOR, 2009).

Cenários – a avaliação foi realizada em um número reduzido de cenários de transição entre configurações arquiteturais. Aumentar a quantidade de cenários é necessário para obter diversidade nas situações demonstradas. Entretanto, cada um dos cenários envolveu situações diversificadas de utilização da abordagem. Tanto situações triviais quanto não triviais foram tocadas. Além disso, estão presentes no ambiente de execução da avaliação componentes de quatro sistemas distintos, a saber, o SA-SCIADS, o SAF, o CM@RT e os componentes da plataforma OSGI.

5.6 CONSIDERAÇÕES FINAIS

A avaliação realizada permite identificar indícios dos benefícios gerados pela abordagem CM@RT na realização de estudos sobre a evolução dinâmica de SA. Os dados registrados pelo CM@RT-Repository e representados no CM@RT-Visualizer fornecem informações para a obtenção de respostas a perguntas de interesse da área de SA. Além disso, a abordagem fornece métricas de qualidade sobre o SA. Estas métricas permitem avaliar as modificações independente da experiência do usuário da ferramenta.

No entanto, a obtenção das repostas desejadas nem sempre é feita de forma direta. Este fato se deve a generalidade dos recursos de análise disponíveis. Acredita-se que com a evolução da pesquisa serão criadas novas ferramentas que tornem mais simples a obtenção de respostas a perguntas que atualmente são complexas.

De forma intencional, a abordagem não suporta o registro de aspectos internos do SA, o que implicaria em elevado grau de acoplamento. Esta limitação pode impactar na análise do comportamento do SA por imprecisão na descrição dos *issues*. No entanto, esta limitação não impediu a realização de perguntas específicas ao domínio do SA utilizado na demonstração. Além disso, o CM@RT possibilita responder perguntas não relacionadas a GC. Por exemplo, é possível avaliar qual a movimentação do paciente ao longo de sua rotina diária, por meio do registro de sua movimentação no contexto de execução.

O próximo capítulo apresenta as conclusões e trabalhos futuros, possibilitados pela abordagem desenvolvida. Estes trabalhos são apenas alguns exemplos selecionados entre as ideias que surgiram durante a realização da pesquisa.

CAPÍTULO 6 – CONCLUSÃO

6.1 EPÍLOGO

Enfrentar desafios relacionados a SA implica em estudar sua evolução dinâmica, durante e após sua execução. Isto demanda informações capazes de revelar questões como: o que mudou, por que mudou, quando mudou, onde mudou, quem solicitou a mudança e como esta foi realizada. Estas questões são pertinentes tanto durante, como após a execução do SA.

O trabalho realizado propôs uma abordagem motivada pelos seguintes objetivos: (1) permitir o estudo da evolução dinâmica de SA, (2) registrar a evolução dinâmica de SA e (3) permitir o monitoramento e auditoria da evolução de SA. A abordagem proposta foi implementada em dois módulos, bem como outros para sua avaliação. Por meio deles foi realizada a avaliação dos benefícios da abordagem mediante sua aplicação em cenários pertinentes a um SA crítico.

O capítulo corrente apresenta as contribuições, limitações dos módulos construídos e oportunidades de trabalhos futuros. A seção 6.2 descreve as contribuições do trabalho. A seção 6.3 apresenta as limitações dos módulos construídos. Na seção 6.4 são relatadas oportunidades de trabalhos futuros.

6.2 CONTRIBUIÇÕES

As principais contribuições deste trabalho são:

- Uma alternativa ao uso de mecanismos de log para registro da evolução de SA em tempo de execução
- Evolução dinâmica do SA registrada por meio de modelagem independente de abordagem do mesmo.
- Modelagem capaz de responder questões baseadas em questões de interesse dede 5W+1H
- Modelagem permite a extração de métricas sobre a evolução dinâmica de SA e sua configuração arquitetural
- Solução voltada tanto ao ambiente de desenvolvimento quanto ao de operação
- Algoritmo para determinação de diferenças entre configurações arquiteturais
- Algoritmo para determinação de diferenças entre contextos de execução

Além das contribuições citadas anteriormente, pode ser listado o seguinte legado:

- Módulo para registro da evolução dinâmica de SA

- Módulo para visualização dos dados de evolução dinâmica registrados
- Protótipos para integração da abordagem a plataforma OSGi

6.3 LIMITAÇÕES

Apesar da abordagem CM@RT atingir os objetivos estabelecidos em sua concepção, existem limitações no trabalho. A principal limitação refere-se ao repositório de dados do CM@RT-Repository. Durante a implementação do protótipo de integração do CM@RT-Repository com a plataforma OSGi, a utilização de um banco de dados como repositório não obteve sucesso. Foram buscadas alternativas em diversas ocasiões, entretanto os resultados encontrados não produziram soluções estáveis. Entre as razões para esta dificuldade está a natureza dinâmica do OSGi e arquitetura do JPA, que não considera este cenário. Este problema está entre as motivações para a *Apache Software Foundation* criar o projeto *Apache Aries*[17]. O projeto *Aries* fornece um conjunto de *bundles* específicos para o suporte a JPA. Entretanto, esta alternativa foi vislumbrada recentemente e sua complexidade impediu a sua utilização. Ademais, o projeto *Apache Aries* ainda se encontra em estado de incubação. Este fato implica em dificuldades adicionais, como erros não conhecidos e ainda não solucionados[18].

O CM@RT-Visualizer apresenta um número significativo de recursos para realização de monitoramento e auditoria. Entretanto, recursos de usabilidade adicionais são necessários a fim de tornar a interação do usuário mais simples e eficiente. Por exemplo, a existência de um mecanismo de busca no índice de dados simplificaria a identificação de elementos desejados. Outra limitação importante refere-se à visualização do comparativo entre configurações arquiteturais. A visualização não evidencia as modificações nas conexões entre componentes. Isto ocorre devido à versão atual do algoritmo de comparação entre configurações arquiteturais não considerar esta informação.

6.4 TRABALHOS FUTUROS

Nesta seção são relatadas oportunidades para realização de trabalhos futuros a partir dos resultados desta pesquisa. Acreditamos que estes trabalhos são apenas uma pequena parte das possibilidades criadas por esta pesquisa. A seguir são descritos os trabalhos em questão.

[17] Site: http://aries.apache.org/
[18] https://issues.apache.org/jira/browse/ARIES-978

6.4.1 ANÁLISE AUTOMÁTICA DO COMPORTAMENTO DE SISTEMAS AUTOADAPTÁVEIS

Atualmente, a abordagem CM@RT requer a participação de um usuário humano para a realização de estudos sobre a evolução dinâmica apresentada por um SA ao longo de seus períodos de execução. Entretanto, existe o interesse em construir mecanismos para a realização de análises automatizadas sobre a evolução de um SA. Entre as alternativas vislumbradas está a aplicação de métricas de qualidades para obtenção de indicadores de desvio no comportamento apresentado.

VILLEGAS et al. (2011) propõem um framework para avaliação de propriedades de adaptação de SA. A abordagem define uma série de propriedades que servem de base para o cálculo de um conjunto de métricas de qualidade. Entre os desafios enfrentados pela proposta está a imaturidade das métricas e a necessidade de mecanismos capazes de capturar as propriedades.

Estes desafios podem ser solucionados a partir da abordagem CM@RT. A abordagem pode ser modificada para registrar a evolução das propriedades necessárias. A partir delas podem ser calculadas as métricas propostas para diferentes SA. Em relação à imaturidade das métricas, existe o caso da métrica tempo de resposta. VILLEGAS et al. (2011) cita duas definições para a métrica, o que fornece indícios de sua imaturidade.

6.4.2 ALTERNATIVAS PARA MECANISMO DE DECISÃO

Mecanismos de decisão para determinação das adaptações usualmente são baseados nas expectativas de funcionamento do SA. Entretanto, tendo um conjunto suficientemente grande de registros da evolução de um determinado SA, outras técnicas de mecanismos de decisão se tornam disponíveis. Por exemplo, envolvendo redes neurais e mineração de dados.

Existe uma relação direta entre o contexto de execução e adaptações realizadas pelo sistema. Logo, aplicar redes neurais para responder quais são as adaptações necessárias possivelmente seria mais eficiente que realizar o processamento de centenas de Contratos de Adaptação Arquitetural.

O uso de mineração de dados sobre a mesma base de dados também seria capaz de determinar as alterações arquiteturais necessárias em face de um determinado contexto de execução. Além disso, a mineração de dados revelaria elementos do contexto de execução cujos valores tivessem baixa influência sobre as adaptações. Isso permitiria a redução do custo de monitoramento do contexto de execução e ganho de eficiência do sistema.

6.4.3 INTEGRAÇÃO A OUTROS AMBIENTES OSGI

Confirmando as tendências percebidas no início de nossa pesquisa, dois trabalhos envolvendo a construção de sistemas autoadaptáveis, baseados no ciclo MAPE-K, realizando modificações na configuração arquitetural e concretizados por meio da plataforma OSGi chegaram ao nosso conhecimento. FONSECA *et al* (2012) descrevem a construção de um *framework* para execução de planos de reconfiguração arquitetural, baseados na plataforma OSGi e no modelo de componentes iPOJO. Nossa proposta foi avaliada utilizando-se protótipos que utilizam o modelo de componentes baseados em Serviços Declarativos. CAVALCANTI; SOUZA e ROSA (2013) também desenvolveram uma infraestrutura baseada em OSGi e iPOJO, voltada à construção de SA com adaptações sensíveis a QoS. A integração do CM@RT a estas infraestruturas representa uma oportunidade para demonstrarmos a flexibilidade da proposta e exploração de novos cenários de aplicação. Além disso, abrem caminho para realização de avaliações experimentais em sistemas complexos. Os autores destes trabalhos foram contatados e a integração da abordagem CM@RT despertou o interesse de ambos.

6.4.4 OTIMIZAÇÃO DO MONITORAMENTO

Idealmente um SA monitora exclusivamente variáveis de ambiente relevantes para a determinação de adaptações necessárias. Entretanto, a determinação deste conjunto ideal não é um processo trivial. Além disso, a evolução do SA pode afetar negativamente o conjunto de variáveis monitoradas. Por exemplo, Sistemas de Ultralarga Escala podem possuir contextos de execução igualmente extensos, logo, adicionar variáveis redundantes ou removê-las incorretamente é uma possibilidade. Existe também o fator incerteza relativo ao contexto de execução em diferentes instâncias do SA, sujeitas a ambientes heterogêneos. Uma mesma variável de ambiente pode apresentar comportamento significativamente divergente em ambientes distintos. Por exemplo, sendo estável em um caso e muito volátil em outro. Como consequência, o primeiro SA não teria seu comportamento afetado pela variável em questão, enquanto que o segundo realizaria adaptações constantemente. Isto permite ao primeiro SA deixar de monitorar a variável estável, enquanto que o segundo deve ter seu comportamento ajustado.

Aplicar técnicas estatísticas ou de mineração de dados sobre o histórico do contexto de execução pode revelar variáveis de ambiente com comportamento divergente das estimativas. Com esta informação, o SA pode ser configurado para melhor se ajustar as suas condições de operação. Entretanto, este ajuste pode ser afetado pelo ambiente de execução de determinadas

instâncias do SA, o que motiva a realização de pesquisas mais aprofundadas, provavelmente envolvendo cruzamento de resultados entre instâncias do SA.

6.4.5 SISTEMA DE CONTROLE DE VERSÃO

O CM@RT-Repository realiza o registro da evolução da configuração arquitetural. Está entre os objetivos futuros o desenvolvimento de um módulo para controlar a evolução de configurações arquiteturais. Este módulo funcionaria de forma análoga a SCV e seria chamado de CM@RT-VCS. O CM@RT-VCS forneceria operações como restauração e rotulação. Além disso, permitiria a realização atividades de verificação sobre as modificações ocorridas na configuração arquitetural.

A integração do CM@RT-VCS a um SA permitiria acesso a importantes funcionalidades. Por exemplo, em caso de instabilidade no SA, seria possível restaurar a configuração arquitetural rotulada como de maior estabilidade. Esta operação poderia ser realizada por de intervenção humana ou de forma automatizada. No caso de intervenções automatizadas, as configurações arquiteturais sugeridas podem ser obtidas com base no trabalho futuro descrito na seção 6.4.2.

6.4.6 SISTEMAS DE ULTRALARGA ESCALA

Este trabalho está limitado a SA cujas adaptações ocorrem no escopo do ambiente de execução local. Entretanto, é possível que adaptações sejam necessárias em ambiente de execução distribuídos, como, por exemplo, no caso de Sistemas de Ultralarga Escala (NORTHROP *et al.*, 2006). Este cenário impõe desafios técnicos e de pesquisa, tanto para o registro do comportamento quanto para sua compreensão. Entre os desafios técnicos está a captura e consolidação de informações de fontes distribuídas e seu registro no repositório, também distribuído. Em relação a compreensão do comportamento existe um grande desafio referente a escala das informações a serem analisadas. Por exemplo, Sistemas de Ultralarga Escala militares tem amplitude global e novos sistemas integrados constantemente (NORTHROP *et al.*, 2006).

Acreditamos que analisar o comportamento de Sistemas de Ultralarga Escala militares durante ou após cenários de combate, seja uma tarefa complexa e de importância significativa. Além disso, pensamos que a abordagem CM@RT pode ser uma alternativa importante para lidar com estas demandas. No entanto, são necessárias pesquisas sobre técnicas e novas abordagens para lidar com uma escala tão elevada de informações. Por exemplo, desenvolver

mecanismos flexíveis para automação das análises, uma vez que serão inúmeros sistemas integrados a serem avaliados.

6.4.7 SISTEMAS MULTIAGENTE

Assim como SA, Sistemas Multiagente (SMA) possuem comportamento significativamente complexo e mutável em função do contexto de execução (JENNINGS, 2000). Porém, ao contrário de SA, a decisão sobre as modificações necessárias nem sempre ocorre de forma centralizada. Além disso, cada agente busca seus próprios objetivos, seja em benefício próprio ou do grupo. Logo, existem dois níveis de comportamento, o do agente e do SMA como um todo.

Para estudar a evolução do SMA é preciso também estudar a evolução dos agentes, uma vez que o primeiro é resultado do último. Atualmente, estudar a evolução de SMA implica na utilização de log ou ferramentas proprietárias. Como dito anteriormente, o uso de log implica na avaliação de grandes volumes de informações, favorecendo a ocorrência de erros. Ferramentas proprietárias são capazes de fornecer visualizações para uso durante o desenvolvimento, como, por exemplo, a oferecida pelo *framework* Jadex [19]. Porém, esta última alternativa ignora estudos de longo prazo, uma vez que as informações representadas são voláteis.

Acreditamos que é possível aplicar a abordagem CM@RT para registro e estudo da evolução de agentes de SMA. Porém, seria necessária a realização de ajustes na modelagem, para comportar demandas de SMA. A abordagem CM@RT seria de grande valia para desenvolvedores, arquitetos e auditores ao disponibilizar dados históricos sobre a evolução dos agentes. Por exemplo, os dados históricos permitem a aplicação de técnicas de mineração de dados sobre a evolução de um SMA e mesmo de um agente específico. A mineração poderia revelar padrões comportamentais não perceptíveis de outra maneira, como, por exemplo, influência do comportamento de um indivíduo sobre o grupo.

6.4.8 SISTEMAS UBÍQUOS

Uma importante aplicação possível para a abordagem CM@RT está relacionada a sistemas ubíquos (SU). Assim como SA, SU lidam com constantes modificações em sua composição arquitetural. Além disso, impõem um desafio significativo: a existência de requisitos de interoperabilidade que caracteriza SU. Por exemplo, um SU para auxílio à saúde

[19] http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Features

pode possuir em sua composição, dispositivos de fabricantes diferentes e que interagem entre si. Outro desafio importante é a limitação de recursos inerente a SU, que impõe restrições à utilização de ferramentas adicionais.

Analisar o comportamento resultante da interação entre componentes de um SU é uma tarefa de complexidade significativa, especialmente se não houver um controle central. A utilização da abordagem CM@RT poderia permitir o registro do comportamento de um SU a partir de um dispositivo próprio de monitoramento. Entretanto, é preciso pesquisar uma alternativa de monitoramento capaz de permitir o monitoramento das decisões pontuais dos componentes do SU, sem ignorar a questão da interoperabilidade.

6.4.9 VERIFICAÇÃO E VALIDAÇÃO

Em TAMURA *et al.* (2013) são investigadas as implicações e desafios de se realizar Verificações e Validações (V&V) em SA. Realizar V&V é uma atividade fundamental para a obtenção de sistemas confiáveis, e que ainda não possuem mecanismos estabelecidos para SA. A proposta da pesquisa é integrar V&V a diferentes fases do ciclo de adaptação, realizando diferentes atividades de V&V ao longo do mesmo. Entretanto, os autores reconhecem os riscos para o desempenho e outras propriedades de um SA ao realizar estas atividades durante a operação do sistema. Eles sugerem que formas inovadoras de permitir a realização de V&V devem ser buscadas, como meios para realizar verificações de composição da arquitetura ou do modelo que a representa.

Os dados obtidos pelo CM@RT durante o rastreamento das adaptações fornecem uma alternativa para a realização de V&V, sem impactar nas adaptações realizadas. V&V pode ser realizada posteriormente às adaptações sobre dados históricos, atestando a qualidade das adaptações durante longos períodos de execução do sistema. Possibilitaria, por exemplo, a aplicação de V&V em sistemas críticos submetidos a condições reais de operação.

REFERÊNCIAS

ALLIANCE, O. S. G. Osgi service platform release 4 version 4.2 compendium specification. [S.l.]: August, 2009.

APPLEBY, K.; FAKHOURI, S.; FONG, L.; GOLDSZMIDT, G.; KALANTAR, M.; KRISHNAKUMAR, S.; PAZEL, D. P.; PERSHING, J.; ROCHWERGER, B. Oceano-SLA based management of a computing utility. In: INTEGRATED NETWORK MANAGEMENT PROCEEDINGS, 2001 IEEE/IFIP INTERNATIONAL SYMPOSIUM ON; 2001, Seattle, WA. **Proceedings**... Seattle, WA: Conference Publications, 2001. p. 855–868.

BECK, K. Embracing change with extreme programming. IEEE COMPUTER, [S.l.], 1999. v. 32, p. 70–77.

BENCOMO, N.; BLAIR, G. Using architecture models to support the generation and operation of component-based adaptive systems. SOFTWARE ENGINEERING FOR SELF-ADAPTIVE SYSTEMS, [S.l.], 2009. p. 183–200.

BENCOMO, N.; SAWYER, P.; BLAIR, G.; GRACE, P. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In: 12TH INTERNATIONAL CONFERENCE OF SOFTWARE PRODUCT LINES; 2008, Limerick, Ireland. Limerick, Ireland: [s.n.], 2008. p. 23–32.

BENCOMO, NELLY. On the use of software models during software execution. In: MODELING IN SOFTWARE ENGINEERING, 2009. ICSE WORKSHOP ON; 2009, Vancouver, BC. **Proceedings**... Vancouver, BC: IEEE, 2009. p. 62–67.

BLAIR, G.; BENCOMO, N.; FRANCE, R. B. Models@run.time. IEEE COMPUTER, Los Alamitos, CA, 2009. v. 42, p. 22–27.

BOEHM, B. W. A spiral model of software development and enhancement. COMPUTER, Los Alamitos, CA, 1988. v. 21, p. 61–72.

BRAGA, C.; CHALUB, F.; SZTAJNBERG, A. A Formal Semantics for a Quality of Service Contract Language. ELECTRONIC NOTES IN THEORETICAL COMPUTER SCIENCE, Amsterdam, The Netherlands, The Netherlands, abr. 2009. v. 203, p. 103–120.

BRUN, Y.; DI MARZO SERUGENDO, G.; GACEK, C.; GIESE, H.; KIENLE, H.; LITOIU, M.; MÜLLER, H.; PEZZÈ, M.; SHAW, M. Engineering self-adaptive systems through feedback loops. SOFTWARE ENGINEERING FOR SELF-ADAPTIVE SYSTEMS, Berlin, Heidelberg, 2009. p. 48–70.

CANDEA, G.; KICIMAN, E.; KAWAMOTO, S.; FOX, A. Autonomous recovery in componentized internet applications. CLUSTER COMPUTING, Hingham, MA, USA, 2006. v. 9, p. 175–190.

CARVALHO, S. T.; COPETTI, A.; LOQUES, O. Sistema de computação ubíqua na assistência domiciliar à saúde. JOURNAL OF HEALTH INFORMATICS, [S.l.], 2011. v. 3, p. 51–57.

CARVALHO, S. T.; COPETTI, A.; LOQUES, O. Um Sistema Computacional Inteligente de Assistência Domiciliar à Saúde. In: XII CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE; 2010, Porto de Galinhas, Brasil. Porto de Galinhas, Brasil: [s.n.], 2010. p. 1–6.

CARVALHO, S. T.; LOQUES, O.; MURTA, L. Dynamic Variability Management in Product Lines: An Approach Based on Architectural Contracts. In: IV BRAZILIAN SYMPOSIUM ON SOFTWARE COMPONENTS, ARCHITECTURES AND REUSE; 2010, Bahia, Brazil. Bahia, Brazil: IEEE, 2010. p. 61–69.

CARVALHO, S. T.; MURTA, L.; LOQUES, O. A Contract-based Approach for Managing Dynamic Variability in Software Product Line Architectures. **Technical Report**. Niterói, Brazil: Tempo Laboratory, Real-Time and Embedded Systems, Fluminense Federal University. 2011.

CARVALHO, S. T.; MURTA, L.; LOQUES, O. Variabilities as First-Class Elements in Product Line Architectures of Homecare Systems. In: 4TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING; 2012, Zurich, Switzerland. Zurich, Switzerland: IEEE, 2012. p. 33–39.

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; 2013, Belfast, United Kingdom. **Proceedings**... Belfast, United Kingdom: IEEE, 2013. p. 323–327.

CHENG, B. *et al.* Software engineering for self-adaptive systems: A research roadmap. SOFTWARE ENGINEERING FOR SELF-ADAPTIVE SYSTEMS, Heidelberg, Berlim, 2009. p. 1–26.

COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.; PILATO, C. M. Version Control with Subversion. 2. ed. Sebastpol, CA, USA: O'Reilly Media, 2008.

CONRADI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. ACM COMPUTING SURVEYS, New York, NY, USA, jun. 1998. v. 30, p. 232–282.

DASHOFY, E.; VAN DER HOEK, A.; TAYLOR, R. N. An Infrastructure for the Rapid Development of XML-Based Architecture Description Languages. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE); 2002, Orlando, FL, USA. **Proceedings**... Orlando, FL, USA: [s.n.], 2002. p. 266–276.

DOBSON, S. *et al.* A survey of autonomic communications. ACM TRANSACTIONS ON AUTONOMOUS AND ADAPTIVE SYSTEMS (TAAS), New York, NY, USA, dez. 2006. v. 1, p. 223–259.

DOWLING, J.; CAHILL, V. Self-managed decentralised systems using K-components and collaborative reinforcement learning. In: PROCEEDINGS OF THE 1ST ACM SIGSOFT WORKSHOP ON SELF-MANAGED SYSTEMS; 2004, New York, NY, USA. **Proceedings**... New York, NY, USA: ACM, 2004. p. 39–43.

ESFAHANI, N.; MALEK, S. Uncertainty in Self-Adaptive Software Systems. SOFTWARE ENGINEERING FOR SELF-ADAPTIVE SYSTEMS 2. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012. v. 7475. , p. 214–238.

FERREIRA, J.; LEITÃO, J.; RODRIGUES, L. A-OSGi: A Framework to Support the Construction of Autonomic OSGi-Based Applications. Em: VASILAKOS, A.; BERALDI, R.; FRIEDMAN, R.; MAMEI, M. (Org.). AUTONOMIC COMPUTING AND COMMUNICATIONS SYSTEMS. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Berlin, Heidelberg: Springer-Verlag, 2010. v. 23. , p. 1–16. Disponível em: <http://dx.doi.org/10.1007/978-3-642-11482-3_1>.

FONSECA, F. L.; BENEDITTO, M. E. M. D.; WERNER, C. M. L. Um mecanismo extensível para a execução de um plano de reconfiguração arquitetural sob o framework OSGi+iPOJO. In: BRAZILIAN CONFERENCE ON SOFTWARE: THEORY AND PRACTICE; 2012, Natal, Rio Grande do Norte, Brazil. Natal, Rio Grande do Norte, Brazil: [s.n.], 2012.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. M. Design Patterns: Elements of Reusable Object-Oriented Software. 1. ed. USA: Addison-Wesley Publishing Co., 1994.

GARG, A.; CRITCHLOW, M.; CHEN, P.; VAN DER WESTHUIZEN, C.; VAN DER HOEK, A. An Environment for Managing Evolving Product Line Architectures. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE (ICSM); 2003, Amsterdam, Netherlands. **Proceedings**... Amsterdam, Netherlands: [s.n.], 2003. p. 358–367.

GARLAN, D.; CHENG, S. W.; HUANG, A. C.; SCHMERL, B.; STEENKISTE, P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. IEEE COMPUTER, Los Alamitos, CA, USA, 2004. v. 37, p. 46–54.

GARLAN, D.; SCHMERL, B.; CHENG, S. W. Software architecture-based self-adaptation. IN AUTONOMIC COMPUTING AND NETWORKING, [S.l.], 2009. p. 31–55.

GEORGAS, J. C.; TAYLOR, R. N. Towards a knowledge-based approach to architectural adaptation management. In: PROCEEDINGS OF THE 1ST ACM SIGSOFT WORKSHOP ON SELF-MANAGED SYSTEMS; 2004, New York, NY, USA. **Proceedings**... New York, NY, USA: ACM, 2004. p. 59–63.

GEORGAS, J. C.; VAN DER HOEK, A.; TAYLOR, R. N. Architectural runtime configuration management in support of dependable self-adaptive software. SIGSOFT SOFTW. ENG. NOTES, [S.l.], Maio 2005. v. 30, p. 1–6.

GEORGAS, J. C.; VAN DER HOEK, A.; TAYLOR, R. N. Using Architectural Models to Manage and Visualize Runtime Adaptation. IEEE COMPUTER SOCIETY PRESS, Los Alamitos, CA, USA, 2009. v. 42, p. 52–60.

HORN, P. Autonomic Computing: IBM's Perspective on the State of Information Technology. **Research report**. USA: International Business Machines Corporation. 2001. Disponível em: <http://researchweb.watson.ibm.com/autonomic/>.

IEEE. Std 1028 - IEEE recommended practice for software requirements specifications. **Technical Report**. New York, NY, USA: Institute of Electrical and Electronics Engineers. 1998. p. 1–47.

INSTITUTE FOR SOFTWARE RESEARCH. ArchStudio - Software and Systems Architecture Development Environment. Disponível em: <http://www.isr.uci.edu/projects/archstudio/>. Acesso em 30 nov. 2012.

JENNINGS, N. R. On agent-based software engineering. ARTIFICIAL INTELLIGENCE, [S.l.], mar. 2000. v. 117, p. 277–296.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. IEEE COMPUTER SOCIETY PRESS, Los Alamitos, CA, USA, jan. 2003. v. 36, p. 41–50.

KIPLING, R. Just So Stories. Londres, UK: Penguin Books, 1902.

LADDAGA, R. Active software. SELF-ADAPTIVE SOFTWARE. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001. v. 1936. , p. 11–26.

LÜER, C.; VAN DER HOEK, A. JPloy: User-Centric Deployment Support in a Component Platform. In: IFIP/ACM WORKING CONFERENCE ON COMPONENT DEPLOYMENT (CD); 2004, Edinburgh, UK. **Proceedings**... Edinburgh, UK: [s.n.], 2004. p. 190–204.

MARELI, D. F. M. **Reconhecimento de Atividades em um Sistema Computacional Pervasivo de Assistência Domiciliar à Saúde**. Niterói, RJ - Brasil: Ciência da Computação, IC-UFF, 2011.

MCCAFFERY, F.; CASEY, V.; MCHUGH, M. How can software SMEs become medical device software SMEs. SYSTEMS, SOFTWARE AND SERVICE PROCESS IMPROVEMENT. [S.l.]: Springer, 2011. p. 247–258. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-642-22206-1_22>. Acesso em 3 abr. 2013.

MORIN, B.; BARAIS, O.; JÉZÉQUEL, J.-M.; FLEUREY, F.; SOLBERG, A. Models@ Run.time to Support Dynamic Adaptation. IEEE COMPUTER, v. 42, n. 10, p. 44–51, out. 2009.

MUKHIJA, A.; GLINZ, M. Runtime adaptation of applications through dynamic recomposition of components. SYSTEMS ASPECTS IN ORGANIC AND PERVASIVE COMPUTING-ARCS 2005. Lecture Notes in Computer Science. [S.l.]: Springer Berlin Heidelberg, 2005. p. 124–138.

NAKAMURA, T.; BASILI, V. Metrics of Software Architecture Changes Based on Structural Distance. In: PROCEEDINGS OF THE 11TH IEEE INTERNATIONAL SOFTWARE METRICS SYMPOSIUM; 2005, [S.l.]. **Proceedings**... [S.l.]: IEEE, 2005. p. 8–1.

NORTHROP, L.; FEILER, P. H.; POLLAK, B.; PIPITONE, D. Ultra-large-scale systems: the software challenge of the future. Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2006.

O'BRIEN, T.; CASEY, J.; FOX, B.; SNYDER, B.; ZYL, J. V.; REDMOND, E. Maven: The Definitive Guide. 1st. ed. Sebastopol, CA, USA: O'Reilly, 2008.

O'MADADHAIN, Joshua; FISHER, Danyel; WHITE, Scott; SMYTH, Padhraic; BOEY, Yan-biao. Analysis and Visualization of Network Data using JUNG. JOURNAL OF STATISTICAL SOFTWARE, [S.l.], 2005. 10, p. 1–25.

PARASHAR, M.; HARIRI, S. Autonomic computing: An overview. UNCONVENTIONAL PROGRAMMING PARADIGMS, [S.l.], 2005. p. 97–97.

PARNAS, D. L. Inspection of safety-critical software using program-function tables. In: IFIP CONGRESS; 1994, [S.l.]. **Proceedings**... 1994. p. 270–277.

ROSENMÜLLER, M.; SIEGMUND, N.; PUKALL, M.; APEL, S. Tailoring dynamic software product lines. In: PROCEEDINGS OF THE 10TH ACM INTERNATIONAL CONFERENCE ON GENERATIVE PROGRAMMING AND COMPONENT ENGINEERING; 2011, [S.l.]. **Proceedings**... 2011. p. 3–12.

ROSHANDEL, R.; VAN DER HOEK, A.; MIKIC-RAKIC, M.; MEDVIDOVIC, N. Mae-a system model and environment for managing architectural evolution. ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY (TOSEM), [S.l.], 2004. v. 13, p. 240–276.

ROUILLARD, J. P. Real-time log file analysis using the simple event correlator (SEC). PROCEEDINGS OF LISA XVIII, [S.l.], 2004. p. 133–49.

ROYCE, W. W. Managing the development of large software systems. In: PROCEEDINGS OF IEEE WESCON; 1970, Los Angeles, US. **Proceedings**... Los Angeles, US: TRW, 1970. p. 1–11.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. ACM TRANS. AUTON. ADAPT. SYST., New York, NY, USA, maio 2009. v. 4, p. 14:1–14:42.

SMITH, R.; KOREL, B. Slicing event traces of large software systems. ARXIV PREPRINT CS/0101005, [S.l.], 2001.

STANDARD, S. H. Federal Information Processing Standard Publication# 180. US DEPARTMENT OF COMMERCE, NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, SHA-1, 1993. v. 56, p. 57–71.

TAMURA, G. *et al.* Towards practical runtime verification and validation of self-adaptive software systems. SOFTWARE ENGINEERING FOR SELF-ADAPTIVE SYSTEMS 2. LNCS. [S.l.]: Springer, 2013. v. 7475. , p. 108–132.

TAVARES, A. L. C.; VALENTE, M. T. A gentle introduction to OSGi. ACM SIGSOFT SOFTWARE ENGINEERING NOTES, [S.l.], 2008. v. 33, p. 8.

TUCKER, C.; SHUFFELTON, D.; JHALA, R.; LERNER, S. OPIUM: Optimal package install/uninstall manager. In: SOFTWARE ENGINEERING, 2007. ICSE 2007. 29TH INTERNATIONAL CONFERENCE ON; 2007, [S.l.]. **Proceedings**... 2007. p. 178–188.

VALDMAN, J. Log file analysis. **DCSE**. Pilsen: Department of Computer Science and Engineering (FAV UWB). 2001. Disponível em: <https://www.kiv.zcu.cz/site/documents/verejne/vyzkum/publikace/technicke-zpravy/2001/tr-2001-04.pdf>. Acesso em 24 nov. 2012.

VAN DER HOEK, A. Design-Time Product Line Architectures for Any-Time Variability. SCIENCE OF COMPUTER PROGRAMMING, [S.l.], 2004. v. 53, p. 285–304.

VAN DER HOEK, A.; DINCEL, E.; MEDVIDOVIC, N. Using service utilization metrics to assess the structure of product line architectures. In: SOFTWARE METRICS SYMPOSIUM, 2003. PROCEEDINGS. NINTH INTERNATIONAL; 2003, [S.l.]. **Proceedings**... 2003. p. 298–308.

VAN DER HOEK, A.; MIKIC-RAKIC, M.; ROSHANDEL, R.; MEDVIDOVIC, N. Taming Architectural Evolution. In: ACM SIGSOFT SOFTWARE ENGINEERING NOTES; 2001, [S.l.]. **Proceedings**... 2001. p. 1–10.

VILLEGAS, N. M.; MÜLLER, H. A.; TAMURA, G.; DUCHIEN, L.; CASALLAS, R. A framework for evaluating quality-driven self-adaptive software systems. In: SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS; 2011, Waikiki, Honolulu, HI, USA. Waikiki, Honolulu, HI, USA: ACM, 2011. p. 80–89.

WHITE, J.; SCHMIDT, D. C.; GOKHALE, A. Simplifying autonomic enterprise java bean applications via model-driven development: A case study. MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS. [S.l.]: Springer Berlin Heidelberg, 2005. p. 601–615.

ZAIDMAN, A.; DEMEYER, S. Managing trace data volume through a heuristical clustering process based on event execution frequency. In: SOFTWARE MAINTENANCE AND REENGINEERING, 2004. CSMR 2004. PROCEEDINGS. EIGHTH EUROPEAN CONFERENCE ON; 2004, [S.l.]. **Proceedings**... 2004. p. 329–338.