MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2018/04.03.19.17-TDI

UM FRAMEWORK PARA ADAPTAÇÃO DINÂMICA DE SOFTWARE APLICADO A SISTEMAS DE SEGMENTO SOLO

Moacyr Gonçalves Cereja Júnior

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Nilson Sant'Anna, aprovada em 04 de maio de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3QRDJ45>

INPE
São José dos Campos
2018

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2018/04.03.19.17-TDI

UM FRAMEWORK PARA ADAPTAÇÃO DINÂMICA DE SOFTWARE APLICADO A SISTEMAS DE SEGMENTO SOLO

Moacyr Gonçalves Cereja Júnior

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Nilson Sant'Anna, aprovada em 04 de maio de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3QRDJ45>

INPE
São José dos Campos
2018

Dados Internacionais de Catalogação na Publicação (CIP)

Cereja Júnior, Moacyr Gonçalves.

C334f Um framework para adaptação dinâmica de software aplicado a sistemas de segmento solo / Moacyr Gonçalves Cereja Júnior. – São José dos Campos : INPE, 2018.

xxiv + 168 p. ; (sid.inpe.br/mtc-m21c/2018/04.03.19.17-TDI)

Tese (Doutorado em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2018.

Orientador : Dr. Nilson Sant'Anna.

1. Sistemas auto adaptáveis. 2. Sistemas solo. 3. Computação autônoma. 4. Arquiteturas de software. 5. Qualidade de serviço. I.Título.

CDU 629.7.08:004.5

Aluno (a): ***Moacyr Gonçalves Cereja Júnior***

Título: "UM FRAMEWORK PARA ADAPTAÇÃO DINÂMICA DE SOFTWARE APLICADO A SISTEMAS DE SEGMENTO SOLO".

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de ***Doutor(a)*** em

Engenharia e Tecnologia Espaciais/Eng. Gerenc. de Sistemas Espaciais

Dr. Maurício Gonçalves Vieira Ferreira

Presidente / INPE / SJCampos - SP

( ) Participação por Video - Conferência

Dr. Nilson Sant'Anna

Orientador(a) / INPE / SJCampos - SP

( ) Participação por Video - Conferência

Dra. Maria de Fátima Mattiello-Francisco

Membro da Banca / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Walter Abrahão dos Santos

Membro da Banca / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Fábio Fagundes Silveira

Convidado(a) / UNIFESP / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Reinaldo Gen Ichiro Arakaki

Convidado(a) / FATEC / SJCampos - SP

( ) Participação por Video - Conferência

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 04 de maio de 2018

"Se, a princípio, a ideia não é absurda, então não há esperança para ela.".

Albert Einstein

A minha esposa ***Camila Kozlowski Della Côrte*** *e a meus filhos* ***Ana Clara*** *e* ***Artur****. A minha mãe* ***Deocília*** *e a memória de meu pai* ***Moacyr****.*

AGRADECIMENTOS

Agradeço a Deus em primeiro lugar, que determina o tempo certo para cada coisa e que tudo provê.

Agradeço à minha esposa Camila Kozlowski Della Côrte pelo companheirismo, atenção, dedicação e finalmente a parceria sem a qual não seria possível a conclusão deste trabalho.

Agradeço à minha mãe Deocília pelo apoio nesta empreitada e aos meus irmãos Isabel Cecília, Eduardo, Rafael e Renata.

Agradeço ao meu orientador Nilson Sant'Anna pela confiança e por ter me dado a oportunidade de dar prosseguimento a este trabalho no qual teve seu início no seu projeto de pós-doutorado na Politecnico di Milano, Itália.

Um agradecimento especial ao Sam Guinea e Luciano Baresi da Politecnico di Milano, Itália, referência na pesquisa de sistemas auto-adaptáveis, pelas contribuições e colaboração com o Dr. Nilson Sant´Anna. O trabalho que ambos realizaram até então forneceram os fundamentos deste trabalho de pesquisa.

Agradecimento especial ao João Paulo Firmeza da Open Labs S.A. pela compreensão e apoio neste trabalho desde o início.

Agradecimento especial também ao João Ricardo Silva da Open Labs S.A. pelo apoio na realização deste trabalho.

Agradeço ao Marcos Paulo Salgueiro Moraes pela parceria neste trabalho, no qual trouxe contribuições relevantes para os aspectos relacionados à qualidade de serviço.

Agradeço ainda aos meus sogros Marcantônio Dela Côrte e Mathilde Helena Kozlowski por olharem os netos e também por me apoiarem neste trabalho.

Agradeço também aos membros da banca examinadora Dra. Fátima Matiello, Dr. Maurício, Dr. Walter, Dr. Reinaldo e Dr. Fábio pelas valiosas contribuições.

Por último um agradecimento aos amigos Anderson Borges, Anderson Pontes, Márcio Ito, Thiago Pascon, Cibele Niero, Rafael Lago, João Castilho e Renata Quirino pelo apoio.

RESUMO

Arquiteturas de sistemas espaciais são compostas de três segmentos: espacial, lançador e solo. Nos últimos anos, com referência ao segmento espacial, inovações tecnológicas, como cubesats, têm produzido impactos como a diminuição do tamanho dos satélites e aumento do volume de objetos no espaço. Desta forma, ao invés de adotar grandes satélites nas missões espaciais, são utilizados constelações e mega constelações. Esta alteração arquitetural produz um número grande de satélites que devem ser controlados de forma simultânea e o aumento da volumetria de dados que devem ser processados pelos sistemas de segmento solo.

Em virtude deste contexto, foi realizada neste trabalho uma vasta pesquisa, no campo dos sistemas de software auto adaptáveis, apresentados conceitos e um framework denominado Arctic Fox, que permite um sistema de Segmento de Solo alterar suas condições estruturais e toda a sua dinâmica. Esta adaptação é realizada pela implementação de duas classes de conhecimento fundamentais o Conhecimento Contextual (Estrutura do Sistema) e o Conhecimento Situacional (Dinâmica e Comportamento do Sistema), elementos estes responsáveis pela tomada de decisão de adaptação. Vários elementos compõem este conhecimento como os Componentes de Sistemas existentes (Serviços), elementos de hardware e principalmente as métricas de Qualidade de serviço.

A utilização deste conhecimento é realizada através do Modelo MAPE-K, que proporciona o monitoramento, a análise, o planejamento e a execução das alterações estruturais e comportamentais do sistema. A base de conhecimento é atualizada por elementos responsáveis pela aquisição do conhecimento, que reflete as inclusões e as alterações dos estados dos elementos que compõem a arquitetura do Sistema Solo. De maneira a avaliar o framework proposto, foi implementado um protótipo, e um estudo de caso para o EMBRACE. (programa de Estudo e Monitoramento BRAsileiro de Clima Espacial) do INPE. Os sistemas de solo do Clima Espacial são responsáveis pelo armazenamento, processamento e disseminação de dados de diversos tipos de instrumentos. Desta forma, situações como a inoperância de um instrumento, ou falha em algum servidor, produzem impacto na operação regular do Sistema Solo, com degradação da qualidade do serviço prestado para a missão.

No estudo de caso em questão, a utilização do framework é exercitada e através de um experimento controlado foram analisados os impactos da auto-adaptação. Utilizando solicitações de uso de um serviço típico do EMBRACE, o Gtex, verificou-se ganhos para a “disponibilidade do sistema”, via a observação da Taxa de Sucesso da chamada ao Serviço. Desta forma, conclui-se neste experimento que o uso do framework contribui de forma significativa para a Qualidade do Sistema.

Palavras-chave: Sistemas auto adaptáveis. Sistemas Solo. Computação Autônoma. Arquiteturas de Software. Qualidade de Serviço.

A FRAMEWORK TO PROVIDE SELF-ADAPTIVE SUPPORT TO SPACE GROUND SYSTEMS

ABSTRACT

Space systems architectures are composed of three segments: space segment, ground systems and launchers. The space segment, in recent years has driven lots of technological innovations, such as “cubesats”. These new kinds of technologies produce impacts such as decreasing of the size of satellites and increasing the amount of objects in space. Thus, instead of adopting large satellites in space missions, constellations and mega constellations of very small satellites are used. These architectural changes produce a large number of satellites that must be controlled simultaneously and have their large amount of data processed, most of the time, in real time by the ground systems.

In this work, a vast research of self-adaptive software systems was carried out, concepts and a framework called Arctic Fox are presented. Using the “framework” the ground systems can have their structural elements and behavior conditions changed. This adaptation is accomplished by the implementation of two fundamental knowledge classes that are used for decision making: Context Awareness (Structural) and Situational Awareness (Behavior). This knowledge is composed by several elements such as the existing system components (services), hardware elements and especially the quality of service metrics.

The use of this knowledge is performed through the MAPE-K model, which provides the monitoring, analysis, planning and execution of adaptations in the ground systems. The knowledge base is updated by elements responsible for the knowledge acquisition, which reflects the inclusions and the changes of the elements states that compose the ground System architecture.

In order to evaluate the proposed “framework”, a prototype was implemented, and a case study for the Brazilian Space Weather Program EMBRACE of INPE is conducted. The "Space Weather Systems" are responsible for the storage, processing and data dissemination of different types of instruments. In this way, situations such as some instrument data provider or server computers failure, impact the regular ground systems operation, with degradation of the service quality provided by the mission.

In the stated case study, the use of the “framework” is exercised and through a controlled experiment the impacts of self-adaptation were analyzed. Using services calls and derived "success rate" metrics to a typical EMBRACE service, “Gtex Service”, availability gains was observed. The experiment demonstrates that the use of the framework contributes significantly to improve the ground systems segment quality.

Keywords: Self-Adaptive Systems. Space Ground Systems. Autonomic Computing. Software Architecture. Quality of Service.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

SLA	–	Service Level Agreement (Acordo de Nível de Serviço)
MAPE	–	Monitoramento, Análise, Planejamento e Execução
INPE	–	Instituto Nacional de Pesquisas Espaciais
ARCTICFOX	–	Framework para adaptação dinâmica de sistemas
AOP	–	Aspect Oriented Programing
IDE	–	Integrated Development Environment
SAS	–	Self-Adaptive Software
FESAS	–	Framework for Engineering Self-Adaptive Systems
FUSION	–	FeatUre-oriented Self-adaptatION
OOMW	–	Object Oriented Middleware
FLAGS	–	Fuzzy Live Adaptive Goals for Self-adaptive systems
RESIST	–	RESISTing Reliability Degradation through Proactive Reconfiguration
ADC	–	Anticipatory Dynamic Configuration
EMBRACE	–	Estudo e Monitoramento BRAsileiro de Clima Espacial
GNSS	–	Global Navigation Satellite System
GPS	–	Global Positioning System
GTex	–	GNSS-TEC Exchange

SUMÁRIO

Pág.

1 INTRODUÇÃO

Arquiteturas de sistemas espaciais (LARSON; WERTZ, 1999) (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2008) são compostas de três segmentos: espacial, lançador e solo. Nos últimos anos, com referência ao segmento espacial, inovações tecnológicas, como cubesats, têm produzido impacto no sentido de miniaturizar os satélites. Esta realidade aumentou a complexidade das arquiteturas de sistemas espaciais nos aspectos custo e operacional.

Com referência à operação, as novas arquiteturas promovem cada vez mais a utilização de constelações e mega-constelações de satélites. Esta alteração arquitetural produz uma volumetria maior de dados a serem processados e de satélites que devem ser controlados pelos sistemas de segmento solo. De outro lado, estas inovações também prometem reduzir os custos das missões, dadas as padronizações das especificações e homologação de componentes para construção de nano-satélites, pico-satélites e mini-satélites.

Cita-se como exemplo destas arquiteturas de sistemas espaciais a promoção de Internet banda larga para as regiões mais longínquas do globo terrestre (AIRBUS SPACE & DEFENSE, 2016) e (VIRGILIA et al., 2016). Para tal feito, propõe-se a injeção em órbita baixa terrestre de uma mega-constelação (KITAJIMA et al., 2016) de mini-satélites com instrumentos para prover sinal de Internet para todo o território terrestre. Outros exemplos são a construção autônoma de telescópios no espaço (SAUNDERS et al., 2016), a exploração do cosmo em busca de exoplanetas (TESSENYI et al., 2016), bem como missões com a utilização de enxames de pico-satélites para a exploração de asteroides (HINCHEY et al., 2005).

Ainda com referência ao segmento espacial, na realidade do INPE há o projeto CONASAT (CARVALHO et al., 2013), que prevê a substituição dos micro satélites de coleta de dados (SCD) com 120 kg de massa desenvolvidos no âmbito da primeira Missão Espacial Completa Brasileira (MECB) para coleta de dados (ORLANDO; KUGA, 2007) por uma constelação de 6 cubesats que desempenham as mesmas funções de missão dos satélites SCD.

Com referência ao segmento solo, estas arquiteturas de sistemas espaciais geram um grande volume de dados para processamento pelos centros de disseminação de dados científicos e também o aumento dos satélites que devem ser controlados pelos centros de rastreamento e controle de satélites.

No contexto de sistemas de segmento solo para disseminação de dados científicos, há o programa de Estudo e Monitoramento BRAsileiro de Clima Espacial - EMBRACE (TAKAHASHI et al., 2012), cujo objetivo principal é realizar o monitoramento do espaço Sol-Terra, magnetosfera e ionosfera-atmosfera por satélites e observações terrestres. Em paralelo, viabiliza o fornecimento de informações úteis às comunidades científicas, áreas industriais e tecnológicas (TAKAHASHI et al., 2008) (SANTANNA et al., 2014).

1.1 Definição do Problema

O EMBRACE implementou uma arquitetura responsável pelo armazenamento, processamento e disseminação de dados de diversos tipos de instrumentos geograficamente distribuídos, tais como: magnetômetros, ionossondas, rede de GPS, dados de rádio e raios X de explosões solares, entre outros. Esta arquitetura, denominada PIPELINE, recupera, transmite e carrega os dados dos instrumentos diretamente para os servidores EMBRACE no INPE para processamento pelas aplicações de Clima Espacial.

No contexto do EMBRACE, situações como a inoperância de um instrumento, falha em algum servidor, baixa qualidade de serviço prestada por determinada carga útil, inclusão de novos instrumentos,e aumento de demanda de utilização pela comunidade científica, geram a necessidade de incluir nos projetos de desenvolvimento de sistemas, requisitos relacionados à operação autônoma.

1.2 Motivação

Arquiteturas de software restringem a forma como os sistemas são alterados e são modeladas por meio de componentes e conectores (GARLAN et al., 2000). Os arranjos de componentes e conectores são denominados configurações ou topologia (TAYLOR et al., 2009) (BASS et al., 2003). Eventos como falhas de sistema ou aumento de carga são consequência de erros ou inconsistências na arquitetura do software e requerem alteração destes elementos arquitetônicos, e o padrão arquitetural de software em si contribui para facilitar a alteração (adaptação) do sistema e a melhoria da manutenção do sistema como um todo.

Para que os sistemas de software continuem a prestar o serviço para o qual foram projetados, são necessárias mudanças para: correção de defeitos (bugfix), atender a requisitos de novas necessidades e ajustes no ambiente no qual se insere o sistema de software.

A motivação deste trabalho de pesquisa está em propor uma solução para a questão da operação autônoma dos sistemas de software de segmento solo por meio da adaptação dinâmica de software, isto é, auto-adaptação de software.

1.3 Solução Proposta

Sistemas auto-adaptáveis se apresentam como uma solução para atender aos requisitos relacionados à operação autônoma dos sistemas de software. Segundo Oreizy et al. (1999), um sistema auto-adaptável pode mudar o seu comportamento em resposta às alterações ou às condições adversas em seu ambiente operacional.

Sistemas de software auto-adaptáveis possuem propriedades para atender a seus respectivos objetivos de adaptação. Estes objetivos podem estar relacionadas à capacidade de configuração automática e dinâmica em resposta às mudanças no ambiente (autoconfiguração), diagnóstico das condições do sistema e reparo automático (auto-reparo), gerenciamento do desempenho e alocação automática de recursos (auto-otimização) e defender o sistema contra ataques maliciosos, bem como antecipar problemas e tomar medidas para evitá-los ou para mitigar seus efeitos (auto-proteção) (SALEHIE; TAHVILDARI, 2009) (KEPHART; CHESS, 2003) (BERNS; GHOSH, 2009). Metodologias propostas para adaptação envolvem algumas características comuns baseadas em monitoramento do sistema, análise, planejamento e implantação da adaptação (KEPHART; CHESS, 2003) (KRUPITZER et al., 2016). As condições necessárias à adaptação de software também são descritas.

Neste trabalho de pesquisa, é apresentado o *framework* denominado Arctic Fox para adaptação dinâmica de sistemas, o qual provê suporte autoadaptativo a sistemas de segmento solo. É apresentada a arquitetura do framework, baseada no modelo MAPE-K (KEPHART; CHESS, 2003), que possui um mecanismo de gerenciamento do ciclo de vida de adaptação responsável pelo monitoramento do ambiente no qual se insere o sistema, análise de dados observáveis, planejamento e execução de alterações em sistemas de software de forma autônoma. O framework permite o desenvolvimento de heurísticas especializadas para certas propriedades de adaptação, bem como de recursos para a implementação de lógica de adaptação.

De maneira a validar a arquitetura proposta, foi implementado um protótipo do framework e um estudo de caso para o serviço GTex do EMBRACE, utilizado para produzir mapas de cintilação da ionosfera e, de maneira a avaliar o protótipo implementado, são comparados no estudo de caso, em um cenário de alta carga, a disponibilidade do serviço Gtex, com e sem o uso do framework.

1.4 Organização do Trabalho

O trabalho está organizado em oito capítulos e um apêndice, da seguinte maneira:

Definições e conceitos relacionados aos sistemas espaciais com ênfase para o segmento solo são apresentados no capítulo 2, onde também é apresentado o Programa BRasileiro de Clima Espacial, que é explorado no estudo de caso, bem como um panorama sobre diversas missões espaciais autônomas, como a exploração do espaço profundo, oferta de Internet banda larga para todo o globo terrestre e montagem autônoma de instrumentos no espaço, que influenciaram na motivação deste trabalho. No final do capítulo, é apresentada uma seção que resume os principais aspectos dos sistemas espaciais e direciona a discussão para o próximo capítulo.

No capítulo 3, são apresentadas as definições e os conceitos relacionados a arquiteturas de software e sistemas auto-adaptáveis. Este capítulo aborda os principais aspectos de um sistema auto-adaptável, propriedades, trabalhos correlatos e lacunas observadas. O *framework* é comparado com os trabalhos correlatos, onde são descritos prós e contras do *framework* Arctic Fox em relação às demais abordagens.

No capítulo 4, são exploradas as tecnologias para implementação de sistemas auto-adaptáveis que influenciaram na arquitetura do framework e na implementação do protótipo.

No capítulo 5, é apresentada uma visão geral do *framework* objeto deste trabalho, isto é, as principais premissas que norteiam o trabalho, os requisitos para adaptação de sistemas, conhecimento do contexto e consciência situacional, o modelo de adaptação adotado e os mecanismos de extensão. Também é apresentada uma visão de arquitetura com os componentes do framework e uma visão comportamental deste. No final do capítulo, há uma visão geral da implementação dos componentes em um protótipo.

No capítulo 6, é explorado, com maior detalhamento, o protótipo implementado do framework, isto é, de que maneira o ciclo de vida de adaptação é gerenciado no framework, suporte a múltiplas heurísticas para adaptação, APIs de desenvolvimento e interface de administração.

Um estudo de caso que simula um cenário de uma aplicação de Clima Espacial, onde se aplica o uso do protótipo, é apresentado no capítulo 7. O estudo de caso apresenta o serviço selecionado para aplicação da adaptação, a arquitetura do EMBRACE, a utilização do framework Arctic Fox no estudo de caso, metodologia, análise dos

resultados e conclusões.

Considerações finais sobre os resultados do trabalho, comparativo com outras abordagens de adaptação, contribuições e trabalhos futuros são apresentados no capítulo 8.

O apêndice A apresenta os modelos de classes UML com os objetos centrais que representam o domínio de adaptação sugerido para framework, onde são também apresentados quatro diagramas de classes que representam os seguintes domínios de informação: Qualidade de Serviço, Elementos Arquiteturais (Componentes), Internet das Coisas (Dispositivos) e elementos do próprio framework (Heurísticas, Transações, Planos de Adaptação, Condições, etc).

2 SISTEMAS ESPACIAIS E O SEGMENTO SOLO

Neste capítulo são descritos os elementos que compõem os sistemas espaciais de modo geral, com foco maior de detalhe no segmento solo. Ainda no contexto do segmento solo, é apresentado o programa para Estudo e Monitoramento Brasileiro de Clima Espacial, objeto do estudo de caso deste trabalho. Também são abordadas neste capítulo novas arquiteturas de sistemas espaciais propostas para as mais diversas missões e os impactos que estas novas arquiteturas trazem na operação da missão.

Na seção final do capítulo, é promovida uma discussão sobre estes impactos nos sistemas de segmento solo, e também é explicitado um posicionamento sobre novos requisitos e aspectos relacionados à necessidade de maior autonomia dos sistemas, direcionando a discussão para o capítulo seguinte do trabalho.

2.1 Sistemas Espaciais

A vida moderna requer sistemas para promover comunicação, cobertura de TV, localização e orientação, estudos climáticos e científicos, observação da terra e do cosmo, defesa civil, monitoramento ambiental, dentre outras necessidades. De uma forma geral, essas necessidades requerem observação da Terra, e, consequentemente, para cada conjunto de necessidades, são definidas missões que são suportadas por sistemas espaciais.

Para continuar discorrendo sobre esse tema, é importante trazer a definição de sistema. Para National Aeronautics and Space Administration (NASA) (2007), um sistema é a combinação de elementos que funcionam em conjunto para produzir a capacidade de satisfazer uma necessidade. Os elementos incluem qualquer tipo de hardware, software, equipamentos, facilidades, pessoal, processos e procedimentos necessários para este propósito. O produto final (que desempenha funções operacionais) e os produtos acessórios (que fornecem serviços de suporte ao ciclo de vida dos produtos finais) também compõem um sistema.

Em termos conceituais, entende-se por programa espacial o desenvolvimento de uma série de funções atreladas a diferentes projetos que visam atender aos requisitos exigidos para a missão proposta. Uma missão espacial corresponde a um ou mais objetivos estabelecidos pelas partes envolvidas no processo, promovendo o desenvolvimento de um sistema onde parte dos objetivos são considerados, como a utilização de satélites ou de outros artefatos no espaço.

Um sistema espacial pode ser definido do ponto de vista de sua arquitetura. Para Larson e Wertz (1999), um sistema espacial é composto por um elemento observável, cargas úteis ou instrumentos que adquirem dados sobre o elemento observável, uma plataforma de serviços que provê suporte aos instrumentos, um lançador, uma órbita proposta, um sistema solo, arquitetura de comunicações e facilidades de operação.

De modo a atender aos objetivos de missão, os satélites artificiais são munidos de cargas úteis com instrumentos diversos, como câmeras óticas e infra-vermelho, espectrômetros, equipamentos de comunicação, sensores que são desenvolvidos para atender aos requisitos das diversas missões para as quais foram concebidos.

European Cooperation for Space Standardization (ECSS) (1996) observa que os sistemas espaciais são compostos dos seguintes segmentos: lançador, espacial e solo, os quais podem ser visualizados na figura 2.1, e cujo detalhamento vêm a seguir:

- **Segmento Lançador** (Launch Service Segment):

  Veículo Lançador (Launch Vehicle): Veículo responsável por injetar o segmento espacial em órbita.

  Plataforma de Lançamento (Launch Facilities): Plataforma, centros de controle, procedimento pré lançamento e recursos para iniciar o processo de lançamento do segmento espacial.

- **Segmento Espacial** (Space Segment):

  Plataforma de Serviços (Platform): Fornece a estrutura do segmento espacial, subsistema de armazenamento de energia, controle de órbita e atitude, comunicação, propulsores e subsistema térmico.

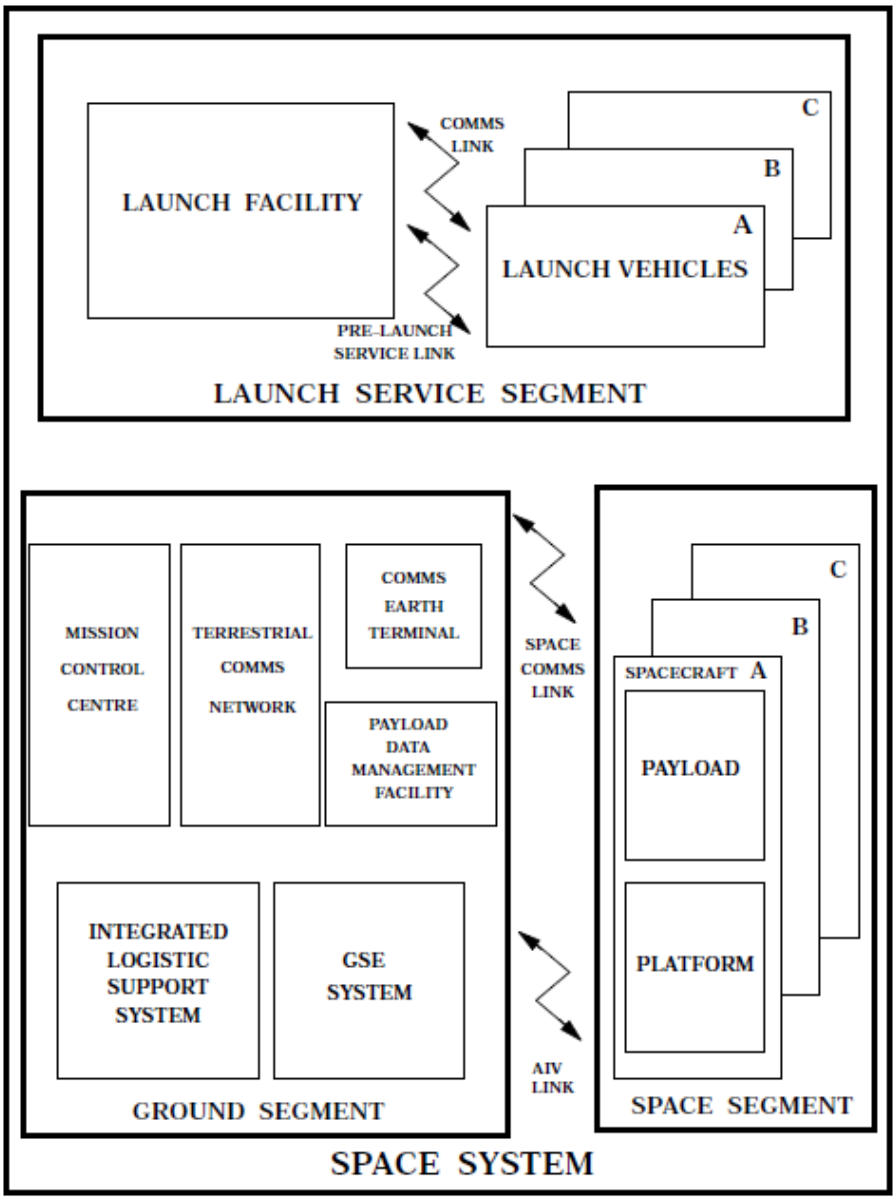
  Carga Útil (Payload): São os instrumentos projetados para a missão, como câmeras, espectrômetros, antenas de comunicação, magnetômetros, etc.

- **Segmento Solo** (Ground Segment):

  Estação Terrena : Responsável por abrigar as antenas de comunicação e os sistemas de rastreio e comunicação do segmento espacial com os centros de controle.

  Centro de Controle e Rastreio : Responsável por recepcionar telemetria do segmento espacial, enviar telecomandos, controle de voo e outras ações relacionadas à plataforma de serviços do segmento espacial.

Figura 2.1 - Elementos de um sistema espacial.



Fonte: European Cooperation for Space Standardization (ECSS) (1996).

Centro de Missão: Responsável pelo controle da carga útil do segmento espacial.

Centro de Disseminação de Dados: Responsável pela recepção e disseminação dos dados da carga útil.

Este trabalho de pesquisa tem como foco o segmento solo. Desta maneira, as próximas seções enfatizam o segmento solo, seus subsistemas, componentes e relações.

Um maior enfoque é atribuído à função de disseminação de dados dos instrumentos dos sistemas espaciais, que é objeto deste estudo.

2.2 Segmento Solo

Segundo Larson e Wertz (1999), para apoiar uma missão espacial real, um sistema terrestre deve cobrir simultaneamente diversos veículos espaciais em diferentes órbitas, com níveis de disponibilidade e segurança. Tais sistemas geralmente incluem muitos elementos em várias configurações.

O sistema solo compreende todos os sistemas terrestres que são usados para apoiar as atividades de preparação das operações da missão, a condução das próprias operações e todas as atividades pós-operacionais (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2008). Os sistemas de segmento solo são a parte do sistema espacial responsável por controlar, rastrear e recepcionar dados dos sistemas de segmento espacial a partir de sistemas terrestes.

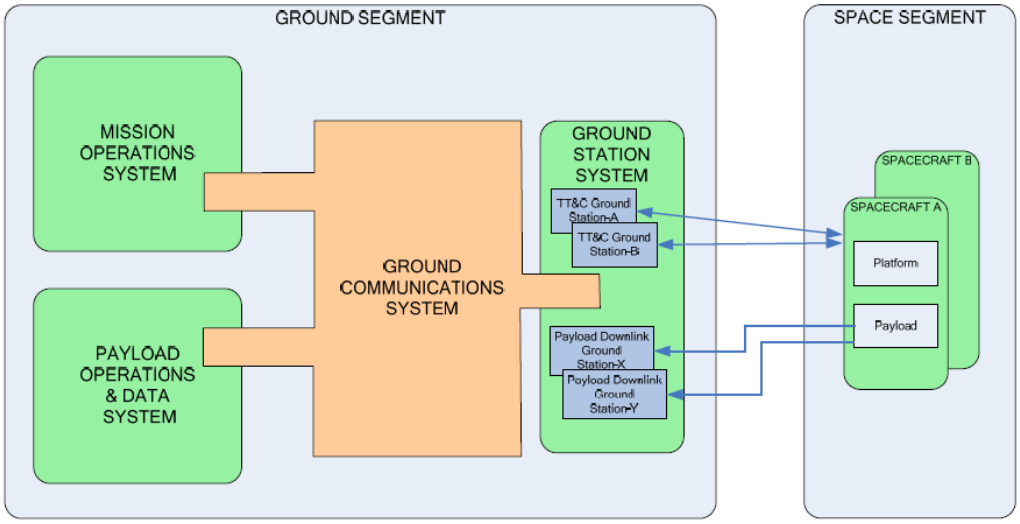
O segmento solo, apresentado na figura 2.2, representa melhor o senso de uma infraestrutura de solo, de acordo com European Cooperation for Space Standardization (ECSS) (1996), para várias missões. Consiste tipicamente dos sistemas que podem ser distribuídos entre vários centros, dependendo da arquitetura selecionada para cada missão, e consistem de sistemas para (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2008):

- Sistemas de operação e missão;
- Operações da carga útil e disseminação de dados;
- Estação terrena;
- Sistemas de comunicações de solo.

É visualizada na figura 2.2 uma representação dos elementos acima citados e seus relacionamentos de acordo com o padrão ECSS-70-C (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2008).

A maior parte dos subsistemas do segmento solo são responsáveis por rastreio e controle do segmento espacial e controle e recepção de dados das cargas úteis. As cargas úteis dos satélites fornecem a informação de que os usuários necessitam para desempenhar as duas atividades. O Centro de Controle e Operação de Missão, responsável pelas atividades de controle e rastreio do segmento espacial, é detalhado

Figura 2.2 - Sistemas que compõem o Segmento Solo.



Fonte: European Cooperation for Space Standardization (ECSS) (2008).

na próxima seção. O Centro de Dados Científicos, por sua vez, que é responsável por disseminar a informação dos instrumentos para os usuários, é detalhado na seção seguinte, com foco no programa de clima espacial, missão selecionada para o estudo de caso.

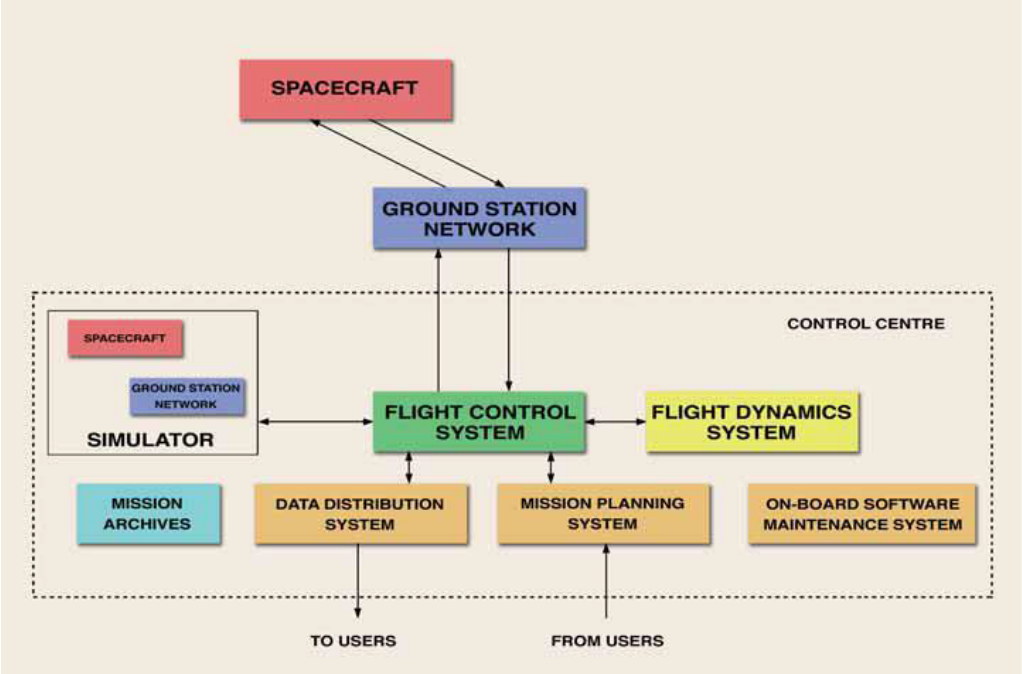
2.2.1 Centro de Controle e Operação de Missão

Um maior detalhamento dos componentes de controle da missão pode ser observado na figura 2.3, de acordo com o que estabelece a agência espacial européia ESA (EUROPEAN SPACE AGENCY (ESA), 2006). Um sistema de segmento solo é composto dos seguintes subsistemas:

Um sistema de segmento solo é composto dos subsistemas:

- Estação Terrena (*Ground Station Network*);
- Sistema de Controle de Vôo (*Flight Control System*);
- Sistema de Dinâmica de Vôo (*Flight Dynamics System*);
- Simulador de Segmento Espacial (*Spacecraft Simulator*);
- Simulador de Estação Terrena (*Ground Station Network Simulator*);
- Sistema de Distribuição de Dados (*Data Distribution System*);

Figura 2.3 - Componentes típicos do Segmento Solo com foco no Controle da Missão.



Fonte: European Space Agency (ESA) (2006)

- Sistema de Planejamento de Missão (*Mission Planning System*);
- Arquivos de Missão (*Mission Archives*);
- Sistema de Manutenção de Software de Bordo (*On-Board Software Maintenance System*).

Uma descrição da funções dos principais componentes de um sistema de segmento solo é verificada na tabela 2.2.1.

Tabela 2.1 - Funções dos Componentes de Sistemas Solo.

Componente	**Função**
Preparação de Operações	Preparação e manutenção de
	- Plano de operação de vôo (FOP)
	- Banco de dados de operações
	- Banco de dados de planejamento de missões.
Planejamento de Missão	Processamento de entradas de planejamento externas.
	Geração e validação de timelines de operação de satélites.

Continua na próxima página

Tabela 2.1 – *Funções dos Componentes de Sistemas Solo - Continuação*

Componente	**Função**
Controle de Rede	Controle de ligação (link) entre o centro de controle e a estação terrana para: - dados de telemetria - telecomandos - informação de rastreamento de satélites
Monitoramento do segmento espacial	recepção de telemetria da estação terrena. correlação de tempo espaço-solo. Conversão de telemetria de satélites para valores de engenharia. verificação de limites e monitoramento da situação das telemetrias dos satélites. Visualização de dados de satélite em mogo gráfico ou numérico em tempo real ou em modo de rápido avanço/retrocesso acesso on-line a dados históricos do satélite.
Telecomando do segmento espacial	Geração de mensagens de comando para todos os tipos de comandos de satélite. liberação on-line ou agendada de comandos para a estação terrena encaminhar ao satélite. verificação da pré-transmissão de comandos. verificação das ações comandadas pós-transmissão. log e visualização de histórico de comandos.
Facilidades de Usuário	Controle de acesso de usuário baseado em privilégios.
Software de bordo	Manutenção de software de bordo Validação das alterações do software antes de embarcar no segmento espacial
Distribuição de dados	distribuição de dados on-line e off-line para usuários externos (outros centros de controle, centros de dados, institutos de pesquisa).
Análise de desempenho	Acessar e recuperar dados históricos de satélite. Visualizar e analisar periodicamente algoritmos definidos pelo usuário. geração de relatórios de operação.
Determinação de Atitude e Controle	Processar dados de telemetria, Estimar altitude, preparar manobras de atitude, Momento angular e controle de combustível
Calibração	calibração de atuadores e propulsores, calibração de sensores óticos, calibração de “Gyro“
Determinação de Órbita e Controle	processamento de dados de rastreamento calculos de elementos orbitais previsão de visibilidade na estação terrena, períodos de eclipse, etc preparação para apontamento de antenas preparação de manobras orbitais consumo de combustível
Apoio ao Planejamento de Missão	Geração de informação relacionada planejamento orbital preparação de planos de missão relativos a atitude e órbita previsão de momento angular e consumo de combustível.
Geração de Dados Auxiliares	geração de dados históricos de atitude geração de dados relacionados a órbita geração de dados relacionados ao segmento espacial.

Continua na próxima página

Tabela 2.1 – *Funções dos Componentes de Sistemas Solo - Continuação*

Componente	**Função**
Garantia da Qualidade	validação independente de sistemas
	Supervisão do desempenho e qualidade de toda a dinâmica de vôo
	Verificação da qualidade e desempenho da dinâmica de vôo.
Subsistemas do satélite	Representação de hardware, software, mecanismos, motores, etc.
	telemetria realistica em todos os modos
	respostas corretas a comandos.
Sistemas de software de bordo	emulação de hardware
	software de bordo executável
	dinâmicas do satélite
	propriedades de massa do satélite
	execução de manobras de atitude
	sensores de atitude
Ambiente	manobras orbitais
	entradas térmicas e eclipse
	períodos de contato com a estação terrena
	catalogos de estrelas (incluindo planetas)
Interfaces de Rede	Estaçõs terrenas
	Comunicação.
Controle do Simulador	Mais rápido do que em tempo real
	injeção de anomalias pelo operador
	monitoramento do simulador
	facilidades para re-envio(replay) de comandos e telemetrias.

Fonte Adaptada de European Space Agency (ESA) (2006)

Toda a informação recepcionada pelos sistemas de segmento solo deve ser processada, armazenada e distribuída para os usuários. Neste contexto, é necessário um centro de disseminação de dados (EUROPEAN SPACE AGENCY (ESA), 2006) (MATIELLO-FRANCISCO, 2003), o qual é detalhado na seção seguinte, com enfoque no programa brasileiro de estudo e monitoramento de clima espacial.

2.2.2 Centro de Dados Científicos

A utilização das informações facultadas pelos sistemas de segmento solo em contextos integrados tem um papel cada vez mais importante em vários setores. Estas informações, complementadas com outras tecnologias convergentes, tais como a localização baseada em sistemas GNSS e as comunicações, oferecem um amplo leque de sistemas e serviços às comunidades de usuários em diversos setores, tais como os transportes, a segurança, a agricultura, a defesa, o tráfego, o planejamento urbano, o desenvolvimento de infra-estruturas, previsões climáticas, energia, óleo e gás, etc.

Segundo Matiello-Francisco (2003) centros de missão se destinam as tarefas de ar-

mazenamento e disponibilização dos dados da missão aos usuários. Nesse contexto, a exploração dos dados de missão deve ser definida para atender aos requisitos dos usuários das missões espaciais, e isso inclui processamento de dados, bancos de dados de produtos, bancos de dados de usuários e disseminação de dados (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2008).

Neste contexto de disseminação de dados de missões espaciais, enquadra-se o programa de Estudo e Monitoramento Brasileiro de Clima Espacial (EMBRACE), cujos usuários necessitam de previsões sobre o impacto de fenômenos solares no funcionamento dos satélites para poder, desta maneira, permitir a adoção de medidas preventivas de modo a reduzir o risco na operação destes usuários.

A área de Clima Espacial estuda os fenômenos solares que afetam o meio entre o Sol e a Terra, e o espaço em torno da Terra. Nestes ambientes espaciais, podem ocorrer fenômenos capazes de causar interferências em sistemas de satélites de posicionamento, como o GPS, e sensoriamento remoto por radar, além da possibilidade de induzir correntes elétricas em transformadores de linhas de transmissão de energia e afetar a proteção de dutos para transporte de óleo e de gás.

O EMBRACE é um programa cujo objetivo principal é realizar o monitoramento do espaço Sol-Terra, magnetosfera e ionosfera-atmosfera por satélite e observações terrestres. Em paralelo, viabiliza o fornecimento de informações úteis às comunidades acadêmicas, áreas industriais e tecnológicas (TAKAHASHI et al., 2008).

Localizado no município de São José dos Campos, o EMBRACE implementou uma arquitetura responsável pelo armazenamento, processamento e disseminação de dados de diversos tipos de equipamentos geograficamente distribuídos, tais como: magnetômetros, ionossondas, redes de GPS, dados de rádio e raios X de explosões solares, entre outros. Para que os dados cheguem até à comunidade, diversos meios para sua disponibilização foram desenvolvidos para cada equipamento, e podem ser visualizados através das páginas web contendo gráficos, tabelas, download dos dados brutos, e processados para cada tipo de equipamento, web services para integração entre sistemas tecnologicamente distintos, boletins diários com informações atualizadas (sol, Meio Interplanetário, Terra/Atmosferas, Terra/Campo Magnético), aplicações para celulares e outros.

No trabalho de SantAnna et al. (2014), é possível observar um modelo denominado pipeline, o qual descreve todo o fluxo desde a coleta até a disponibilização dos dados. Resumidamente, os dados brutos coletados são disponibilizados em uma área

para que um agente coletor realize a abertura e pré-processamento das informações, identificando seu tipo, sensor, origem e outros. Em seguida, os dados brutos e processados são persistidos em uma base de dados. Em algumas aplicações do programa EMBRACE, os dados pré-processados são utilizados para a geração de novas informações, como os dados de GNSS, que são responsáveis por calcular o conteúdo total eletrônico suspenso na ionosfera, sendo disponibilizados em tempo real ou não. A visão geral da arquitetura é composta pelos seguintes componentes:

- **Collector Agent**: embarcado no sistema de arquivos, o agente coletor tem como função realizar o pré-processamento dos arquivos, formatação em formato de linguagem estruturada XML, compressão e envio dos dados ao ReceptorGW;

- **ReceptorGW**: responsável por colocar na fila de mensagens os dados recebidos pelo Collector Agent;

- **Consumer**: consome os dados que estão na fila de mensagens e armazena-os no banco de dados;

- **Processing Queue**: API (*Application Programming Interface*) encarregada de permitir que duas ou mais aplicações se comuniquem por mensagens, garantindo a interoperabilidade e possibilitando a leitura, criação, envio e o recebimento de dados através de mensagens;

- **Core**: possui interfaces que permitem o acesso à base de dados;

- **Scheduler**: agendador executor de processos responsável em verificar conjuntos de dados que necessitam de um pós processamento;

- **Viewer**: camada de visualização permite exibir os dados em forma de tabelas, gráficos, mapas e índices.

De modo a destacar o aumento da complexidade das arquiteturas de sistemas espaciais, são apresentadas, na seção seguinte, diversas referências de missões espaciais modernas onde são observados diversos cenários relacionados à operação autônoma das missões espaciais.

2.3 Missões Espaciais Modernas

2.3.1 Constelações, Mega Constelações e Enxames

Novos sistemas espaciais para as mais diversas finalidades visam impulsionar o desenvolvimento da humanidade para novos horizontes. Entre as missões pelas quais esses sistemas são desenvolvidos estão a observação do cosmo em busca de exoplanetas que possam abrigar vida e servir como uma segunda casa para a espécie humana, a observação da galáxia por meio do uso de grandes telescópios montados no espaço de forma autônoma devido às restrições de capacidade dos lançadores, e a cobertura de sinal de internet banda larga para as regiões mais longínquas do globo terrestre.

Para levar internet banda larga para todo o globo terrestre, observa-se em Airbus Space & Defense (2016) e Virgilia et al. (2016) a produção de 900 satélites para fornecer internet de alta velocidade com cobertura de 100% do território. Para atingir esse objetivo, está prevista para iniciar-se em 2018 a injeção, na baixa órbita terrestre, de uma mega constelação de 648 mini-satélites com alta taxa de revisitação. Para isso, já foram definidas plantas industriais para produção dos satélites nos EUA e Europa, bem como os fornecedores dos lançadores.

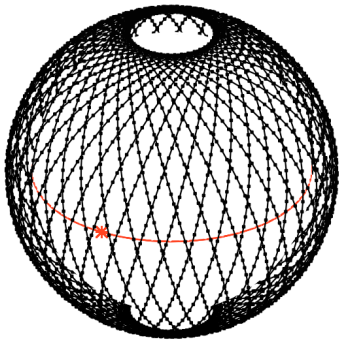
Mega-constelações produzem um impacto significativo na indústria espacial, desde a capacidade de produção dos satélites até o sequenciamento e coordenação da injeção dos satélites em larga escala nos planos orbitais, alta taxa de revisitação, crescente produção de detritos espaciais devido ao volume de artefatos na baixa órbita, colisões, entre outros.

Mega-constelações indicam o lançamento, em um curto período de tempo, de 100 a 5000 satélites em órbita para cumprimento de suas respectivas missões (KITAJIMA et al., 2016). Esse tipo de arranjo pode ser observado na figura 2.4. Dentre os impactos relacionados a estas constelações estão a avaliação de colisões, a produção de detritos, a segurança da baixa órbita terrestre.

Este mesmo artigo avalia diferentes cenários com a injeção progressiva de satélites em órbita, o tempo de duração das missões, bem como o volume de detritos gerados, tanto pelos satélites em si como também partes de lançadores desconectadas em determinadas altitudes. Também é levada em consideração a substituição dos satélites por equipamentos de gerações mais novas. São levantados e analisados dados de cenários simulados relativos ao aumento do volume de objetos, a estabilização da quantidade de objetos na órbita baixa terrestre. Nesse sentido, são avaliadas em

quais altitudes as colisões ocorrem com maior frequência de impacto do descarte de objetos após 25 anos em órbita.

Figura 2.4 - Mega-constelação com 4000 satélites.



Fonte: Kitajima et al. (2016)

Algumas observações interessantes verificadas em Virgilia et al. (2016) são listadas a seguir:

a) As constelações de satélites com baixa massa e área pequena são melhores porque encorajam tempos de decaimento mais curtos, reduzem a seção transversal da colisão e reduzem o número de fragmentos que podem ser gerados em caso de colisão;

b) A separação dos planos orbitais da constelação pode ajudar a reduzir o risco de colisão se o número de satélites por plano permanecer inalterado em relação ao nominal;

c) Ao sucesso com que os satélites são descartados deve ser dada a mais alta prioridade; isso, porém, não anula a importância do descarte bem sucedido para qualquer outro satélite ou estágio superior devido à interação potencial entre as duas populações;

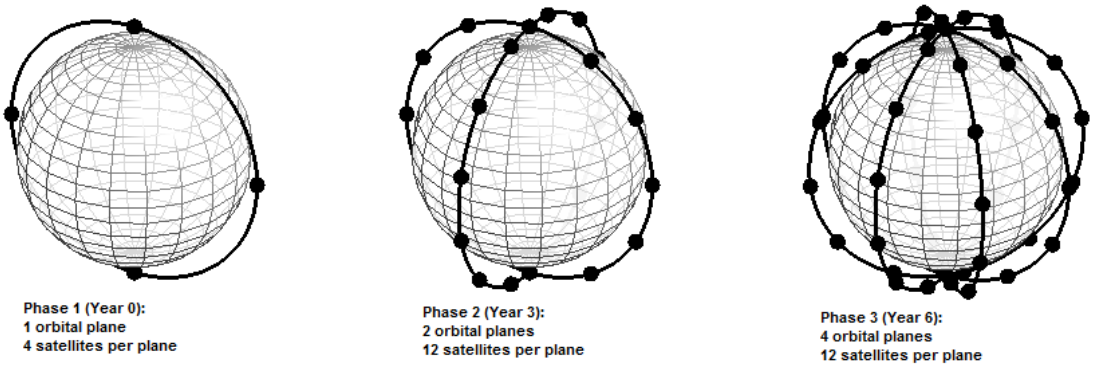
d) A redução da vida útil residual dos satélites da constelação proporciona benefícios a curto prazo, mas não é suficiente, por si só, para limitar o crescimento da população de detritos a longo prazo caso as taxas de sucesso de descarte pós missão não sejam suficientemente elevadas;

e) Os satélites de constelação colocados com sucesso em órbitas de eliminação podem, ainda, interagir com a população de fundo em altitudes inferiores a

700 km - o que é exacerbado pela implantação de um número relativamente elevado de pequenos satélites nessa altitude.

Para o problema da injeção em órbita progressiva de satélites (PALERMO; GAUDENZI, 2016), observa-se o desenvolvimento de uma estratégia para implantação que leva em conta diferentes arquiteturas de sistemas e objetivos múltiplos de missão. Na figura 2.5, é apresentada uma estratégia com duração de oito anos para injetar quarenta e oito satélites em quatro planos orbitais com três fases.

Figura 2.5 - Implantação progressiva de satélites para uma constelação de 48 satélites.



Fonte: Palermo e Gaudenzi (2016)

Há avanços não apenas na pesquisa e desenvolvimento de mini-satélites, constelações e mega constelações, mas também com relação às cargas úteis, que estão evoluindo. Para a exploração de exoplanetas, observa-se o mapeamento de mais de três mil planetas fora do sistema solar.

A missão Twinkle (TESSENYI et al., 2016) observará a composição química e o clima de mais de cem exoplanetas na Via Láctea, incluindo Super-Terras (planetas rochosos com um a dez vezes a massa da Terra), Netunos, sub-Netunos e gigantes de gás como Júpiter. Será também capaz de acompanhar observações fotométricas de mais de mil exoplanetas por meio de espectrômetros ópticos e infravermelho, bem como de realizar a observação de objetos do Sistema Solar, estrelas brilhantes e discos.

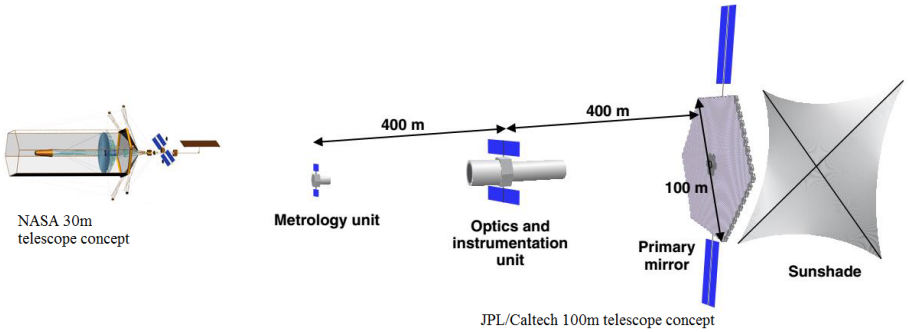
Os satélites Twinkle serão lançados em uma órbita polar sol-síncrona na órbita baixa da terra a partir de 2019. A carga útil do satélite é composta por um espectrógrafo infravermelho e outro óptico.

Ainda no campo da observação do cosmo, além de outras necessidades de missões como maior resolução de imagens ou melhorar a sensibilidade de instrumentos verifica-se nas arquiteturas de sistemas espaciais atuais as seguintes limitações (SAUNDERS et al., 2016):

- As arquiteturas atuais para uma missão espacial estão limitadas pela massa e volume disponíveis nos atuais lançadores. Uma missão é tipicamente lançada "de uma só vez".
- Há um desejo constante para a construção de instrumentos maiores, principalmente para fins de maior resolução ou sensibilidade.
- Para os domínios ótico e infravermelho, este é um desafio significativo.
- Telescópios, por exemplo, são constituídos de espelhos monolíticos; uma alternativa seria a construção de telescópios com espelhos fracionados, isto é, uma coleção de pequenos elementos que são montados em órbita (automontagem), de modo a formar um instrumento maior.
- A pesquisa visa inovar as missões espaciais com a montagem em órbita promovendo a redução de custos.

Na figura 2.6, é apresentado um exemplo de uma nova arquitetura onde é possível a escala de um telescópio com o uso de espelhos fracionados e montagem em órbita.

Figura 2.6 - Escala de instrumentos em tamanho com montagem em órbita.

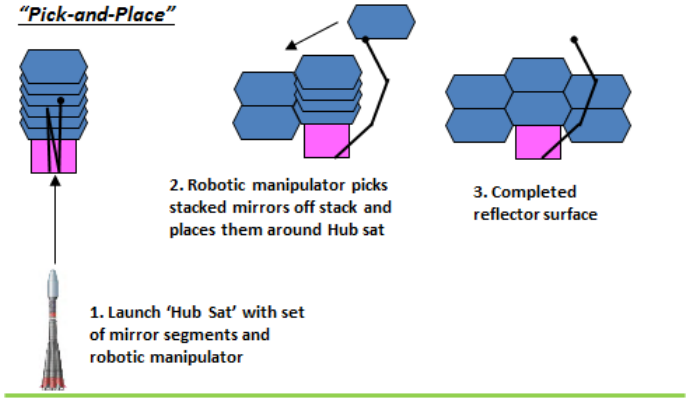


Fonte: Saunders et al. (2016)

Para obter o telescópio de alta resolução face às limitações de capacidade dos

veículos lançadores atuais, é apresentado, na figura 2.7, o processo de acoplamento/montagem robótica e autônoma em órbita do espelho do telescópio.

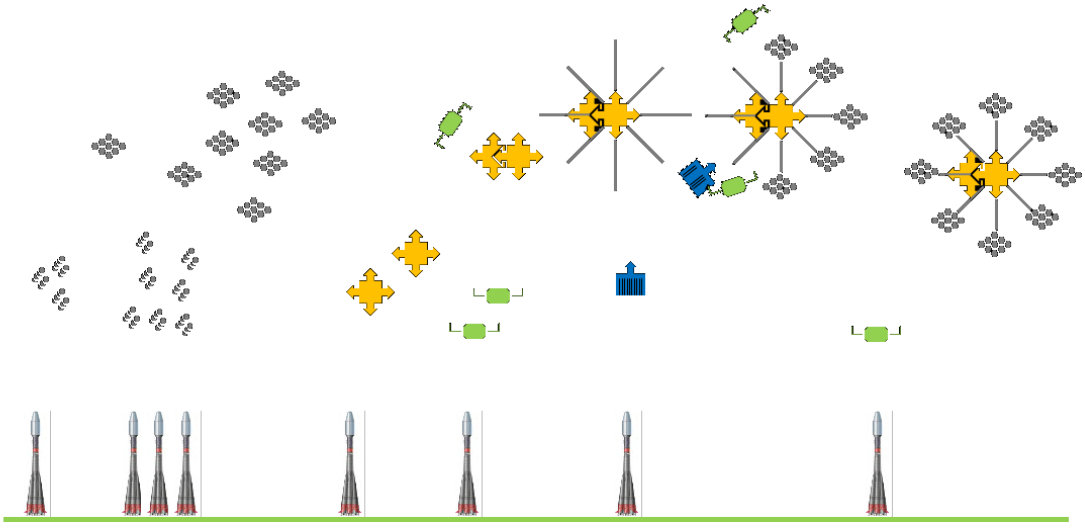
Figura 2.7 - Acoplamento robótico de fragmentos em órbita.



Fonte: Saunders et al. (2016)

Na figura 2.8, é apresentado o processo completo com vários lançamentos e montagens em sequência.

Figura 2.8 - Processo de montagem autônoma em órbita.



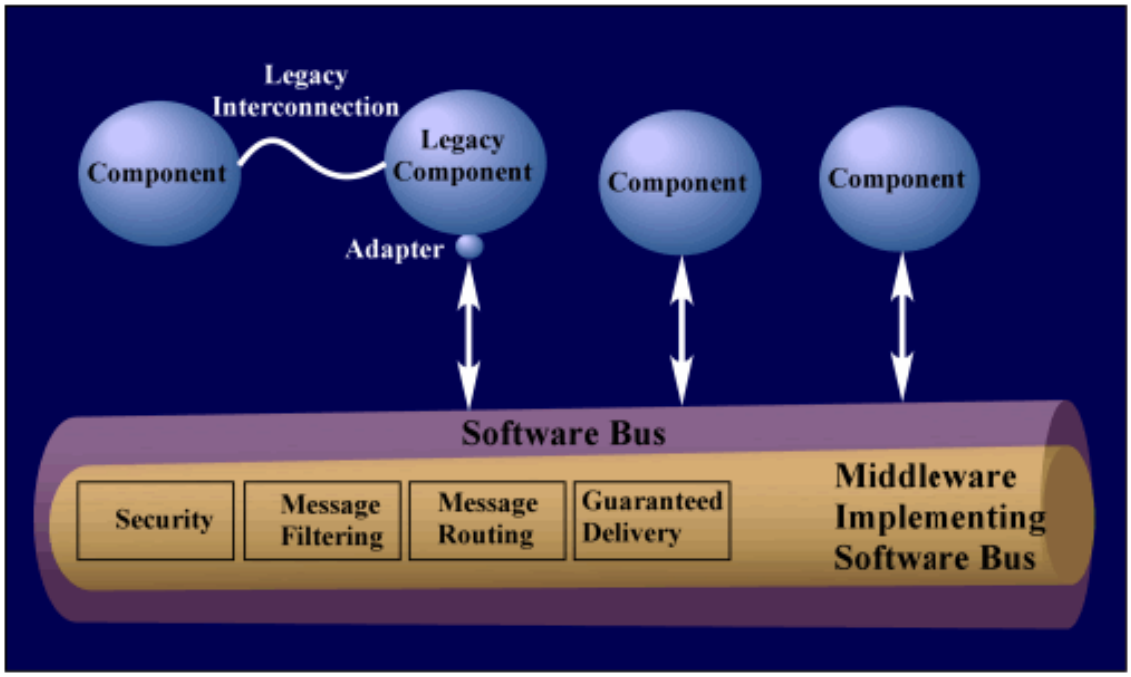
Fonte: Saunders et al. (2016)

2.3.2 Sistemas Espaciais Autônomos

Esta seção se destina a apresentar algumas soluções pesquisadas na literatura sobre a aplicação de soluções em sistemas espaciais autônomos.

Uma solução relacionada a aspectos de auto-configuração e auto-cura para sistemas solo de operação e controle de satélites é definido em Li (2006). Os autores apresentam uma arquitetura de adaptação baseada em regras e trocas de mensagens para integração de diversos componentes distribuídos e instalados em diversos equipamentos de hardware para operações de missões de rastreamento e controle de satélites.

Figura 2.9 - Arquitetura GMSEC para controle autônomo de satélites.



Fonte: Li (2006)

A Arquitetura GMSEC (LI, 2006), apresentada na figura 2.9, é baseada no estilo arquitetônico Publish-Subscribe, no qual agentes autônomos locais e globais encaminham e lêem informações do barramento de software. Baseado no conhecimento do contexto e nas regras de negócio definidas, os agentes autônomos tomam as suas ações.

A arquitetura GMSEC, bem como a ferramenta autonômica CAT (LI, 2006), foram

implantadas em muitas missões da NASA no intuito de incrementar a automação e a autonomia, reduzir o custo operacional e tornar-se um padrão de desenvolvimento para os sistemas solo atuais e para as futuras missões da NASA.

A solução de computação autônoma para sistemas solo é utilizada para substituição do pessoal de operações em tarefas de monitoramento e manobras de satélites. As capacidades de auto-configuração e auto-cura dos elementos autonômicos são cruciais para operações totalmente autônomas ou "Lights Out".

Uma prova de conceito, para adaptação de sistemas espaciais, baseado em swarm computing (VARGHESE; MCKEE, 2009), apresenta a necessidade de construção de sistemas confiáveis para aplicações espaciais e sistemas auto-gerenciáveis. Nesse sentido, a computação autônoma é considerada. Nesse contexto, o trabalho considera recursos computacionais como um enxame "swarm" de recursos, e considera as tarefas a serem executadas como um enxame de tarefas.

O processo inicia-se com o monitoramento da vizinhança e, independentemente do que estiver em processamento, cada agente investiga outros componentes, questionando: Vocês estão vivos? e, a partir deste ponto, começam a coletar informações do contexto do sistema. No caso de algum componente falhar, o processamento deve ser movido para outro recurso, e as dependências devem ser re-estabelecidas. A viabilidade da proposta foi validada em um simulador multi-agentes.

Um projeto da ESA denominado IRONCAP (STEEL et al., 2012) procura explorar e definir os conceitos, técnicas e interações necessários para planejar e programar as atividades de um rover interplanetário. Seu objetivo está em desenvolver um protótipo de sistema de apoio às atividades de ciência e engenharia de planejamento de um rover interplanetário, utilizando métodos e técnicas de planejamento e agendamento em conjunto com os sistemas solo e tecnologias existentes e/ou em desenvolvimento. O protótipo vai apoiar a análise situacional do rover, facilitar o planejamento e a programação de atividades, assim como a observação para os níveis de autonomia aplicáveis, além de apoiar as equipes em suas atividades diárias.

Futuras missões da NASA (HINCHEY et al., 2005) irão explorar novos paradigmas para a exploração do espaço, e estarão fortemente focadas nas emergentes tecnologias de sistemas autônomos. Conceitos de missões espaciais tradicionalmente dependentes de uma grande espaçonave (satélite ou sonda), estão sendo complementados com missões conceitos que envolvem várias espaçonaves menores, as quais operam em colaboração, analogamente aos enxames na natureza. Esta abordagem oferece várias

vantagens, entre elas: a capacidade de enviar naves espaciais para explorar regiões do espaço para onde seria impraticável o envio de espaçonaves tradicionais, maior redundância (e, consequentemente, maior proteção de ativos), e redução de custos e risco. Citam-se como exemplo os projetos:

- **SARA: Saturn Autonomous Ring Array** (HINCHEY et al., 2005), lançará 1000 pico-satélites, organizados em dez sub-enxames, cada um com instrumentos especializados, para exploração local dos anéis de Saturno, de modo a compreender sua constituição e de que maneira eles foram formados. A missão vai exigir a auto-configuração das estruturas de propulsão e de controle, de modo a possibilitar a operação autônoma de manobras em torno dos anéis de Saturno, e também para evitar colisões;
- **PAM: Prospecting Asteroid Mission** (HINCHEY et al., 2005), também lançará 1000, pico-satélites, porém, com o objetivo de explorar o cinturão de asteroides e recolher dados sobre determinados asteroides de interesse;
- **LARA: ANTS Application Lunar Base Activities** (HINCHEY et al., 2005), projeto para exploração de novas tecnologias NASA desenvolvidas no domínio da robótica miniaturizada, que podem formar a base de rovers remotos enviados à lua a partir de bases remotas. O projeto explora técnicas inovadoras para permitir rovers a se mover em terrenos irregulares da lua.

2.4 Necessidades de Missões Espaciais

Sistemas espaciais concebidos para diversas missões são compostos dos seguintes segmentos: solo, espacial, lançador e usuário. Nesse contexto, são identificados os desafios para as novas gerações de sistemas espaciais, nas quais verificam-se as seguintes necessidades:

- Necessidade de interconexão de diversos equipamentos via comunicação solo-espaço.
- Necessidade de interconexão de diversos equipamentos via comunicação espaço-espaço.
- Necessidade de interconexão de diversos equipamentos via comunicação solo-solo.

- Equipamentos tais como antenas, instrumentos, computadores de bordo e outros, que possuem capacidade de processamento inferior à de servidores de mercado.
- Necessidade de controle e rastreio de plataformas de constelações de satélites.
- Necessidade de controle de cargas úteis.
- Necessidade de processamento massivo de telemetrias.
- Necessidade de geração de telecomandos para diversas plataformas espaciais.
- Necessidade de geração de telecomandos para diversas cargas úteis.
- Necessidade de processamento de dados de cargas úteis em tempo real.
- Necessidade de monitoramento de parâmetros de qualidade de serviço de componentes de segmento solo.
- Necessidade de monitoramento de parâmetros de qualidade de serviço de componentes de segmento espacial.
- Necessidade de otimização de recursos.

2.5 Discussão sobre o impacto de novas tecnologias em Sistemas de Segmento Solo

Observa-se que, devido aos esforços para a redução de custos e inovações tecnológicas acrescidas nos sistemas espaciais atuais apresentados neste capítulo, há grandes desafios aos sistemas solo atuais, principalmente na realidade do INPE.

No caso dos sistemas responsáveis pela disseminação de dados, o advento das constelações de pequenos satélites, cubesats e pico-satélites, pode gerar uma massa de dados volumosa com consequências diretas para a operação da missão. Esta realidade implica diretamente no aumento da complexidade das arquiteturas dos sistemas de segmento solo, os quais são intensivos em software e, cada vez mais, será necessário prever requisitos para controle de constelações e mega-constelações de satélites, bem como a atualização dinâmica de informação de cargas úteis e plataforma de satélites, além do monitoramento da situação operacional destes elementos. Este cenário implica na necessidade de levantamento de requisitos relacionados à operação autônoma, e aumentam as exigências de qualidade de serviço dos sistemas espaciais.

Os requisitos relacionados à operação autônoma de sistemas solo devem ser previstos no desenvolvimento dos projetos. Neste trabalho, são discutidos em maior detalhe aspectos relacionados a: arquiteturas de sistemas de software, sistemas adaptáveis, qualidade de serviço, bem como a proposta de *framework* para fornecer suporte à auto-adaptação de sistemas solo.

3 ARQUITETURAS DE SOFTWARE AUTO-ADAPTÁVEIS

Este capítulo visa apresentar os conceitos relacionados a arquiteturas de software e os conceitos relacionados à auto-adaptação de software centrado em arquitetura. No final do capítulo, é apresentada uma seção estabelecendo uma discussão sobre a aplicação dos conceitos apresentados no software de segmento solo, bem como um direcionamento sobre o framework objeto deste trabalho de pesquisa.

3.1 Arquiteturas de Software

Cervantes e Kazman (2016) definem que arquitetura de um programa de computador ou sistema de software constitui a estrutura ou estruturas do sistema, composta(s) por elementos de software, as propriedades externas visíveis dos referidos elementos, e as relações entre eles. É a organização fundamental de um sistema, consubstanciado em seus componentes, as relações entre eles, e com o ambiente, bem como os princípios que regem a sua concepção e evolução (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE), 2000).

A arquitetura de um sistema de software é o conjunto das principais decisões de projeto definidas para o sistema; uma arquitetura de referência, por sua vez, é o conjunto das principais decisões de projeto simultaneamente aplicáveis a vários sistemas relacionados, normalmente dentro de um domínio de aplicação, com pontos de variação explicitamente definidos.

Segundo Taylor et al. (2009), pode-se desenvolver uma analogia entre arquiteturas utilizadas para a construção de prédios e as arquiteturas de software. Na construção civil, para a concepção de uma arquitetura de um dado edifício, requisitos são coletados, um projeto é definido e refinado em desenhos técnicos, a construção é baseada nos desenhos e, então, é ocupada e utilizada. Para o domínio do software, requisitos são especificados, um projeto de alto nível é criado, algoritmos detalhados são desenvolvidos, baseados no projeto, códigos são escritos para implementar os algoritmos e, finalmente, o sistema é implantado e utilizado.

A arquitetura de edifícios e construção civil apresenta algumas idéias que podem ser refletidas no domínio do software. Por exemplo, o processo de arquitetura de edifícios tem como foco a satisfação das necessidades do futuro ocupante. Ele permite a especialização do trabalho, isto é, o projetista da estrutura não precisa ser o empreiteiro que executa a construção. O processo tem muitos pontos intermediários, onde tanto os planos quanto o progresso podem ser revistos.

Na visão de desenvolvimento de software, seu ciclo de vida prevê que a especificação de requisitos de um sistema precede sua concepção; que o projeto é criado por especialistas e não pelos usuários finais de um sistema. A codificação do software pode ser terceirizada. Protótipos e maquetes podem ser criados em vários pontos do desenvolvimento, para permitir que o cliente possa avaliar antecipadamente se o sistema em desenvolvimento, de fato, atende as necessidades especificadas.

Desta maneira, identifica-se um conjunto de percepções sobre arquiteturas: A primeira percepção é a de que toda construção possui uma arquitetura que pode seguir um conceito separado, como a adoção de um padrão arquitetônico, porém conectado à própria estrutura física da edificação. A arquitetura de uma edificação, isto é, seus elementos principais, composição e organização, podem ser descritos, discutidos e comparados a outras edificações.

A segunda percepção é a de que as propriedades das estruturas são induzidas pelo projeto de suas arquiteturas. Como exemplo, cita-se um castelo medieval que possui paredes altas e espessas com janelas estreitas ou mesmo inexistentes; observa-se, neste exemplo, que a edificação em questão possui excelentes propriedades defensivas (contra ataques de exércitos armados com espadas e flechas). Algumas aplicações de software possuem algumas propriedades como resistência contra alguns tipos de ataques de segurança, determinadas pelo projeto de suas arquiteturas.

A terceira percepção é o reconhecimento do papel central de um arquiteto, o responsável pela criação da arquitetura. A disciplina de arquitetura reconhece, há muito tempo, que os arquitetos necessitam de uma formação ampla. É necessário um fino senso de estética, profunda compreensão de como as pessoas trabalham, divertem-se, comem e vivem, de modo que o projeto da arquitetura e a consequente criação de edifícios satisfaçam os seus ocupantes e sejam ocupados de forma eficaz ao longo dos anos. De maneira semelhante, habilidades de programação não são suficientes para a criação de sistemas de software complexos que as pessoas possam efetivamente utilizar.

A quarta percepção é a de que o processo não é tão importante quanto a arquitetura. Isto não quer dizer que processo não seja importante. Pelo contrário, arquitetos e empresas de construção claramente dependem de processos padrão para orientar suas atividades diárias e garantir que todos os aspectos do projeto, e respectivas atividades, sejam abordados. Mas não existe nenhuma dúvida de que arquitetura seja o foco central.

Seguir um processo padrão não garante que um edifício, após construído, atenderá às necessidades de seus proprietários e ocupantes com sucesso. Os arquitetos e engenheiros responsáveis pela estrutura devem manter seu projeto com qualidade e na vanguarda da construção civil. O processo está presente para servir a esses fins.

A quinta percepção ou visão é a de que a arquitetura amadureceu ao longo dos anos. Este amadurecimento gerou um corpo de conhecimento existente sobre como criar uma ampla gama de edifícios para atender a vários tipos de necessidades. O corpo de conhecimento vai além de princípios básicos e processos de desenvolvimento genéricos.

Uma maneira de capturar experiências e lições de gerações anteriores de arquitetos e construtores resume-se na noção de estilos-arquitetônicos, um conceito com poderosa aplicação no domínio de software. As frases “vila romana”, “catedral gótica”, “casa de estilo fazenda”, “chalé suíço” e “arranha-céu nova iorquino” caracterizam tipos de edificações que têm várias características em comum. Para entender melhor o conceito de estilo arquitetônico, cita-se o exemplo dos chalés suíços construídos em regiões com invernos rigorosos e neve em grande volume, pois são tradicionalmente feitos de madeira, têm telhados íngremes, e amplas varandas que compõe um estilo arquitetônico próprio. As ideias apresentadas sobre arquitetura, arquitetos e estilos arquitetônicos em particular, permitem uma melhoria substancial da qualidade e produtividade na criação de sistemas de software, mas demandam um estudo cuidadoso, boas ferramentas e uso disciplinado das técnicas, que podem se traduzir nos potenciais benefícios descritos a seguir:

- Integridade conceitual;
- Uma base adequada e eficaz para reuso, a partir de conhecimentos, experiências, projetos e código obtidos em projetos anteriores;
- Comunicação eficaz no projeto;
- Gestão de um conjunto relacionado de variantes de sistemas.

Das percepções observadas nesta seção, extrai-se três conceitos que auxiliam a situar a arquitetura dentro da engenharia de software:

a) Cada aplicação possui uma arquitetura;

b) Cada aplicação possui ao menos um arquiteto;

c) A arquitetura não é uma fase do ciclo de desenvolvimento de software.

3.1.1 Estilos Arquitetônicos

Estilos arquitetônicos são uma forma primária de caracterizar lições a partir de experiências em projetos; podem ser um elemento-chave no desenvolvimento inicial ou concepção detalhada da arquitetura de um sistema de software. Estilos arquitetônicos são amplamente aplicáveis, podem ser úteis na determinação da estrutura de uma sub-rotina ou no nível superior de uma estrutura de aplicação.

Definição:

"Um estilo arquitetônico é uma coleção nomeada de decisões de projeto que são aplicáveis em um determinado contexto de desenvolvimento, restringem decisões de projetos de arquitetura de um sistema em particular para um contexto específico e extraem qualidades que beneficiam cada sistema resultante". (TAYLOR et al., 2009)

Seguem alguns exemplos de estilos arquitetônicos que podem ser observados em muitas aplicações e sistemas atuais (AKMEL et al., 2017).

- Pipes & Filters: componentes são filtros que transformam em saídas os dados que entram.
- Publish/Subscribe: assinantes se registram para receber mensagens ou conteúdos específicos; na outra ponta, publicadores produzem o conteúdo ou mensagem.
- Sensor-Controlador-Atuador: usado para estruturar aplicações embarcadas de controle.
- Black Board: componentes operam no quadro negro via compartilhamento de memória, armazenamento em banco de dados. O controle do sistema é totalmente orientado pelo estado do quadro negro.
- Orientado a Serviços: funcionalidades de negócio são agrupadas e encapsuladas em unidades reutilizáveis denominadas serviços.

3.1.2 Elementos de Arquiteturas de Software

Arquitetura de software compreende a composição de diferentes elementos. Elementos estes que endereçam aspectos chave do sistema (TAYLOR et al., 2009).

- Processamento, também referenciado como funcionalidade ou comportamento;
- Estado, também referenciado como informação ou dados;
- Interação, também referenciado como interconexão, comunicação, coordenação ou mediação.

3.1.2.1 Componentes e Interfaces

Componentes de software são construções que viabilizam os princípios de engenharia de software de encapsulamento, abstração e modularidade. Por sua vez, esta tem uma série de implicações positivas sobre um comportamento tal como modularidade, reutilização e evolução (GARLAN et al., 2000) (MAGEE; KRAMER, 1996) (TAYLOR et al., 2009).

São os elementos que encapsulam o processamento e os dados (estado) de uma arquitetura de sistemas. Um componente pode ser simples como uma operação ou tão complexo como um sistema, dependendo da arquitetura, da perspectiva adotada pelos projetistas e das necessidades do sistema. O aspecto chave de qualquer componente é que ele pode ser reconhecido por seus usuários, sejam eles humanos ou software.

Para Garlan et al. (2000), os componentes representam os principais elementos computacionais e de armazenamentos de dados de um sistema. Intuitivamente, eles correspondem às caixas em descrições caixa-e-linha de arquiteturas de software. Exemplos típicos de componentes incluem clientes, servidores, filtros, objetos, *blackboards* e bancos de dados. Na maioria das ADL (Linguagens de Descrição de Arquiteturas), os componentes podem ter múltiplas interfaces, cada interface definindo um ponto de interação entre um componente e seu ambiente.

Todo componente expõe ao menos uma interface. Cada interface exposta pelo componente representa um ponto de interação entre o componente e seu ambiente externo.

3.1.2.2 Conectores

Conectores de software são a abstração arquitetônica que gerencia as interações entre componentes (GARLAN et al., 2000) (MAGEE; KRAMER, 1996) (TAYLOR et al., 2009).

Conector é um elemento da arquitetura que torna efetiva e regula as interações

entre os componentes de software. Componentes são responsáveis pelos dados ou processamento, ou ambos simultaneamente. Outro aspecto fundamental dos sistemas são as interações entre os blocos de construção dos sistemas. Muitos sistemas modernos são construídos a partir de um grande número de componentes complexos, distribuídos em múltiplos hosts (em alguns casos móveis) e atualizados dinâmica e periodicamente. Em alguns sistemas, garantir as interações entre os componentes pode ser mais importante e desafiador para desenvolvedores do que a funcionalidade dos componentes individuais.

3.1.2.3 Configuração (Topologia)

Componentes e conectores são compostos de um modo específico em uma arquitetura de software para cumprir os objetivos do sistema (GARLAN et al., 2000) (MAGEE; KRAMER, 1996) (TAYLOR et al., 2009). A composição representa a configuração do sistema, também referenciada como topologia. Uma configuração arquitetural é um conjunto de associações específicas entre componentes e conectores de um sistema de software, segundo Taylor et al. (2009).

3.1.2.4 Táticas

"Uma tática é uma decisão de projeto que controla a resposta de um atributo de qualidade a partir de um estímulo". (BASS et al., 2003)

Táticas estão diretamente ligadas a atributos de qualidade de serviço. Shade et al. (2012) e Choi et al. (2007) descrevem qualidade de serviço (QoS) como uma combinação de várias qualidades ou propriedades de um serviço, tais como: Disponibilidade, Segurança, Tempo de Resposta e Taxa de Transferência. Ainda, as métricas de QoS definidas auxiliam provedores na avaliação dos serviços prestados.

O conceito descrito acima é ilustrado na figura 3.1.

Cada tática constitui uma opção de projeto para o arquiteto, adotada para uma dada categoria de atributo de qualidade de serviço, conforme observado em Bass et al. (2003) e Sabry (2015). Por exemplo, para o atributo disponibilidade, uma tática associada à categoria recuperação de falha poderia ser o uso de redundância para aumentar a disponibilidade de um sistema.

Para o aumento de disponibilidade, redundância não é a única tática disponível. Pode-se perceber a necessidade da criação de cópias de segurança, que podem ser utilizadas em caso de falha do original. Esta situação exige sincronização para o caso

Figura 3.1 - As táticas controlam respostas de um atributo de qualidade a partir de estímulos.

Fonte: Bass et al. (2003)

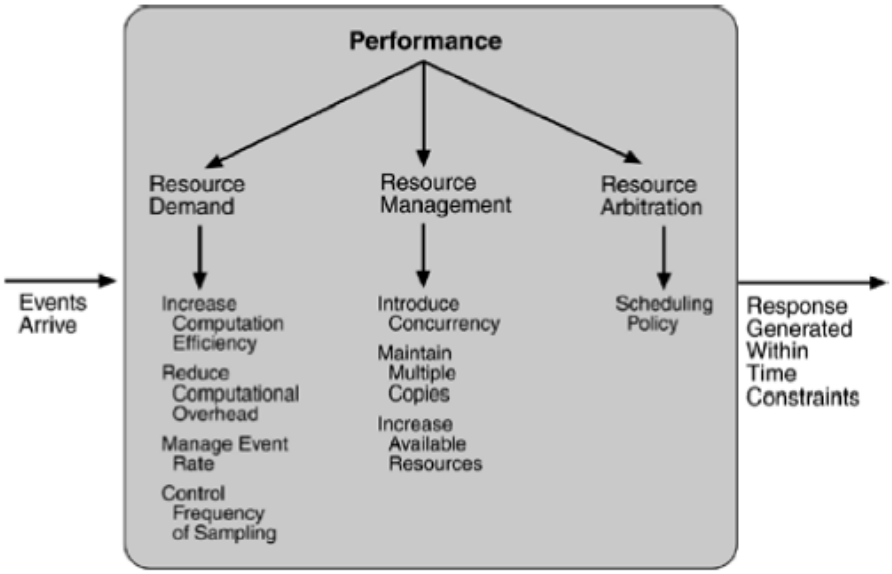
da redundância de dados.

Táticas podem refinar outras táticas. Redundância pode ser identificado como uma tática, e, como tal, pode ser refinada em redundância de dados ou redundância computacional, ambas são consideradas táticas.

Exemplificando, um padrão para disponibilidade provavelmente faria uso de uma tática de redundância e uma tática de sincronização. Esse tipo de organização é denominado padrões de táticas (*Patterns package tactics*).

Podem ocorrer mais refinamentos para tornar cada tipo de tática mais concreta. Para cada atributo de qualidade, segundo Bass et al. (2003), táticas específicas podem ser adotadas. Táticas são organizadas de forma hierárquica, conforme exemplo apresentado na figura 3.2, que exemplifica como podem ser organizadas táticas para o atributo de qualidade desempenho.

Figura 3.2 - Sumário de Táticas de Desempenho.



Fonte: Bass et al. (2003)

3.2 Software Auto-Adaptável

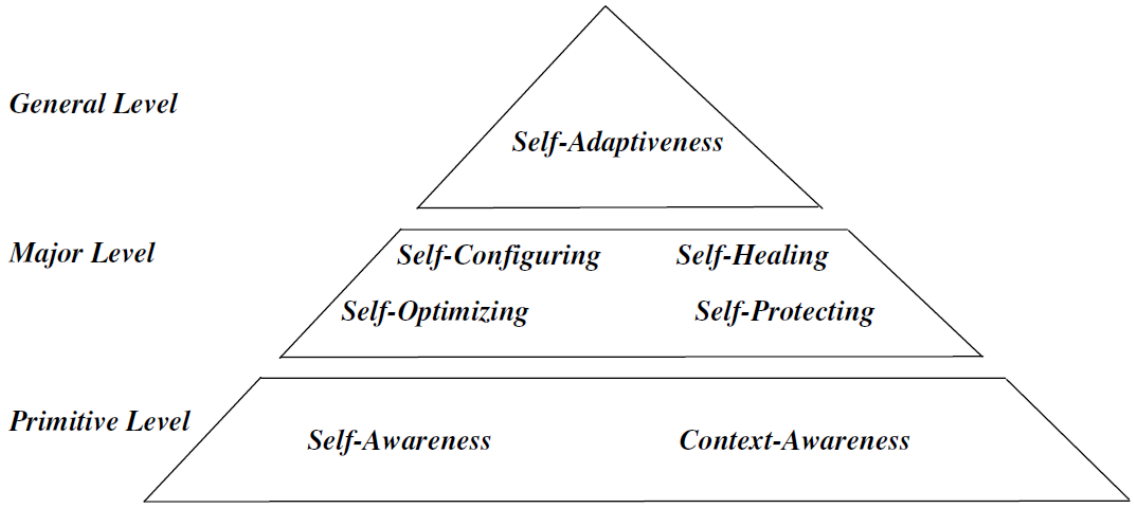
Segundo Oreizy et al. (1999), Software Auto-Adaptável modifica o seu comportamento em resposta a mudanças em seu ambiente operacional. Por ambiente operacional, entende-se qualquer elemento observável pelo sistema de software, isto é, entradas do usuário, dispositivos de hardware externos e sensores ou programas de instrumentação.

Taylor et al. (2009) define adaptaptação como a modificação de um sistema de software para satisfazer as novas necessidades e circunstâncias mutáveis, e define adaptabilidade como a habilidade de um sistema de software de satisfazer novos requisitos e ajustar-se para novas condições operacionais durante seu tempo de operação.

O conceito de auto-adaptabilidade definido por Salehie e Tahvildari (2009) visa ajustar vários artefatos e atributos em resposta a mudanças em si próprio (self) e no contexto do sistema de software. Por si próprio, significa todo o corpo do software, geralmente implementado em múltiplas camadas. O contexto compreende todo o ambiente operacional que afeta o sistema de software e seu comportamento. Software auto-adaptável é um sistema de circuito fechado, retroalimentado pelo seu contexto e por si próprio. É apresentado o conceito das auto-propriedades dos sistemas de

software para obtenção da auto-adaptatividade do sistema. As auto-propriedades são organizadas em um estrutura hierárquica, conforme apresentado na figura 3.3.

Figura 3.3 - Hierarquia das auto-propriedades de um sistema auto-adaptável.



Fonte: Salehie e Tahvildari (2009)

A hierarquia das auto-propriedades é organizada em três níveis. Nesta hierarquia, auto-adaptabilidade e auto-organização são propriedades gerais que são decompostas em propriedades principais e primitivas em dois níveis diferentes. Apresenta-se abaixo uma descrição de cada camada e propriedade:

- **Nível Geral (General Level)**: Este nível contém propriedades globais do software auto-adaptativo. O subconjunto dessas propriedades, abaixo da hierarquia de auto-adaptabilidade (self-adaptiveness), consiste em autogestão, autogoverno, auto-manutenção, autocontrole e auto-avaliação. Outro subconjunto deste nível é a auto-organização.
- **Nível Principal (Major Level)**: A iniciativa de computação autônoma define um conjunto de quatro propriedades para este nível levantadas a partir de mecanismos de auto-adaptação biológica. Por exemplo, o corpo humano tem propriedades semelhantes, de modo a adaptar-se a mudanças no seu contexto como por exemplo, a mudança de temperatura no ambiente ou uma lesão ou falha de um dos órgãos internos. A lista a seguir discorre um pouco mais sobre o detalhes das propriedades:
  - **Autoconfiguração (self-configuring)**: é a capacidade de reconfigu-

ração automática e dinâmica em resposta às mudanças, isto é, instalar, atualizar, integrar e compor/decompor entidades de software.

  - **Autoreparação (self-healing)**: está ligada ao autodiagnóstico das condições do sistema e/ou auto-reparo. É a capacidade de descobrir, diagnosticar e reagir a perturbações. O sistema também pode antecipar potencial probabilidade de ocorrência de problemas e, consequentemente, tomar medidas adequadas para evitar uma falha. Autodiagnóstico refere-se ao diagnóstico de erros, defeitos e falhas, embora a auto-reparação incida sobre a recuperação a partir delas.
  - **Auto-otimização (self-optimizing)**: também é chamado de auto-ajuste ou auto-regulação é a capacidade de gerenciar o desempenho e alocação de recursos para satisfazer as necessidades dos diferentes usuários. Tempo de resposta fim a fim, vazão, utilização e carga são exemplos de questões importantes relacionadas a esta propriedade.
  - **Autoproteção (self-protecting)**: é a capacidade de detectar falhas de segurança e recuperar o sistemas de suas falhas; possui dois aspectos, ou seja, defender o sistema contra ataques maliciosos, bem como antecipar problemas e tomar medidas para evitá-los ou para mitigar seus efeitos.

- **Nível primitivo (Primitive Level)**: Autoconsciência (self-awareness), auto-monitorado, auto-situado, e consciência do contexto (context-awareness) são as propriedades primitivas discutidas a seguir com mais detalhes:

  - **Autoconsciência (self-awareness)**: significa que o sistema tem conhecimento dos seus estados e comportamentos. Esta propriedade é baseada na propriedade auto-monitoramento, que reflete o que é monitorado.
  - **Consciência do contexto (context-awareness)**: significa que o sistema conhece o seu contexto, que é o seu ambiente operacional.

Para além das autopropriedades acima descritas, são apresentadas cinco adicionais definidas por (BERNS; GHOSH, 2009), listadas a seguir:

- **Autoestabilização (Self-stabilization)**: Capacidade que o sistema possui de recuperar uma configuração segura do sistema a partir de uma configuração arbitrária, e permanecer com estas configurações.

- **Auto-Organização (Self-organization)**: Habilidade de organização autônoma em uma estrutura específica. A funcionalidade do sistema é obtida pela colaboração dos membros desta estrutura. Por meio da re-organização automática, o sistema é habilitado a se recuperar de falhas temporárias que causam algum distúrbio à estrutura estabelecida.

- **Auto-Escala (Self-scaling)**: Auto-escala refere-se à capacidade de um sistema de manter um bom desempenho sob cargas (tamanhos) diferentes do sistema. Um sistema auto-escalável pode precisar mudar a topologia da arquitetura para lidar com uma mudança na carga, quando não é possível realizar estas mudanças via modificações internas.

- **Autoimunidade (Self-immunity)**: Agrega um componente de aprendizado ao sistema. Auto-imunidade habilita os processos a acionar mecanismos de tolerância em tempo de execução quando necessário, independentemente da ocorrência de falhas. Como resultado, o sistema opera com menos sobrecarga, quando falhas ou perturbações deixam de ocorrer, e passa para o modo de mascaramento (auto-imune), quando falhas de qualquer tipo particular começam a ocorrer.

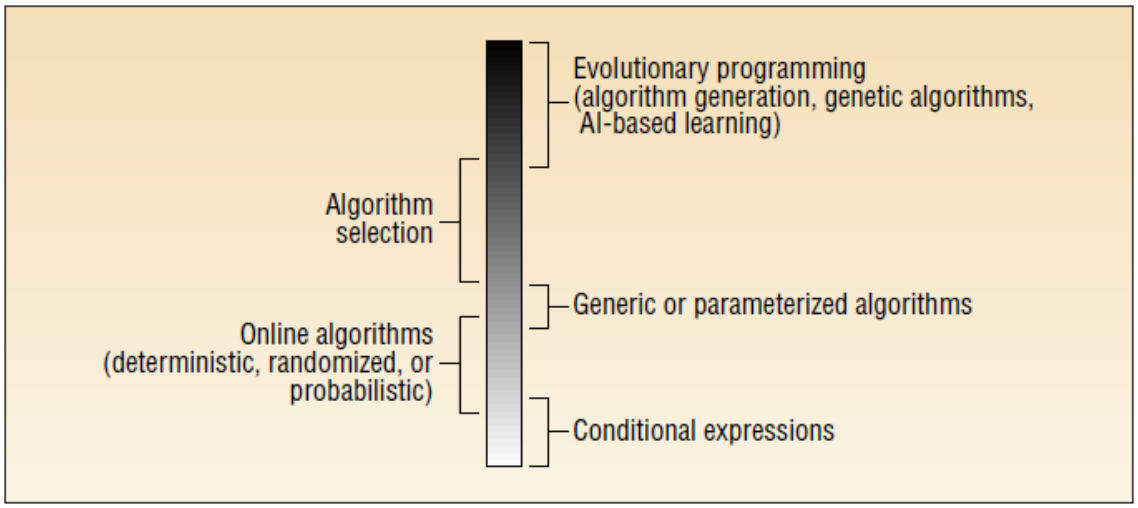
- **Auto-retenção (Self-containment)**: tem como objeto a redução de danos e eventual recuperação via limitação de impactos de ações maliciosas externas.

Segundo Baresi et al. (2006) os tipos de sistemas onde pode-se prever a aplicação de um mecanismo de adaptação são descritos como:

- Sistemas fechados
  - Possuem um conjunto de elementos definidos
  - Adaptação pode atuar apenas nestes elementos para manter o sistema sob supervisão
- Sistemas Abertos
  - Elementos podem aparecer e desaparecer
  - Adaptação deve "descobrir" os elementos existentes e atuar neles para manter o sistema sob supervisão.

Em Oreizy et al. (1999), observa-se que, para atingir os objetivos de adaptabilidade, são necessários alguns processos para monitorar o ambiente, analisar, planejar as mudanças e implantar as modificações em um sistema de software de modo a responder às seguintes questões: Sob quais condições? Adaptação de sistemas abertos ou fechados? Custo efetivo? Qual o tipo de autonomia? Frequência? Tipo e precisão da informação? Neste contexto, a arquitetura de software tem um papel central para a auto adaptabilidade, pois pode ser obtida, por exemplo, pela substituição da versão de determinado componente do sistema. Na figura 3.4, é apresentado um espectro da auto-adaptação, onde elementos mais próximos do nível mais inferior da figura selecionam alternativas pré-determinadas de adaptação. Elementos mais próximos do nível superior apoiam mudanças sem precedentes e fornecem uma separação mais clara dos interesses de adaptação de software.

Figura 3.4 - Espectro de abordagem para auto-adaptação de software.



Fonte: Oreizy et al. (1999)

3.2.1 Abordagens para Adaptação de Software

Esta seção visa apresentar algumas abordagens conceituais de metodologias para implementação de sistemas de software adaptáveis. Na figura 3.5, é apresentada uma metodologia para adaptação, a qual foi sugerida em Oreizy et al. (1999). A metade superior do diagrama, nomeado gerenciamento de adaptação (*adaptation management*) descreve o ciclo de vida dos sistemas de software adaptáveis.

O ciclo de vida pode ter pessoas no loop ou ser inteiramente autônomo. As descrições de cada fase do ciclo de vida do loop de gerenciamento de adaptação pode ser

observada a seguir:

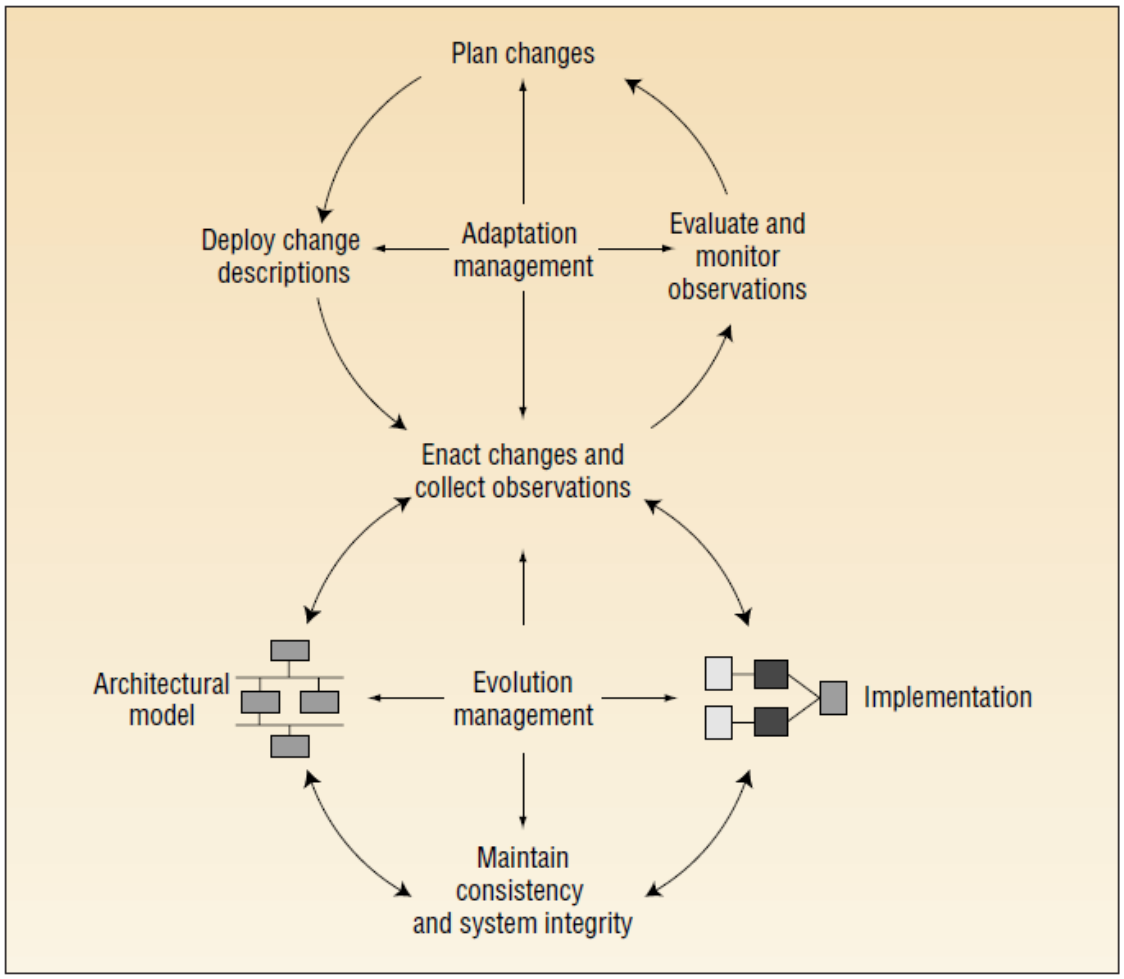
- **"Ativar mudanças e coletar observações (Enact changes and collect observations)"**: coleta de observações, isto é, leitura de métricas do ambiente e repasse destas informações para a fase seguinte; também ativa ou aprova as alterações previstas pela fase de implantação de alterações.
- **"Avaliar e monitorar observações (Evaluate and monitor observations)"** se referem a todas as formas de avaliar e observar a execução de uma aplicação.
- **"Planejar mudanças (Plan Changes)"** refere-se à tarefa de aceitar as avaliações, definir uma adaptação apropriada e construir um plano de execução para implementação da adaptação.
- **"Implantar descrições de mudanças (Deploy change descriptions)"** é a condução coordenada das descrições de mudanças, de componentes e, possivelmente, de novos observadores ou avaliadores para a implementação efetiva da adaptação no sistema.

Por outro lado, a implantação das mudanças pode também extrair dados e possivelmente componentes da aplicação em execução e conduzi-los a algum outro ponto de análise e otimização.

Em Baresi e Guinea (2012), é abordada a dificuldade para prever todas as situações de que o sistema precisa para operar. Devido a esta dificuldade, muitas decisões importantes precisam inevitavelmente adiar a sua execução. Isto implica que os sistemas precisam ser capazes de reagir às mudanças em seu contexto, de modo a continuar assegurando as suas qualidades funcionais e não funcionais. Estes sistemas também devem ser capazes de atender a novos e diferentes *stakeholders*, que podem ter as mais diversas expectativas; o sistema precisa então harmonizar toda essa competição de requisitos. Estes sistemas podem ser organizados em diferentes categorias, dentre as quais destacam-se:

- Sistemas que lidam com o seu contexto de uso e selecionam seus parceiros com referência às necessidades atuais.
- Sistemas que adaptam seu comportamento em relação às novas condições e modificam sua topologia para refletir essas novas necessidades.

Figura 3.5 - Processos de alto nível para uma abordagem abrangente e de propósito geral para sistemas de software adaptáveis.



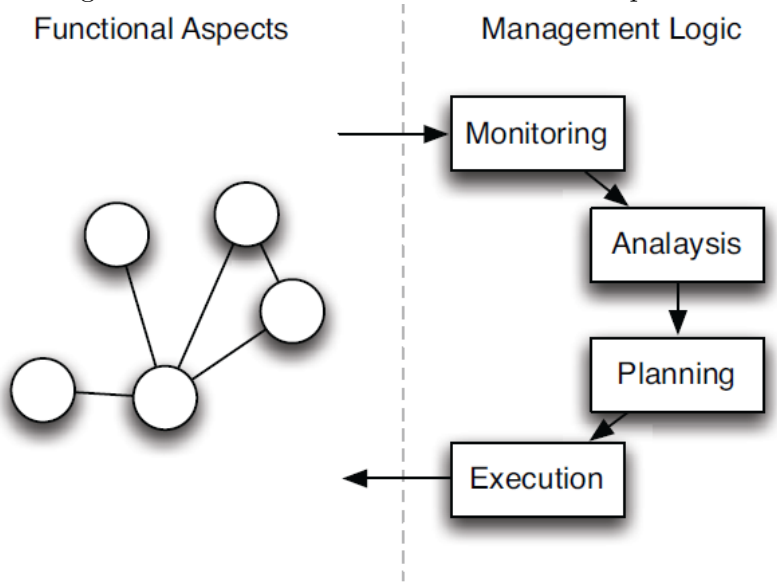
Fonte: Oreizy et al. (1999)

- Sistemas cujas partes mudam com frequência ao longo do tempo, e a coordenação efetiva das partes é a chave para fornecer a funcionalidade esperada.

Estes sistemas integram aspectos funcionais e gerenciamento lógico para decidir quando e como devem sofrer adaptação. Na figura 3.6, é apresentado um exemplo para implementação destas características.

Entre as metodologias para adaptação, no caso de sistemas distribuidos, por exemplo, é descrita a Iniciativa A3, uma solução composta de três elementos (GUINEA; SAEEDI, 2012), que descreve uma abordagem para adaptação de sistemas distribuí-

Figura 3.6 - Visão conceitual de sistemas adaptáveis.



Fonte: Baresi e Guinea (2012)

dos que ataca o seguinte problema: sistemas distribuídos compreendem um número significativo de entidades que devem ser devidamente coordenadas para atingir um objetivo.

Estes requisitos de coordenação geralmente são difíceis de serem refletidos nos projetos de arquitetura dos sistemas. Frequentemente, surgem mudanças no ambiente de execução ou nos recursos disponíveis para o sistema. A adaptação torna-se então necessária para atender a esses requisitos.

No sentido de se obter um framework que combine adaptação centrada em arquitetura e análise baseada em aprendizado, observa-se as características do framework FUSION (ELKHODARY et al., 2010), que apresenta uma forma de adaptação centrada em características (features).

- Adaptação centrada em arquitetura: Derivada da iniciativa de computação autônoma que criou um modelo de referência conhecido como MAPE, que conforme informado anteriormente, organiza um loop contínuo que consiste das seguintes atividades: Monitorar, Analisar, Planejar e Executar. Neste sentido, o resultado final é uma adaptação que é uma solução centrada em arquitetura.

- Adaptação baseado em política: São frameworks de adaptação que empregam lógica e políticas baseadas em métodos de indução. Políticas são especificadas como regras lógicas que podem induzir novas políticas.
- Adaptação baseada em aprendizado: concentra-se em melhorar o comportamento do sistema, aprendendo com prévia experiência e por planos de adaptação gerados dinamicamente em resposta às mudanças ambientais.

3.2.2 Condições para a Adaptação de Sistemas

Conforme observado em Kephart e Chess (2003), é possível categorizar atividades de auto-gerenciamento em quatro funções comuns para a automação de processos de TI, são elas: coletar os detalhes para identificar uma necessidade, analisar os dados para determinar o que deve ser feito para atender à necessidade, criar um plano para atender à necessidade e executar esse plano. Para que um sistema de software se torne auto-gerenciável e, desta maneira, automatizar a tarefas de configuração, cura (healing), otimização e proteção, às seguintes condições devem existir:

a) As tarefas envolvidas na configuração, cura (healing), otimização e proteção do sistema de TI precisam ser automatizadas.

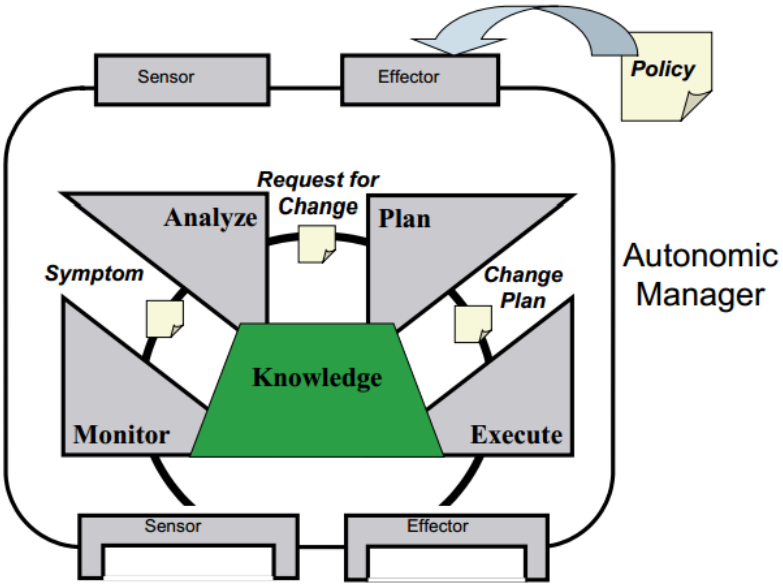
b) Deve ser possível iniciar estes processos com base em situações que podem ser observadas ou detectadas na infraestrutura de TI.

c) O gerenciador autônomo deve possuir conhecimento suficiente para executar a tarefa que é delegada a ser automatizada.

Na figura 3.7, é ilustrado um gerenciador autônomo que é uma implementação que automatiza uma função de gerenciamento. O gerente autônomo é um componente que implementa um loop de controle inteligente responsável por coletar os detalhes do sistema, analisar os detalhes de forma a determinar se existe necessidade de mudança, criar um plano ou sequência de ações e executar estas ações. As situações de adaptação de sistemas podem ser respondidas por meio das seis questões Onde, Quando, O quê, Por quê, Quem e Como (SALEHIE; TAHVILDARI, 2009), descritas abaixo:

- **Onde**: Se refere à localização da necessidade de mudança; quais artefatos, em quais camadas (middleware por exemplo) e qual o nível de granularidade necessário para a mudança. Onde refere-se à localização do problema que precisa ser resolvido por adaptação.

Figura 3.7 - Gerenciador Autônomo



Fonte: Kephart e Chess (2003)

- **Quando**: Relacionado a aspectos temporais que endereçam a mudança. Quando aplicar a mudança e quando é viável a alteração? Pode ser aplicável em qualquer momento solicitado pelo sistema ou existem restrições a alguns tipos de mudança? Qual a frequência que o sistema precisa ser alterado? As mudanças ocorrem continuamente ou apenas quando necessário? É suficiente executar a adaptação de forma reativa ou é necessário prever mudanças e agir proativamente?

- **O quê**: Referente a quais atributos ou artefatos do sistema podem ser alterados por ações de adaptação e o que é necessário alterar em cada situação. Pode variar de parâmetros e métodos a componentes, estilos arquitetônicos e recursos de sistema. Também é importante identificar as alternativas disponíveis para as ações e o intervalo de valores para os atributos (por exemplo, parâmetros). Mais além, é essencial determinar quais eventos e atributos devem ser monitorados para acompanhar as mudanças e quais recursos são essenciais para as ações de adaptação.

- **Por que**: Motivação pela qual se constrói software adaptável. Os objetivos endereçados pelo sistema (por exemplo: robustez).

- **Quem**: Refere-se ao nível de automação e envolvimento humano no software adaptável. Com relação à automação, espera-se o mínimo de inter-

venção humana, ao passo que uma interação eficaz com os proprietários do sistema é necessária para construir diretivas de confiança e de transferência (por exemplo, políticas de negócios).

- **Como**: Um requisito importante para a adaptação é determinar como os artefatos adaptáveis podem ser alterados e quais ações podem ser mais apropriadas de serem aplicadas sob determinadas condições. Isto inclui a ordem das mudanças, custos e efeitos pós tomados para decidir a próxima ação ou plano.

3.2.3 Elementos de um Sistema Auto-Adaptável

Um sistema auto-adaptável (KEPHART; CHESS, 2003) (KRUPITZER et al., 2013) (SALEHIE; TAHVILDARI, 2009) é composto pelos seguintes elementos:

- **Gerenciador Autônomo**: responsável por armazenar informações de contexto e pela execução dos loops de controle MAPE para gerenciamento de um determinado recurso; o gerenciador é acoplado ao recurso gerenciado por meio de sensores e atuadores.
- **Recurso Gerenciado**: recurso de Hardware ou Software que é monitorado e alterado pelo Gerenciador Autônomo;
- **Lógica de Adaptação**: Contém o código necessário para adaptar o recurso gerenciado face às condições verificadas e planejadas pelo Gerenciador Autônomo.

3.2.4 Cenário para orquestração e adaptação de serviços web

The Open Group (2009) descreve que um serviço é uma representação lógica de uma atividade comercial repetida que possui um resultado específico (por exemplo, verifique o crédito do cliente, forneça dados meteorológicos, consolide relatórios de perfuração), é autônomo e pode ser composto por outros serviços. É visto como uma caixa preta para os consumidores do serviço, uma atividade repetitiva que tem um resultado específico, possui um provedor, pode ter um ou mais consumidores e produz efeitos que são de valor para seus consumidores.

Um produto ou serviço de qualidade é aquele que atende perfeitamente, de forma confiável, de forma acessível, de forma segura e no tempo certo, às necessidades do cliente (CAMPOS, 1992).

Serviços, de uma forma geral, estão relacionados à missão comercial da empresa e são conduzidos de forma definida e repetitiva. Os serviços de software de uma Arquitetura SOA existem para suportar os processos de negócios da empresa. Essa relação pode e deve ser simbiótica. A análise dos processos de negócios é a principal maneira de identificar os serviços de software. Por outro lado, a existência dos serviços de software corretos permite que novos processos de negócios sejam desenvolvidos, para atender a novas oportunidades de negócios.

Arquiteturas SOA permitem o desenvolvimento de processos que orquestram acionamento de serviços. O modelo de processo caracteriza as relações e propriedades temporais das ações e eventos associados à interação com o serviço, segundo Advanced Open Standards for the Information Society (OASIS) (2006).

Sistemas baseados em serviços web necessitam, cada vez mais, satisfazer requisitos relacionados a atributos de qualidade, como confiabilidade e eficiência. Para atingir esses objetivos, algumas táticas podem ser aplicadas, como, por exemplo, redundância e replicação de componentes ou mesmo a aplicação de políticas de escalonamento, no caso de recursos limitados de hardware.

No caso de alta disponibilidade, a redundância por meio da replicação de nós (cópias) constitui uma tática que pode ser aplicada. Outra tática é a utilização de serviços similares, e, para tanto, pode ser necessário o desenvolvimento de adaptadores para serviços equivalentes.

Os componentes podem ter diferentes atributos de qualidade de serviço, como tempo de resposta, up-time, disponibilidade. Desta maneira, qual a melhor opção para satisfazer os requisitos de qualidade do usuário?

O monitoramento e a troca da composição do processo podem ser realizados a qualquer momento no caso de o framework detectar uma necessidade de adaptação a partir da observação de dados de qualidade de serviço.

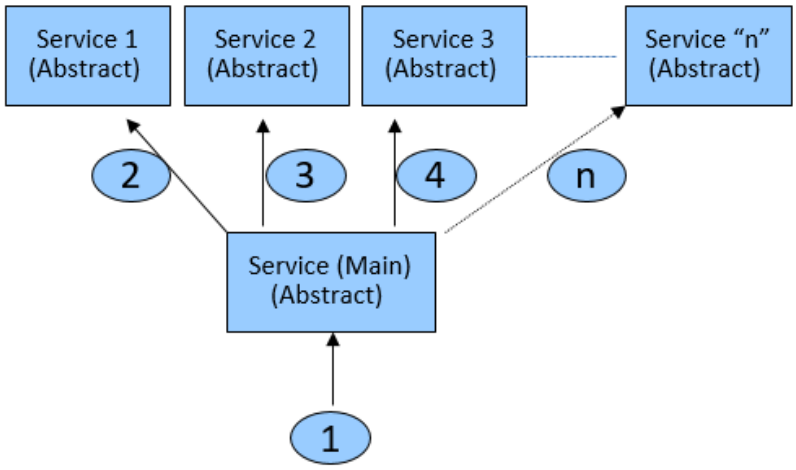
Um orquestrador, por sua vez, é um agente computacional que organiza a composição de serviços web. Acionamento de Serviços web podem ser alterados em tempo de execução com base em algumas estratégias determinadas para certos atributos de qualidade de serviço. Neste caso, a estratégia principal está na troca de componentes por meio da atuação nos conectores dos serviços.

Por exemplo, no caso de serviços web, a adaptação por meio do uso da tática do uso de cópias/serviços similares pode ser obtida:

- No momento da Implantação do processo (Deployment);
- No momento da Invocação do Processo;
- No momento da Invocação de um Serviço (Componente);

Na figura 3.8, é apresentada uma composição (orquestração) de serviço e respectivas ordens de execução. Serviços alternativos podem substituir serviços originais.

Figura 3.8 - Composição de Serviços com ordem de acionamento.

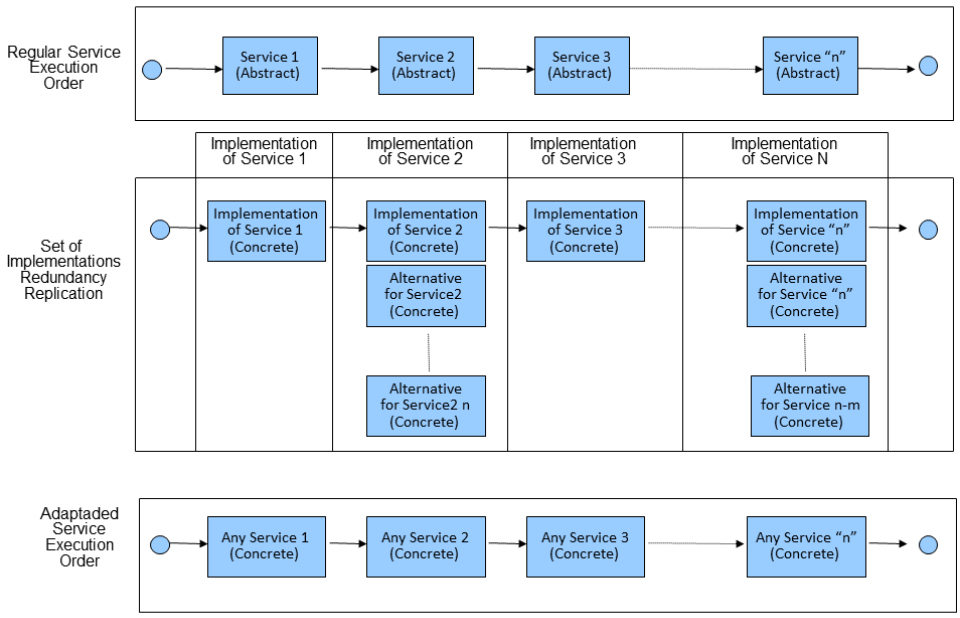


Fonte: Autor

Esses serviços podem ser adaptados através da execução de táticas como a replicação. Um serviço pode ter dois (2) ou três (3) implementações, qualquer uma delas pode ser utilizada em tempo de execução. O orquestrador, no tempo certo (quando), escolhe o serviço que será executado com base em algum critério ou heurística. O conhecimento do contexto é um elemento chave para permitir que o agente otimize a nova composição de forma dinâmica. A heurísticas citada é melhor descrita na figura 3.9.

A solução apresentada neste trabalho é aplicável no contexto do clima espacial, por exemplo, onde um Proxy pode atuar no processo de seleção e adaptação da aplicação cliente para utilização da melhor opção entre as implementações de serviço disponíveis.

Figura 3.9 - Composição de Serviços e Ordem de Execução



Fonte: Autor

3.2.5 Trabalhos Correlatos

Atualmente há muita pesquisa e esforço centrados no desenvolvimento de soluções que possibilitem suporte para software auto-adaptável.

Verificam-se algumas soluções para problemas específicos de adaptação, como auto-reparo ou *frameworks*, mais gerais, com um conceito de adaptação bem definido. Entretanto, devido a uma variedade substancial de auto-propriedades ou características como adaptação "on-the-fly", sempre há limitações que devem ser observadas.

Nesta seção, são apresentadas as abordagens para adaptação disponíveis na literatura, que serviram como fonte para o andamento deste trabalho.

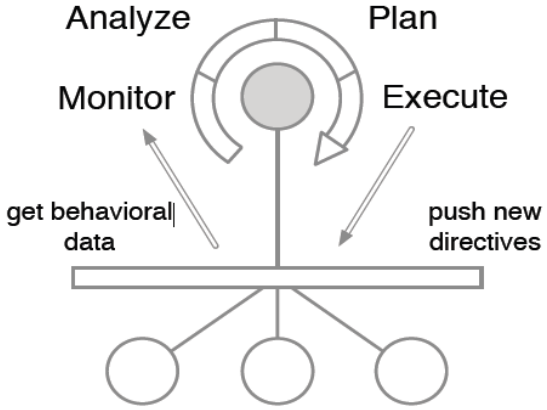
3.2.5.1 Framework A3

O framework A3 (BARESI; GUINEA, 2011) (GUINEA; SAEEDI, 2012) é um framework que promove a adaptação descentralizada de componentes de uma determinada arquitetura.

O framework A3, ilustrado na figura 3.10, é composto de três elementos principais:

- **Grupo**: permite que múltiplos elementos sejam tratados como uma única entidade menos dinâmica;
- **Supervisor**: Responsável por coordenar seus seguidores;
- **Seguidor**: Junta-se a grupos para melhor entender como se comportar dependendo de determinada situação.

Figura 3.10 - Abordagem A3 para adaptação distribuída de sistemas.



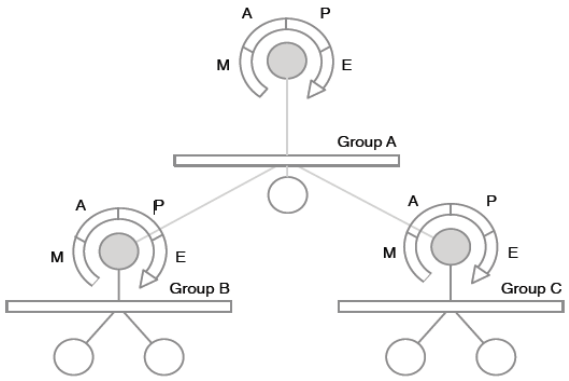
Fonte: Guinea e Saeedi (2012)

A abordagem indica que os grupos controlam os loops MAPE de adaptação. Grupos podem ser compostos de várias maneiras diferentes: os nós podem pertencer a um grupo de cada vez, e podem assumir papéis diferentes em grupos diferentes. Composição de grupos lideram a interação de loops de controle MAPE distribuídos conforme apresentado na figura 3.11.

Como os grupos podem ser replicados em diversos nós, há redução significativa de pontos de falha. O framework também prevê a reorganização dos grupos "on the fly". No caso do surgimento de novos nós na topologia do sistema alvo, grupos podem concentrar nós e ficarem mais robustos ou, caso ocorra o inverso, diminuição da carga por exemplo, grupos podem ser desativados.

A lógica de Adaptação é implementada por meio de técnicas de programação orientada a aspectos (AOP). Os grupos gerenciam tanto o loop de controle MAPE como a execução da lógica de adaptação.

Figura 3.11 - Composição de grupos A3 para sistemas distribuídos adaptáveis.



Fonte: Guinea e Saeedi (2012)

3.2.5.2 Framework FUSION

Sistemas geralmente empregam modelos analíticos especificados em tempo de projeto para avaliar suas características e tomar as decisões adequadas de adaptação em tempo de execução. No entanto, os engenheiros muitas vezes não podem prever as mudanças que ocorrem no ambiente, nem os requisitos ou o perfil operacional do sistema, o que gera análises e decisões de adaptação imprecisas.

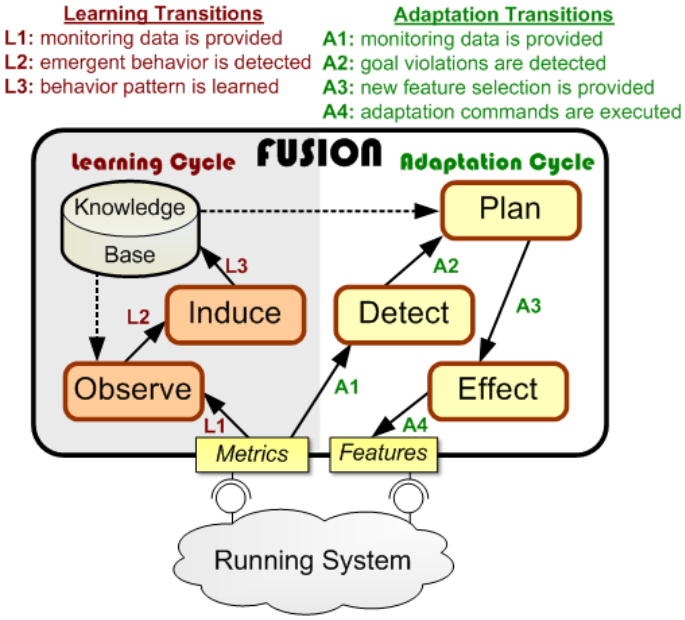
FUSION FeatUre-oriented Self-adaptatION (ELKHODARY et al., 2010) visa resolver o problema, avaliando e aprendendo com o impacto das decisões de adaptação nos objetivos do sistema. O framework possui um componente de aprendizado que permite o ajuste automático da lógica de adaptação a condições imprevistas, também reduz o esforço inicial necessário para a construção de tais sistemas e torna a análise de tempo de execução mais eficiente. Uma visão geral do framework pode ser observado na figura 3.12.

FUSION toma decisões de adaptação usando um ciclo contínuo, chamado ciclo de adaptação. O ciclo de adaptação coleta métricas e otimiza o sistema, executando três atividades na seguinte sequência:

- Analisa as métricas coletadas do sistema em execução para determinar se ocorreu uma violação de objetivo.
- Quando um objetivo é violado, o componente “Plan” procura uma configuração ótima (seleção de recurso).
- Dada a seleção de recursos, o framework determina um conjunto de etapas

de adaptação para aplicação no sistema-alvo.

Figura 3.12 - Visão Geral FUSION.



Fonte: Elkhodary et al. (2010)

O framework também possui um ciclo de aprendizagem para aprender o impacto das decisões de adaptação em termos de seleção de recursos nos objetivos do sistema. O sistema é simulado ou executado em modo off-line e as métricas correspondentes de cada seleção de recurso são coletadas, os dados são utilizados para treinar o framework; desta maneira, este pode induzir um modelo preliminar do comportamento do sistema.

3.2.5.3 Framework e IDE FESAS

FESAS (KRUPITZER et al., 2013) (KRUPITZER et al., 2016) é um framework para a engenharia de sistema auto-adaptáveis orientado a modelos. O framework FESAS promove a reutilização da Lógica de Adaptação e é composto dos seguintes elementos:

- **FESAS Framework**: Conjunto de componentes para o desenvolvimento e a reutilização das lógicas de adaptação e execução dos loops de controle MAPE, formando um gerenciador autônomo customizado.

- **FESAS OOMW**: Middleware FESAS, este componente reside nos nós do sistema alvo e é responsável pela implantação e execução do gerenciador autônomo, também contém um repositório que armazena a meta-informação do ambiente, componentes, sensores, atuadores e lógicas de adaptação.
- **FESAS processo de desenvolvimento**: a abordagem FESAS entende que há a necessidade de um processo de desenvolvimento para sistemas auto-adaptáveis. Nesse sentido, o artigo propõe um processo com dois papéis principais, o Desenvolvedor de Sistemas, que escreve código, e o Projetista de Sistema Auto-Adaptável, que define a configuração necessária para os nós do sistema-alvo, onde os recursos gerenciados se conectam aos sensores e atuadores e aos nós que receberão os gerenciadores autônomos.
- **FESAS IDE**: O ambiente de desenvolvimento FESAS suporta o processo de desenvolvimento e possui duas ferramentas: *FESAS Development Tool* e *FESAS Design Tool*, descritas a seguir:
  - FESAS Development Tool: utilizado para o desenvolvimento do código da lógica funcional, que será armazenado no repositório.
  - FESAS Design Tool: utilizado para configuração dos componentes do sistema-alvo para lógicas de adaptação específicas; essas funções são carregadas em tempo de execução.

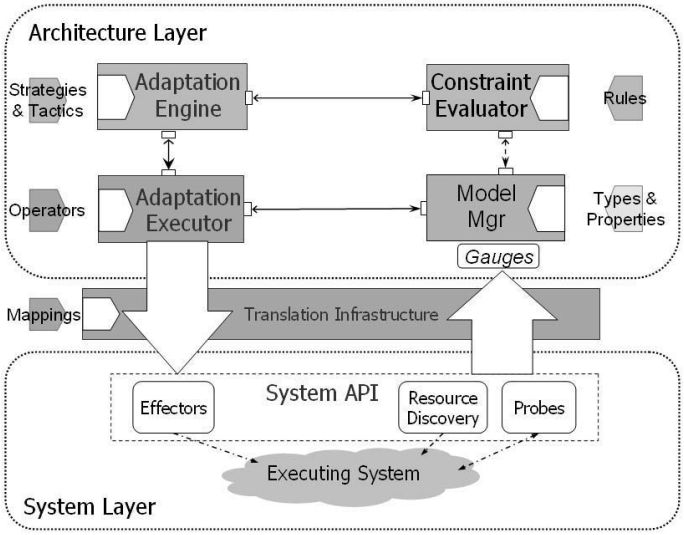
3.2.5.4 Framework Rainbow

O framework Rainbow (GARLAN et al., 2004) monitora as propriedades de um sistema em tempo de execução, avalia violações de restrições definidas em um modelo abstrato e dispara um motor de adaptação no caso de ocorrer alguma violação. O framework é composto de três camadas, que podem ser observadas na figura 3.13:

- **Camada de Arquitetura**: contém medidores que agregam informações provenientes do sistema-alvo e atualiza as propriedades no modelo definido para este; além de um gerenciador de modelos, que define o modelo arquitetônico do sistema; um avaliador de restrições, que verifica o modelo periodicamente e, no caso de ocorrer alguma violação, dispara um motor de adaptação.
- **Infraestrutura de Tradução**: Mantém um mapeamento para tradução de informação entre as camadas de sistema e arquitetura.

- **Camada de Sistema**: Contém componentes para observação e medição, descoberta de recursos e atuadores que interagem com o sistema-alvo.

Figura 3.13 - Camadas Framework Rainbow.



Fonte: Garlan et al. (2004)

3.2.5.5 Abordagem FLAGS

FLAGS *Fuzzy Live Adaptive Goals for Self-adaptive systems* (BARESI et al., 2010), prevê que o desenvolvimento de sistemas auto-adaptáveis deve considerar a adaptação como parte dos requisitos especiais que devem ser considerados no desenvolvimento e, para tanto, há questões como incerteza no levantamento de requisitos e em como endereçar a tratativa da adaptação no processo de desenvolvimento.

A abordagem prevê que há um certo grau de incerteza relacionado à elicitação de requisitos para adaptação; para solucionar este problema, a abordagem FLAGS adota a lógica Fuzzy para refinar os requisitos, diminuindo assim o grau de incerteza e adotando um conceito denominado "objetivo de adaptação", que é traduzido em contra-medidas que devem ser executadas pelo sistema auto-adaptável no caso de uma violação do objetivo.

Os passos do modelo definem que um sistema deve aplicar uma contra-medida assim que violado o objetivo de adaptação. Cada contra-medida está associada a um evento que dispara a sua execução, uma condição para ativação (a máquina de lavar roupas desliga de repente), um objetivo que deve ser alcançado (ligar a máquina de lavar)

e uma sequência de ações que devem ser executadas para alcançar o objetivo. Este itens formam o modelo FLAGS para adaptação de sistemas.

3.2.5.6 Abordagem POISED

A incerteza subjacente às decisões de adaptação têm sido um dos principais obstáculos para a ampla adoção de técnicas de auto-adaptação. Para lidar com a questão da incerteza, a abordagem POISED POssIbilistic SElf-aDaptation (ESFAHANI et al., 2011) baseia-se na teoria da possibilidade para avaliar as consequências positivas e negativas da incerteza.

A abordagem toma decisões de adaptação que resultam na melhor faixa potencial de comportamento. É demonstrada no trabalho a aplicação da abordagem em um problema de melhoria da qualidade de serviço de um sistema de software via reconfiguração de componentes.

3.2.5.7 Framework RESIST

RESIST RESISTing Reliability Degradation through Proactive Reconfiguration (COORAY et al., 2010) é um framework para auto-configuração de software, destinado a satisfazer requisitos de confiabilidade, que leva em consideração outros atributos de qualidade (por exemplo, eficiência).

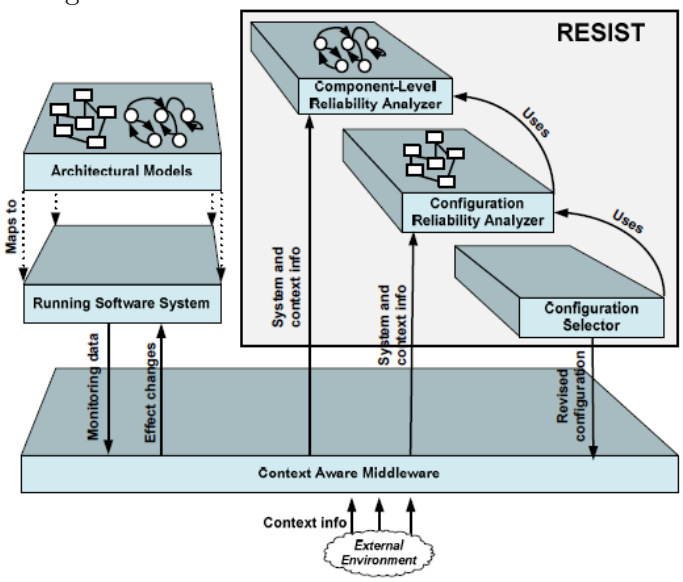
O RESIST executa reconfiguração proativa do software, cujas três principais contribuições são: (1) incorporar múltiplas fontes de informação, em particular informações contextuais, para fornecer previsões de confiabilidade refinadas em tempo de execução; (2) encontrar automaticamente uma configuração arquitetural ótima de modo a alcançar um balanço adequado entre confiabilidade e outros atributos de qualidade; e (3) adaptar proativamente o sistema, posicionando-o na configuração ideal, antes que ocorra a degradação da confiabilidade do sistema. Uma visão geral do framework RESIT é apresentada na figura 3.14.

3.2.5.8 Abordagem ADC

A abordagem ADC - *Anticipatory Dynamic Configuration* - (POLADIAN et al., 2007) ataca a questão da tomada de decisão sobre alocação de recursos escassos a múltiplas aplicações concorrentes, de modo satisfazer um conjunto de parâmetros de qualidade.

Para tanto, a abordagem prega a configuração dinâmica de serviços, onde é apresentada a seleção de um conjunto apropriado de serviços para realizar uma tarefa

Figura 3.14 - Visão Geral Framework RESIST.



Fonte: Cooray et al. (2010)

de usuário e alocar recursos para esses serviços em tempo de execução.

A abordagem apresenta um modelo de previsão de disponibilidade de recursos ao longo do tempo, baseada na teoria da probabilidade, que permite a seleção antecipada de configurações que melhor satisfazem os parâmetros de qualidade do usuário. Ao considerar a previsão de disponibilidade de recursos, o modelo antecipado de configuração escolhe uma configuração que maximiza o valor cumulativo esperado de utilidade ao longo do tempo, o que reduz o número de possíveis reconfigurações futuras.

O modelo apresentado pode inclusive ser incorporado a outras abordagens para sistemas auto-adaptáveis, os quais podem se beneficiar de configurações antecipadas.

3.2.5.9 Framework Dynamic High-level in Self-Adaptive Systems

Rossi et al. (2017) apresentam uma abordagem para auto-configuração de topologia de arquitetura de software baseada em modelos de requisitos. Para materializar a execução da adaptação, são empregados modelos semânticos que representam os modelos de requisitos. Nesse sentido, é introduzido o modelo MAPE-SK do framework, onde a letra “S” significa Semântico.

A arquitetura do framework é baseada no modelo MAPE-K para implementação do

motor de adaptação, e inclui um modelo semântico de requisitos executável que orienta as atividades de auto-adaptação compatível com a abordagem Model@run.time (MORIN et al., 2009).

No trabalho, é apresentado um estudo de caso que inclui um modelo de requisitos semântico para gerenciamento de recursos de uma aplicação do centro responsável pela infraestrutura de TI da Universidade de Bolonha. No estudo de caso, o motor de adaptação atua para modificar a configuração dos recursos, de modo a atingir determinados objetivos de adaptação para gerenciamento autônomo dos recursos; entre as ações estão a escala horizontal de máquinas virtuais e a adição de recursos.

O framework apresentado permite adaptação on-the-fly dada a possibilidade da alteração do modelo semântico desenvolvido para o sistema auto-adaptável, onde é possível a inclusão de novos objetivos de adaptação.

3.2.5.10 Abordagem Self-Adaptive Architecture for Network Inspection in 5G

Inspeção de tráfego em redes móveis de quinta geração (5G) para detecção de ameaças como cavalos de tróia, worms, vírus, entre outros, tem sido um problema quando as taxas de transferência de dados são superiores a 1 Gbps. De forma mais precisa, nos últimos anos, as taxas têm aumentado de um volume médio de 100 Mbps para as atuais taxas superiores a 10 Gbps (MAIMO et al., 2018).

Sistemas de Detecção de Intrusão (IDS) não são eficientes em analisar o tráfego de rede nessas taxas. Uma solução emprega o uso de aprendizado de máquina para situações supervisionadas, aquelas onde as ameaças são conhecidas, e também para situações não supervisionadas, onde uma anomalia observada no tráfego de rede pode representar uma ameaça à segurança da rede móvel.

São empregadas técnicas diferentes de aprendizado de máquina para análise de tráfego de ambas as situações, supervisionado e não supervisionado, ambas baseadas na utilização de redes neurais artificiais. A abordagem adiciona recursos computacionais para análise de tráfego de dados conforme demanda.

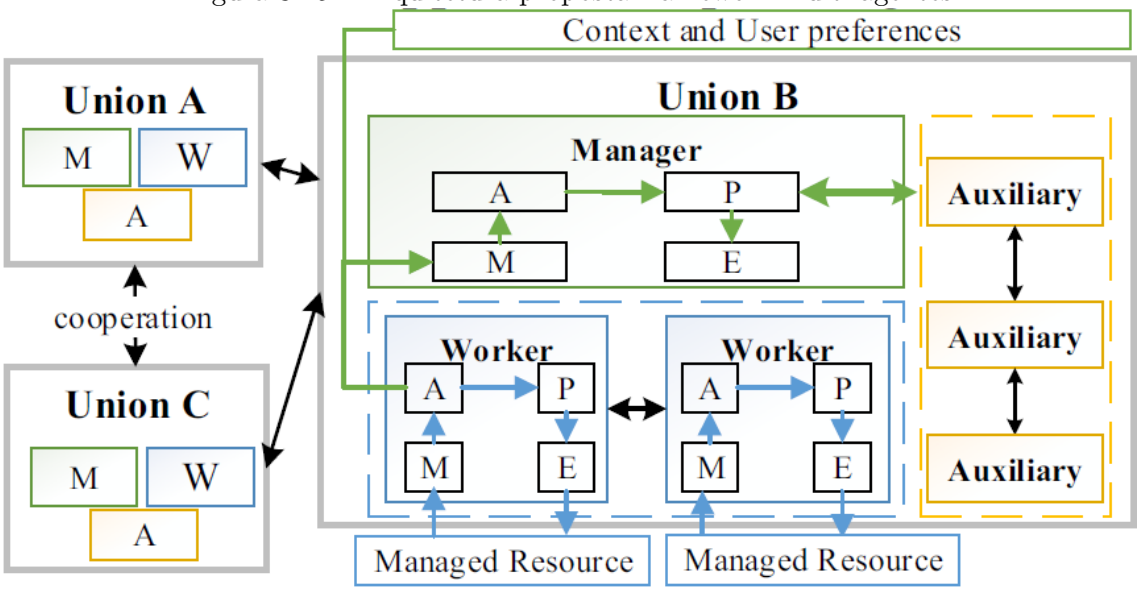
A abordagem apresentada pelos autores é um exemplo de autoimunidade onde são ativados mecanismos de redundância para proteção da rede de dados móveis contra ameaças externas.

3.2.5.11 Framework Multiagent-based Framework with Search-based Optimization

Wang e Li (2016) apresentam um *framework* para desenvolvimento de sistemas auto-adaptáveis multi-objetivo baseado no modelo MAPE. O *framework* propõe a utilização de algoritmos evolutivos de busca para encontrar soluções ótimas para problemas de auto-otimização combinados com tecnologia de agentes.

O *framework* possui dois ciclos MAPE especializados. A arquitetura proposta do *framework* multi-agentes é verificada na figura 3.15, e uma descrição dos dois ciclos é fornecida nos parágrafos posteriores.

Figura 3.15 - Arquitetura proposta framework multi-agentes



Fonte: Wang e Li (2016)

O primeiro ciclo é baseado em regras e trata recursos gerenciados diretamente, como, por exemplo, a verificação de uma avaria em determinado recurso. Mudanças dessa natureza podem disparar execuções de alterações previamente estabelecidas nas regras do gerenciador MAPE. Os elementos do framework que são responsáveis pelo ciclo MAPE dos recursos gerenciados são denominados “Workers”.

O segundo ciclo trata as alterações mais complexas justificadas por alterações de ambiente e influência dos elementos “Worker”, responsáveis pelo primeiro ciclo. O segundo ciclo MAPE faz uso dos algoritmos de busca para encontrar soluções mais

adequadas a estas alterações. O elemento do framework responsável pelo segundo ciclo é denominado “Manager” e recebe informações dos “Workers” e de outro elemento denominado “Auxiliary”, responsável por monitorar alterações no contexto do sistema alvo.

3.2.6 Lacunas observadas nos Trabalhos Correlatos

A primeira lacuna está na observação de que a maioria das abordagens é aplicada a um problema específico. Cita-se como exemplo o framework RESIST (COORAY et al., 2010), que é especializado em adaptação relacionada a confiabilidade. Esta característica impossibilita o suporte a todas as auto-propriedades.

A segunda lacuna é a impossibilidade da exploração de diferentes abordagens para adaptação. Segundo Oreizy et al. (1999), há um espectro de possibilidades para adaptação, desde a adoção de condicionais, algoritmos especializados, até a utilização de técnicas de inteligência artificial, como algoritmos evolutivos e redes neurais. Os trabalhos correlatos pesquisados não permitem a exploração de possibilidades definidas em Oreizy et al. (1999) e apresentadas na figura 3.4.

A terceira lacuna é o suporte à adaptação “*on-the-fly*”, que é a capacidade de definir critérios de adaptação durante a operação do sistema, em tempo de execução, por meio de parametrização. Esta característica não é suportada por sete dos onze trabalhos analisados.

A quarta lacuna observada é que, de uma maneira geral, há a necessidade de conhecimento em auto-adaptação para o desenvolvimento de sistemas auto-adaptáveis. Esta necessidade implica em formação adicional de especialistas e aumento na complexidade do desenvolvimento.

A quinta e última lacuna observada é o acoplamento entre o gerenciador autônomo e a lógica de adaptação. O gerenciamento do ciclo de adaptação nos trabalhos pesquisados, de uma forma geral, está acoplado à lógica de adaptação. Este acoplamento dificulta a adaptação “*on-the-fly*” e pode requerer grande capacidade de processamento nos nós do sistema solo. Cita-se como exemplo a base de conhecimento do *framework* FESAS (KRUPITZER et al., 2013), que contém a meta-informação dos elementos de adaptação e o código responsável pela lógica de adaptação embarcado no middleware FESAS.

3.3 Discussão sobre Sistemas Auto-Adaptáveis na Área Espacial

Conforme apresentado no capítulo 2, os sistemas espaciais têm ganho complexidade e requisitos relacionados à operação autônoma e poderão, em um futuro próximo, ser considerados no seu desenvolvimento.

Foram apresentados, neste capítulo, os conceitos e abordagens relacionados a Sistemas de Software Auto-Adaptáveis. Sistemas auto-adaptáveis se apresentam como uma solução para lidar com esta nova classe de requisitos. Claramente observa-se que sistemas espaciais de segmento solo tradicionais não estão preparados para estes novos cenários, que incluem constelações, mega-constelações, diversidade de instrumentos e recursos que podem ser disponibilizados não apenas pelo INPE, como também por outras agências espaciais ao redor do globo.

No próximo capítulo, é apresentado um conjunto de tecnologias que podem ser empregadas para o desenvolvimento de sistemas auto-adaptáveis e que nortearam o desenvolvimento do framework objeto deste trabalho de pesquisa.

4 TECNOLOGIAS PARA ADAPTAÇÃO

Neste capítulo, é apresentado um panorama geral sobre conceitos e tecnologias que podem ser empregados em uma solução para sistemas auto adaptáveis, aplicáveis ao contexto espacial.

4.1 Qualidade de Serviço (QoS)

Segundo Sommerville (2011), a qualidade subjetiva de um sistema de software, baseia-se, em grande parte, em suas características não funcionais. Estas características não funcionais, conhecidas como requisitos não funcionais, dependem fortemente da arquitetura projetada para o sistema.

De acordo com a definição da Associação Brasileira de Normas Técnicas (ABNT) (2008) sobre qualidade, esta pode ser definida como o grau no qual um conjunto de características inerentes satisfaz a requisitos. Logo, a qualidade de um sistema e de seus componentes pode ser mensurada, quando estes respondem aos requisitos de seus clientes de forma satisfatória.

Para Campos (1992), um produto ou serviço de qualidade é aquele que atende perfeitamente, de forma confiável, de forma acessível, de forma segura, e no tempo certo, às necessidades do cliente.

No contexto dos sistemas auto adaptáveis, as alterações no sistema alvo são justificadas por resultados de métricas coletadas dos sistemas. Nesse sentido, medições da qualidade do sistema são essenciais para a fase de monitoramento prevista no modelo MAPE-K e nas outras abordagens informadas no capítulo 3.

Para cada tipo de arquitetura, existe um conjunto de atributos e métricas de qualidade que podem ser escolhidas e avaliadas com o propósito de mensurar a qualidade de serviço. Shade et al. (2012) e Choi et al. (2007) descrevem qualidade de serviço (QoS) como uma combinação de várias qualidades ou propriedades de um serviço, tais como: Disponibilidade, Segurança, Tempo de Resposta e Taxa de Transferência. Ainda, as métricas de QoS definidas auxiliam provedores na avaliação dos serviços prestados.

Thio e Karunasekera (2005) apresentam a necessidade da realização de medições automáticas de QoS dos sistemas. Ainda segundo o autor, a informação de QoS deve ser mantida e atualizada por componentes do sistema independentes e especializados para tal tarefa. Nesse contexto, medidas de QoS permitiriam que as aplicações cli-

entes usassem e escolhessem componentes de serviço adequados de forma dinâmica e em tempo de execução, produzindo, assim, melhores resultados de eficácia e de eficiência.

É possível, então, estabelecer uma relação entre medições de qualidade de serviço (QoS) e as táticas definidas na seção 3.1.2.4. Para ilustrar esta relação, pode-se aplicar, por exemplo, para as métricas de qualidade de serviço (QoS) para o atributo confiabilidade, Taxa de Confiabilidade de Resposta *(Reliable Response Ratio - RRR)*, Taxa de Falha do Serviço *(Service Failure Ratio - SFR)* e Tempo Médio entre Falha do Serviço *(Mean Time Between Service Failure (MTBF(SRV))* definidas em Choi et al. (2007), as seguintes táticas definidas em Bass et al. (2003) e Sabry (2015): economia de recursos (Spare), votação, redundância ativa e passiva, replicação, redundância funcional e ressincronização de estado.

4.1.1 Acordo de Nível de Serviço

Como verificado nos parágrafos supracitados, a qualidade de um serviço pode ser medida. A coleta e o cálculo de métricas do sistema permitem que a qualidade seja então avaliada. Nesse sentido, de maneira a estabelecer critérios para avaliação da qualidade do sistema, são necessários acordos que determinam qual o nível de qualidade desejada para o sistema e/ou serviço.

De acordo com o *framework* ITIL (2007), um acordo de nível de serviço *(Service Level Agreement - SLA)* descreve o serviço, define metas de nível de serviço, e especifica as responsabilidades entre o produtor e o consumidor do serviço. Adicionalmente, um único SLA pode cobrir múltiplos serviços ou múltiplos consumidores.

Um SLA pode ser criado e aplicado para qualquer tipo de sistema/serviço, seja ele um portal web, uma aplicação standalone, serviços web, entre outros. Deve descrever, de forma clara e adequada, às necessidades particulares de cada parte do sistema.

A produção de métricas para o gerenciamento da qualidade permite a avaliação da eficiência do sistema/serviço em relação a diversos aspectos, tais como: utilização dos recursos de hardware, avaliação da capacidade, avaliação de taxas de erros. O gerenciamento de QoS pode alertar quando algum requisito não está sendo atendido.

De maneira geral, os acordos são descritos de forma textual pelas partes interessadas *(stakeholders)* do projeto, para que possam ser lidos por humanos. Nele, podem conter tabelas, gráficos e todo tipo de informação necessária para o cumprimento do acordo. Porém, no caso dos sistemas auto adaptáveis, surge a necessidade de

garantir a qualidade de serviço em sistemas autônomos, onde não há interação direta de um usuário, neste sentido, os serviços e clientes (consumidores) são máquinas ou componentes de software. Verifica-se, então, a necessidade de representação do acordo SLA em uma linguagem estruturada.

Em European Commission (2013), é possível ver um metamodelo de um ciclo de vida de um SLA para computação em nuvem que mostra as principais fases, estruturas, processos e as interações entre as entidades durante o ciclo de vida do SLA. Ainda, recomenda-se que as descrições sejam suficientemente claras na definição das métricas e adição de atributos de QoS. Dessa forma, é possível representá-lo através de uma linguagem estruturada com XML (eXtensible Markup Language) ou JSON (JavaScript Object Notation), o que permite a leitura do SLA por uma máquina Machine Readable SLAs. Esta representação estruturada de um SLA, permite estabelecer negociações automatizadas.

Outro exemplo de um SLA descrito em linguagem estruturada pode ser observado em Bakalos et al. (2015). No referido trabalho, é apresentado um modelo de especificação criado a fim de permitir negociações automatizadas. A descrição do SLA é especificada em formato XML schema, o que permite a leitura e interoperabilidade de acordos SLA entre aplicações.

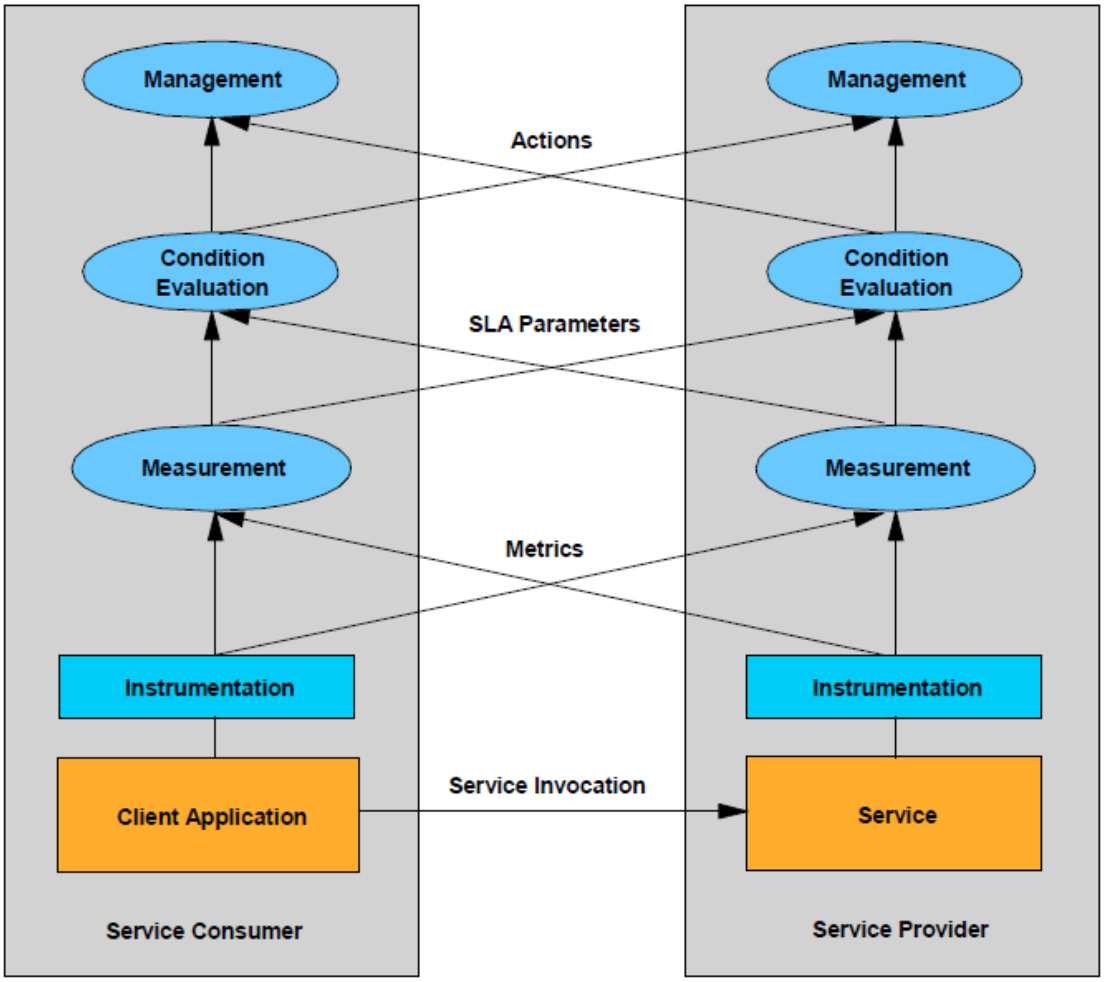
No caso dos serviços web, é apresentado, no trabalho de Ludwig et al. (2003), um modelo abstrato para o gerenciamento de serviços web, em tempo de execução. O modelo em questão é denominado WSLA (Web Service Level Agreement), no qual é dividido em três grupos de funcionalidades para medição e gerenciamento:

- **Medição**: realiza tarefas de cálculo das métricas, como o tempo de resposta.
- **Conjunto de métricas**: São as definições das métricas que devem ser gerenciadas pelo grupo de funcionalidade Medição;
- **Avaliação da condição**: informa as garantias e violações do SLA, baseadas nas medições realizadas.

A leitura do WSLA, onde sua estrutura é descrita em XML, tem as seguintes composições: as partes envolvidas, como o nome dos serviços e consumidores; as definições do serviço comum às duas partes; quais serão as métricas utilizadas; como será realizada a medição; bem como as obrigações e condições para realizar o serviço. A

Figura 4.1 apresenta os níveis de interações entre os produtores e consumidores dos serviços.

Figura 4.1 - Gerenciamento em tempo de execução de um WSLA.



Fonte: Ludwig et al. (2003)

4.1.2 Ferramentas de avaliação da QoS

Não é viável o controle de um satélite sem o auxílio dos instrumentos, estações, e os centros de comunicação e controle. As atividades de operação estariam severamente prejudicadas, caso não fossem sabidas as situações dos componentes internos de uma carga útil, a conexão com as estações de recepção, as informações extraídas do sensor de temperatura do satélite, entre muitas outras definições.

Esse tipo de situação é também necessária em outros segmentos; dessa maneira, são

necessárias ferramentas para monitoramento periódico de um parque computacional de uma organização. É essencial a utilização dos dados coletados para a geração de gráficos e alarmes, que auxiliam na tarefa de manter a operação dos dispositivos de uma arquitetura.

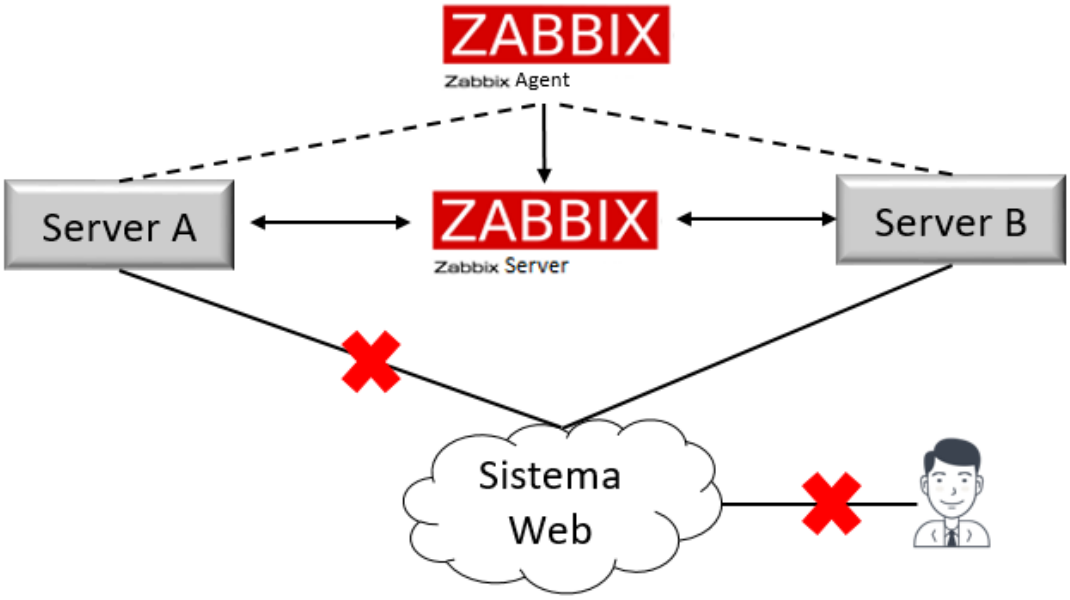
Devido à diversidade das ferramentas, a escolha deve ser realizada de acordo com a necessidade de monitoramento da organização. De uma maneira geral, a maioria das ferramentas disponibiliza serviços para o monitoramento de métricas, tais como: tempo de resposta de um servidor, de um site, de um *web service*, de um sensor, dentre outros; taxa de erros por requisição; espaço livre em discos; quantidade de memória utilizada; tempo de *downtime* e *uptime* de um serviço; gráficos de utilização de CPU; número de processos ativos, etc.

O acompanhamento do comportamento dos dispositivos e serviços permite criar ações de predição de falhas, e ainda pode-se configurar o ambiente para a tomada de ações baseadas em acordos de SLA. Dessa forma, o monitoramento torna a garantia da qualidade de serviço mais eficaz, aumenta a robustez, na medida em que torna o tempo de inoperância do sistema "downtime" aceitável.

O CACTI (https://www.cacti.net/) é uma ferramenta de monitoramento que pode ser configurada facilmente através de um interface web. Nela, é possível criar *scripts* personalizados para a realização da coleta dos dados. Todos os dados coletados ficam armazenados em uma base de dados, podendo ser obtidos a qualquer momento pelo administrador do sistema.

O Zabbix (https://www.zabbix.com/) é um software de monitoramento de código aberto e possui mais de cem parceiros no mundo todo, os quais colaboram com seu desenvolvimento e melhoria. Nele, é possível realizar o monitoramento de diversos tipos de dispositivos de forma gratuita. Ele pode enviar SMS ou email, e permite a comunicação com protocolos do tipo HTTP, MQTT, TCP e outros. Ainda é possível trabalhar com gatilhos (*trigger*) e automatizar tarefas. A Figura 4.2 exemplifica um sistema web que ficou indisponível após atingir um limite máximo de requisições de download e, automaticamente, com base no monitoramento do ambiente, o Zabbix realiza a troca do servidor A pelo servidor B, que passa a ser o principal.

Figura 4.2 - Monitoramento dos servidores usando o Zabbix.



Fonte: ZABBIX

O Zabbix possui dois modos de verificação:

- **Ativo**: os agentes de monitoramento são instalados nos componentes dos sistemas. Assim, os agentes enviam os dados coletados das aplicações monitoradas e, em seguida, enviam-nas para o servidor e, dessa forma, é possível diminuir a sobrecarga de acesso aos sistemas monitorados pelo Zabbix service.

- **Passivo**: o servidor do Zabbix coleta a informação diretamente do sistema alvo.

Caso o Zabbix Server esteja em processo de atualização, inoperante ou em manutenção, os agentes continuam coletando os dados, que são enviados para o servidor, quando a disponibilidade é restaurada.

Para realizar o monitoramento de web sites, a ferramenta gratuita chamada Google Analytics (GA) é uma das principais disponíveis no mercado, permitindo realizar um planejamento analítico e mensurando corretamente diversos indicadores essenciais para melhorar a qualidade de serviço. Nela, é possível estimar como os usuários estão

se comportando dentro do seu site, além de realizar o monitoramento da análise de desempenho, controle de tráfego, otimização do conteúdo a fim de aumentar as visitas do website e diminuir o número de rejeição, bem como estimar as localizações geográficas dos usuários, o tipo de sistema operacional do cliente, dentre outros.

4.1.3 Big Data e Stream Analytics

Outro problema que pode produzir impacto para as novas arquiteturas de sistemas espaciais é a produção de dados em altas taxas por diversos instrumentos embarcados como carga útil de satélites, e também outros instalados em solo, tais como antenas, magnetômetros, entre outros.

Pode ser verificado o mesmo problema em outras áreas. Gartner Inc (2015) aponta que 13,5 bilhões de dispositivos estarão conectados até 2020. Esse aumento considerável de dispositivos conectados, associado ao crescimento da computação em nuvem, provocará uma mudança no paradigma da rede distribuída para a centralização, sendo o núcleo os centros de dados dos provedores de serviços na nuvem. Apesar de suas vantagens, o rápido aumento de dispositivos móveis e sensores conectados à Internet desafia a arquitetura de rede tradicional. Com o IoT, bilhões de dispositivos heterogêneos interligados irão emitir um grande volume de dados para processamento.

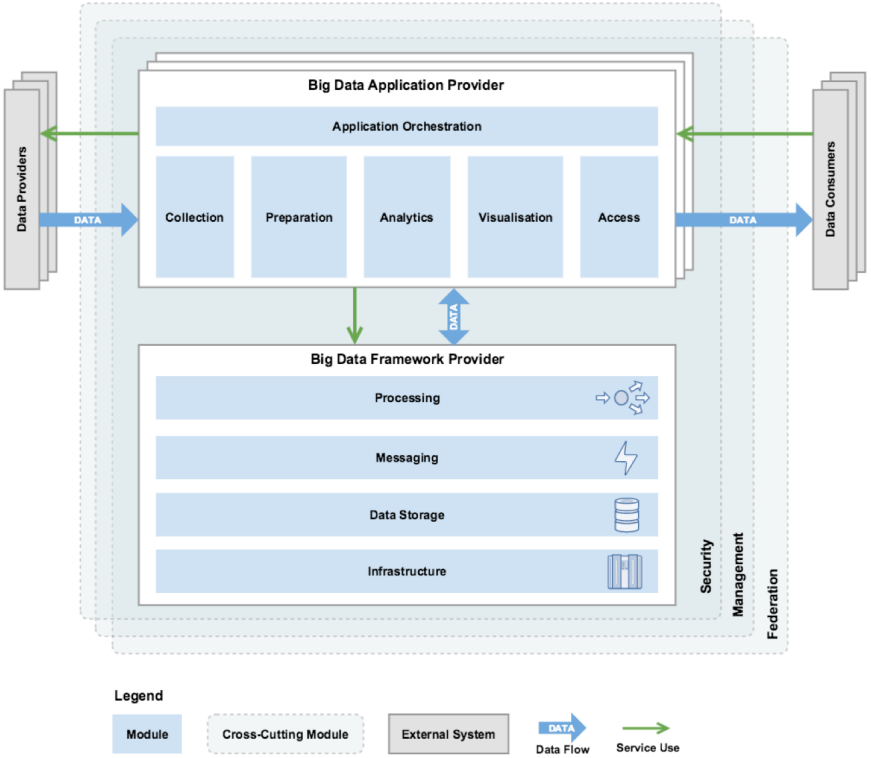
Essa quantidade massiva de dados exigirá processamento e análise em tempo real. Nesse contexto, a transferência de todos os dados brutos para a nuvem pode acarretar sérios problemas relacionados à escalabilidade, além de as arquiteturas atuais não estarem adequadas para a tomada de decisões em tempo real. Para atender a requisitos de escalabilidade dinâmica, processamento eficiente na rede, e comunicação em tempo real de baixa latência, surge a necessidade de fazer evoluírem as arquiteturas de software atuais de modo a atenderem a estes requisitos.

Nesse contexto, Taneja e Davy (2016) apresentam o conceito de *"Fog Computing"*. Essa abordagem consiste em uma arquitetura com três camadas, onde a primeira contém os dispositivos IoT, que enviam os dados brutos para a camada superior; a segunda camada denominada *"Fog Tier"*, a qual contém dispositivos como roteadores, gateways, etc. Essa segunda camada pode ser escalada localmente e permite gerenciamento local e programação de aplicações que enviam dados para a camada 3. A última camada é chamada *Cloud*, a qual recebe os dados da camada Fog. Esta organização permite uma redução drástica de latência e volume de dados para a camada Cloud. Resultados interessantes são apresentados no artigo.

Big Data também procura tratar questões relativas ao processamento massivo de informação. Uma arquitetura de referência para Big Data se concentra no atendimento aos requisitos típicos de defesa (KLEIN et al., 2016) e demonstra como usar essa arquitetura de referência para definir soluções em uma dada missão.

O metamodelo da arquitetura de referência mostrada na Figura 4.3 consiste de uma coleção de 13 módulos, que decompõem a solução em elementos que realizam funções ou capacidades e que abordam um conjunto de preocupações *"concerns"* coesas. As preocupações são abordadas por padrões de solução, ou por estratégias. Há, ainda, um processo de projeto que orienta o projetista na aplicação da arquitetura de referência para produzir uma solução para uma determinada missão.

Figura 4.3 - Arquitetura de Referência para Big Data aplicada ao domínio de defesa.



Fonte: Klein et al. (2016)

IoT e Big Data produzem um volume enorme de dados, os quais precisam ser processados e, de alguma maneira, sumarizados/agrupados em tempo real, fornecendo cálculos enquanto os dados são enviados pelos dispositivos e componentes de software. Arquiteturas de software atuais não estão preparadas para esta realidade. Bancos de Dados relacionais e aplicações convencionais não possuem a capacidade de rece-

ber informação e efetuar cálculos sem primeiramente armazená-los. Esta deficiência exige uma infraestrutura de hardware e armazenamento que, por mais robusta que seja, não suportaria gigabytes ou mesmo terabytes de informação transmitidas em tempo real.

Nesse sentido, surge o stream analytics; o conceito aborda a coleta, recepção e processamento de dados em tempo real sem o armazenamento da informação (GOUSIOS et al., 2016). Algumas soluções e frameworks, como o Go Pulsar e o Apache Flink, vão nessa direção.

Pulsar (http://gopulsar.io/) é uma plataforma de análise em tempo real (stream analytics) de código aberto. Ela pode ser usada para coletar e processar eventos em tempo real para fornecer dados chave e permitir que os sistemas reajam a ações em segundos. Pulsar fornece sessões em tempo real, agregação de métricas multidimensionais e criação de fluxo personalizado através de enriquecimento, mutação e filtragem de dados, usando um idioma de processamento de eventos semelhante à linguagem SQL (Structure Query Language).

Outra solução é o Apache Flink (APACHE FOUNDATION, 2018) (https://flink.apache.org/), um sistema de código aberto para uma análise de dados que é expressiva, declarativa, rápida e eficiente em dados históricos e em tempo real (transmissão). O Flink combina a escalabilidade e a flexibilidade de programação das plataformas MapReduce com a eficiência, a execução e os recursos de otimização de consultas encontrados em bancos de dados paralelos. O Flink unifica processamento computacional em lote e incremental em um pipeline real-streaming. O Flink está convergindo em um sistema completo para processamento paralelo de dados com uma ampla gama de bibliotecas, que vão desde o aprendizado de máquina até o processamento de gráficos. O Apache Flink é originário do projeto Stratosphere (TU BERLIN AND HUMBOLDT UNIVERSITY AND THE HASSO PLATTNER INSTITUTE, 2018), liderado pela TU Berlin.

4.2 Frameworks de Software

Frameworks (RIEHLE, 2000) modelam um domínio específico ou um aspecto importante do software. Eles representam o domínio como um projeto (design) abstrato, consistindo de classes abstratas (ou interface). O design abstrato é mais do que um conjunto de classes, porque define de que maneira as instâncias das classes colaboram umas com as outras em tempo de execução.

Um framework trás implementações reutilizáveis na forma de classes abstratas e concretas. As classes abstratas implementam partes de uma abstração do framework, mas as decisões de implementação que são cruciais para o sistema (como as táticas de adaptação, por exemplo) ficam a cargo das subclasses.

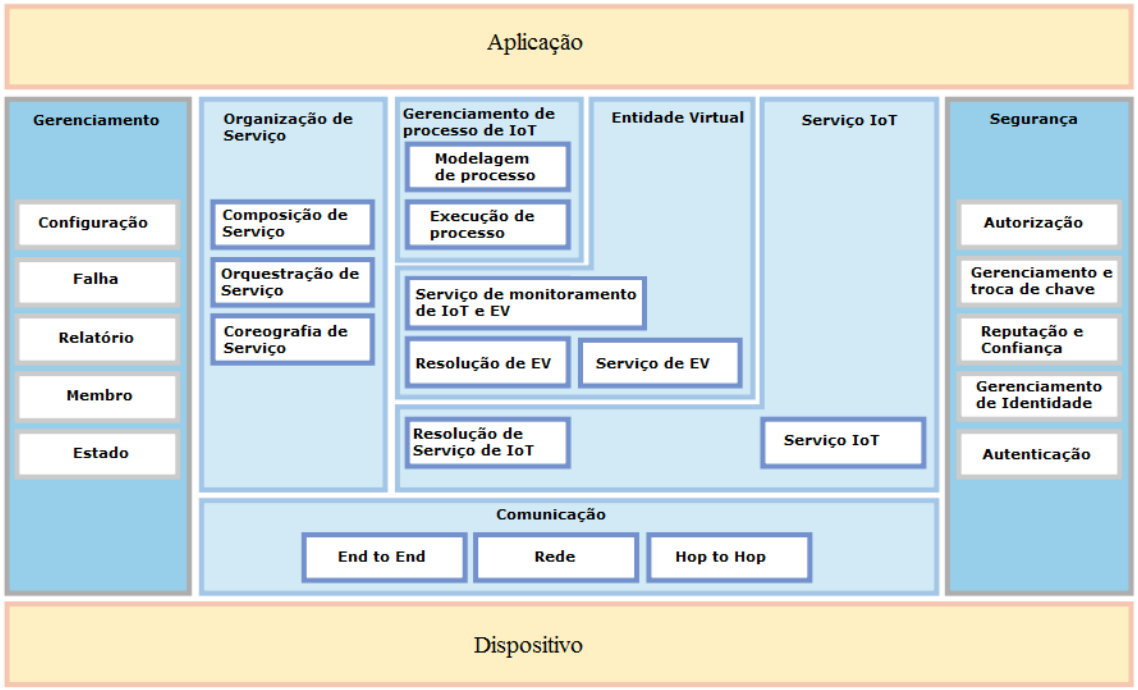
4.3 Internet das Coisas (IoT)

A Internet das Coisas (do inglês Internet of Things IoT) é um paradigma que preconiza um mundo de objetos físicos embarcados com sensores e atuadores, conectados por redes sem fio, e que se comunicam usando a Internet, moldando uma rede de objetos inteligentes capazes de realizar variados processamentos, capturar variáveis ambientais e reagir a estímulos externos. Esses objetos interconectam-se entre si e com outros recursos (físicos ou virtuais) e podem ser controlados através da Internet, permitindo o surgimento de uma miríade de aplicações que poderão se beneficiar dos novos tipos de dados, serviços e operações disponíveis (PIRES et al., 2015).

- infraestrutura robusta e tolerante a falhas;
- gerenciamento de incertezas e resolução de conflitos;
- suporte à adaptação de aplicações sob condições dinâmicas do ambiente;
- escalabilidade;
- governança;
- privacidade e segurança;
- processar massiva quantidade de dados;
- dispositivos interconectados;
- eficiente em termos do consumo de recursos dos dispositivos;
- atender às demandas de aplicações em tempo real ou quasi-real.

Nesse sentido, faz-se necessária a criação de arquiteturas de referência que definem um conjunto inicial de blocos de construção para ambientes de IoT, que leve em consideração os requisitos apontados no parágrafo anterior e que forneçam uma base sólida para alavancar sua ampla adoção. Uma arquitetura de referência pode ser entendida como uma arquitetura abstrata que envolve conhecimento e experiências

Figura 4.4 - MAR - modelo arquitetural de referência IOT-A



Fonte: Adaptada de Bassi et al. (2016)

em um domínio de aplicação específico, sendo capaz de facilitar e guiar o desenvolvimento, a padronização e a evolução de sistemas de software em tal domínio.

O projeto IoT-A (BASSI et al., 2016) propõe um modelo arquitetural de referência (MAR) que envolve uma arquitetura de referência base e a definição de um conjunto de características chave para sua construção. Essa arquitetura de referência é definida em um alto nível de abstração, fornecendo visões arquiteturais e perspectivas que são relevantes para a construção de várias arquiteturas para IoT. A figura 4.4 apresenta uma decomposição funcional do modelo de referência apresentado em (BASSI et al., 2016).

Os conceitos relacionados à arquitetura de referência IoT forneceu um modelo que foi aplicados na arquitetura do framework objeto deste trabalho de pesquisa, detalhado no capítulo 5.

4.4 Inteligência Artificial (IA)

Segundo Russell e Norvig (2009), inteligência procura entender o pensar humano, isto é, como um punhado de matéria pode perceber, compreender, prever e manipular

um mundo muito maior e mais complexo do que ela própria. A Inteligência Artificial (IA) vai além: ela tenta não apenas compreender, mas também construir entidades inteligentes.

Diversas técnicas foram desenvolvidas no campo da IA, dentre elas, destacamos as redes neurais, aprendizado de máquina, algoritmos evolutivos (MITCHELL, 1998), sistemas baseado em regras (GROSAN; ABRAHAM, 2011), etc.

Destas tecnologias, foi empregado, na construção do protótipo apresentado no capítulo 6, a utilização de um motor de inferência para analisar métricas de QoS e verificar se há algum plano de adaptação aplicável.

4.4.1 Sistemas baseados em Regras

Um sistema baseado em regras utiliza regras como representação do conhecimento codificado no sistema (GROSAN; ABRAHAM, 2011). As definições do sistema dependem quase totalmente de sistemas especializados, um sistema que imita o raciocínio do especialista humano na resolução de um problema intensivo em conhecimento. O sistema baseado em regras representa o conhecimento em termos de um conjunto de regras que diz o que fazer ou o que concluir em diferentes situações.

Tal sistema pode ser criado simplesmente usando um conjunto de assertivas e um conjunto de regras que especificam como agir no conjunto de afirmações.

As regras são expressas como um conjunto de instruções if-then:

if P then Q

é equivalente a:

P => Q

Um sistema baseado em regras consiste de um conjunto de regras if-then, um conjunto de fatos e algum interpretador que controla a aplicação das regras de acordo com os fatos dados. A idéia de um sistema especialista é utilizar o conhecimento de um especialista e codificá-lo em um conjunto de regras. Quanto exposto a um conjunto de dados, o sistema especialista atuará de maneira similar ao especialista.

Sistemas baseados em regras são modelos muito simples, que podem ser aplicados a um grande número de problemas. O requisito é que o conhecimento de uma determinada área possa ser expresso na forma de regras, atentando-se para o fato que

um grande número de regras pode tornar a solução ineficiente.

4.4.2 Elementos de um sistema baseado em regras

Qualquer sistema baseado em regras consiste de alguns elementos descritos a seguir:

a) **Um conjunto de fatos**: Um conjunto de assertivas que devem ser qualquer coisa relevante para iniciar o estado do sistema.

b) **Um conjunto de regras**: Um conjunto de ações que devem ser tomadas de acordo com o escopo do problema e o conjunto de assertivas fornecidas. A regra relaciona os fatos como parte do IF e as ações são consequências da parte THEN. O sistema deve conter somente regras relevantes, deve-se evitar as regras irrelevantes de modo a obter uma melhor eficiência do sistema.

c) **Um critério de terminação**: Esta é uma condição que determina se uma solução foi encontrada ou que não existe nenhuma. Isso é necessário para terminar o processamento do sistema especialista e evitar loops infinitos por exemplo.

A tabela 4.1 apresenta um exemplo concreto de uma regra.

Tabela 4.1 - Exemplo que apresenta partes de uma regra e interações

Dados	**Condições**	**Regras**
Nome do Componente solicitações por segundo tempo de resposta	Serviço 1, > 1000, > 100, Tráfego elevado, Desempenho em degradação	**Premissas** IF solicitações por segundo > 1000 AND tempo de resposta > 100 AND nome do componente = Serviço 1 **Conclusão** THEN ativar novo nó

Fonte: Adaptada de Grosan e Abraham (2011)

Abordagens modernas para construção de sistemas baseados em regras, como o Jess (HILL, 2003) e o Jboss Drools (JBOSS, 2017), incluem motores de inferência para

confronto de regras e fatos. Essas abordagens implementam algoritmos de inferência muito eficiente, que permitem a execução de muitas regras com excelente desempenho, além de permitirem a integração com linguagem Orientada a Objetos como o Java.

5 ARCTIC FOX: UM FRAMEWORK PARA ADAPTAÇÃO DINÂMICA DE SISTEMAS

Neste capítulo, é apresentada uma visão geral do framework de adaptação de software, que atenda às necessidades de adaptação de diversas missões da área espacial.

5.1 Visão Geral

Um sistema auto-adaptável altera o seu comportamento em resposta às alterações do seu ambiente. Por ambiente entende-se que sua composição consiste de componentes de software, rede, equipamentos, arquivos de configuração, servidores de aplicação, bases de dados, isto é, tudo aquilo que é parte do sistema. Qualquer alteração que venha a ocorrer em um desses elementos pode impactar severamente no funcionamento do sistema, de uma maneira geral.

Neste trabalho de pesquisa, é apresentado um *framework* denominado Arctic Fox, para adaptação dinâmica de sistemas, o qual provê suporte auto-adaptativo a sistemas de segmento solo. O framework possui um mecanismo de gerenciamento do ciclo de vida de adaptação, baseado em Monitoramento do ambiente, no qual se insere o sistema, análise de dados observáveis, planejamento e execução de alterações no sistemas de software de forma autônoma.

É necessário fornecer uma estrutura base para o desenvolvimento de sistemas auto-adaptáveis, adotar uma estratégia parecida com a abordagem FLAGS, onde a adaptação é tratada no desenvolvimento de software como requisitos específicos, e envolve a aplicação de conhecimento que não é largamente difundido para desenvolvedores de software, o que certamente dificulta a tarefa de construção de sistemas auto-adaptáveis.

Um *framework* (RIEHLE, 2000) modela um domínio específico ou um aspecto importante deste. Ele representa o domínio como um projeto (design) abstrato, consistindo de classes abstratas (ou interface). Um framework fornece implementações reutilizáveis na forma de classes abstratas e concretas, além de interfaces ou anotações. Este trabalho propõe um framework para prover suporte à adaptação dinâmica para sistemas de segmento solo, e sua concepção foi norteada pelas premissas listadas a seguir:

a) Implementação de um framework para prover suporte ao desenvolvimento de sistemas auto-adaptáveis;

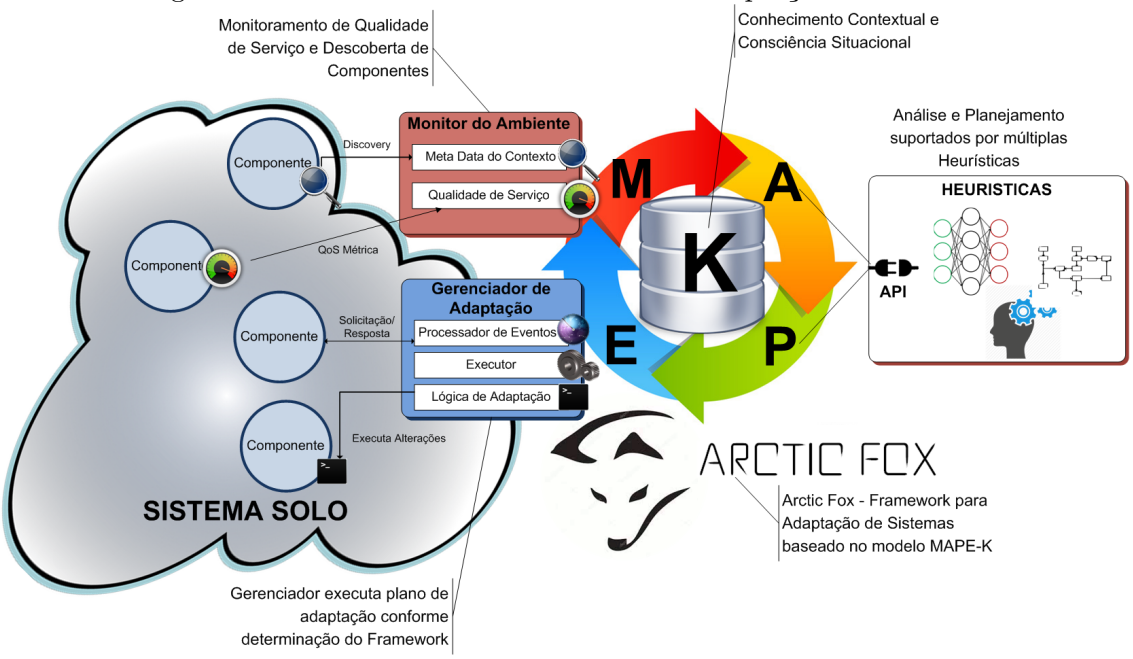
b) Adoção do modelo MAPE-K para gestão do ciclo de vida de adaptação;

c) Monitoramento e Conhecimento do contexto e da situação dos elementos gerenciados;

d) Suporte a múltiplas heurísticas;

Na figura 5.1, é apresentada uma visão geral com as premissas adotadas para o framework.

Figura 5.1 - Visão Geral do Framework de Adaptação de Sistemas



Fonte: Autor

5.1.1 Framework de Adaptação

Para o contexto deste trabalho, foi definido que a implementação do *framework* proposto captura os conceitos centrais de sistemas auto-adaptáveis, além de fornecer meios para a reutilização e extensão para diversos tipos de sistemas.

Os requisitos para o framework ArcticFox levam em consideração tanto as características de um sistema auto-adaptável e suas auto-propriedades como também as necessidades de missão dos sistemas espaciais. Os requisitos em questão são descritos a seguir:

a) O framework deve conhecer os dispositivos do sistema alvo;

b) O framework deve conhecer os componentes de software do sistema alvo;

c) O framework deve permitir o monitoramento de atributos de dispositivos do sistema alvo;

d) O framework deve permitir o monitoramento de atributos de componentes do sistema alvo;

e) O framework deve prover facilidades para cálculo de métricas de qualidade de serviço;

f) O framework deve permitir a definição de critérios de qualidade de serviço;

g) O framework deve permitir a configuração de eventos observáveis do sistema solo;

h) O framework deve prover facilidades de implementação de lógica de adaptação;

i) O framework deve identificar componentes de software;

j) O framework deve identificar dispositivos;

k) O framework deve conhecer os componentes e conectores da arquitetura do sistema solo.

l) O framework deve manter os estados dos dispositivos atualizados;

m) O framework de adaptação deve permitir adaptação on-the-fly;

n) O framework de adaptação deve permitir a definição do nível de automação da adaptação;

o) O framework deve ser extensível para permitir a implementação de novas estratégias de adaptação;

p) O framework deve permitir a configuração de estratégias de análise por recurso gerenciado;

q) O framework deve ser extensível para permitir a implementação de comunicação com diferentes protocolos de comunicação;

r) O framework deve ser escalável para atender a aumento de volume de eventos para processamento;

s) O framework não deve consumir recurso computacional de forma a degradar o desempenho do sistema alvo.

t) O framework deve gerenciar múltiplos gerenciadores autônomos de forma concorrente.

Para atendimento aos requisitos listados, a arquitetura do framework deve prever um conjunto de facilidades que permitam a adequação do sistema de software a um cenário de adaptação autônoma de software. Para tanto, prevê-se que o framework deve fornecer:

- Pacotes de classes, interfaces e anotações para permitir o desenvolvimento de Lógicas de Adaptação por desenvolvedores de software;
- Mecanismo de extensão para cenários que envolvem protocolos específicos de comunicação, como, por exemplo, CCSDS (CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS (CCSDS), 2003) e IP (TANENBAUM; WETHERALL, 2010);
- Mecanismo de extensão do *framework* que permita o desenvolvimento de heurísticas que viabilizem a implementação de algoritmos diferentes, que podem ser mais adequados a diferentes necessidades de adaptação, como, por exemplo, a utilização de redes neurais para cenários de auto-contenção, a utilização de algoritmos evolutivos para cenários de auto-otimização, entre outras.

No *framework* proposto, a lógica de adaptação é implementada por meio de táticas que são executadas para atingir um determinado nível de qualidade de serviço (QoS). Nesse sentido, o framework deverá fornecer facilidades em forma de API de desenvolvimento, anotações, e um processador de anotações que permita ao framework identificar essas táticas, relacioná-las a elementos gerenciados e atributos de qualidade de serviço.

Essas facilidades também englobam elementos que permitam a um determinado agente de software receber um conjunto de instruções do gerenciador MAPE e executar as táticas para alterar o sistema alvo.

As facilidades informadas devem permitir sua reutilização de modo a agilizar e melhorar a qualidade e produtividade de sistemas auto-adaptáveis.

5.1.2 Modelo MAPE-K

Para o gerenciamento adequado do ciclo de vida de adaptação, este trabalho adota o modelo MAPE-K (KEPHART; CHESS, 2003). Conforme apresentado na seção 3.2.3, o modelo em questão é o mais utilizado nas abordagens para adaptação pesquisadas. O modelo em questão restringe a arquitetura da solução, onde cada elemento é descrito a seguir:

- Monitoramento (M): No *framework* Arctic Fox, o monitoramento ocorre por meio de coleta de métricas de qualidade de serviço dos elementos gerenciados.
- Análise (A): O processo de análise é realizado por meio de heurística, que pode ser acoplada ao framework, e que pode ser especializada para cada tipo de arquitetura. A heurística é responsável por definir um plano de adaptação aplicável ao elemento gerenciado.
- Planejamento (P): O planejamento, neste trabalho de pesquisa, é realizado através de rotinas para identificação do elemento gerenciado, ordenação de atividades do plano de adaptação e encaminhamento para execução.
- Execução (E): A parte de execução se dá pela interação entre o framework, que envia os planos de adaptação, e os agentes de software, que os recebem.
- Conhecimento (K): Toda a informação relativa ao ambiente no qual se insere o sistema alvo é armazenada para consulta pelo gerenciador MAPE.

Ainda com relação à adoção do modelo MAPE-K, a lógica de adaptação e o gerenciador autônomo MAPE devem estar desacoplados. Esta abordagem permite alocar mais recurso computacional para a execução das heurísticas diferentes de gerenciamento autônomo. Porém, esta abordagem centraliza as tarefas MAPE em um componente central, o que gera um ponto de falha, sendo esta uma desvantagem em relação à abordagem A3 (BARESI; GUINEA, 2011)(GUINEA; SAEEDI, 2012), discutida na seção 3.2.5.1, por exemplo.

Do ponto de vista da lógica de adaptação, devido ao desacoplamento, esta pode ser distribuída entre componentes e dispositivos do sistema com pouca necessidade de processamento. Um elemento de software é responsável por receber um plano de adaptação para execução.

5.1.3 Conhecimento Contextual e Consciência Situacional

Na seção 3.2.2, podem ser observadas diversas condições para adaptação de sistemas. Para que estas condições sejam satisfeitas, o conhecimento do contexto e a consciência da situação do sistema são elementos centrais para viabilizar a adaptação dinâmica e autônoma.

Consciência e conhecimento do sistema são representados pela letra K, que é a abreviação de "Knowledge Database", do modelo MAPE-K, onde a informação, pela qual a solução toma decisões para gerenciar o ciclo de vida de adaptação dos componentes de software, é armazenada em uma base de dados de conhecimento.

Para este trabalho, contexto significa manter informação relacionada a:

- Elementos de Hardware que compõem o sistema (dispositivos).
- Meta-Informação de Elementos de Software e relacionamentos (componentes).
- Informação relacionada a atributos de qualidade de serviço (QoS).
- Informação relacionada a quais elementos serão gerenciados pelo Framework.
- Informação sobre Lógica de Adaptação, localização e condições de disparo.
- Informação relacionada a agentes de software que monitoram e executam a lógica de adaptação.
- Meta-informação relacionada a algoritmos de análise para suporte ao ciclo de adaptação do sistema.
- Estados dos elementos, tais como ativo, inativo, avariado, sobrecarregado, etc.

Consciência tem uma conotação situacional, sendo, portanto, mais abrangente que conhecimento da estrutura. O framework não apenas conhece sua estrutura, isto é, componentes, conectores e dispositivos, como também seus estados operacionais, bem como métricas de qualidade. Essa diversidade de informação permite ao framework tomar decisão com maior precisão.

A descoberta de componentes e dispositivos é fundamental para a integridade e a consistência da informação. Apesar da indicação de que sistemas espaciais terão maior probabilidade de serem sistemas fechados, onde os elementos são conhecidos, entende-se que atividades de manutenção e evolução dos componentes do sistema alvo podem causar degradação arquitetural.

Alterações no ambiente do sistema solo podem ocorrer constantemente, seja por motivos de degradação de servidores e demais equipamentos, seja por alterações de componentes de software, ou mesmo re-avaliação da utilização de recursos. Após entrada em operação, frequentemente há aumento de incerteza, devido ao fato de os ambientes nos quais os sistemas operam obrigarem os operadores a tomar ações para correção de eventuais problemas de modo a manter o sistema em operação.

O monitoramento do ambiente no qual se insere o sistema solo deve ser tratado pelo framework por meio da coleta e cálculo de métricas de qualidade de serviço (QoS). Anomalias detectadas nas métricas podem justificar adaptações no sistema solo para corrigir problemas, otimizar utilização de recursos, redirecionar tráfego, escalar componentes de software, dentre outras ações previstas para adaptação.

A adaptação, no contexto deste trabalho, dá-se pela execução de táticas, que são disparadas de acordo com a análise dos valores de métricas de qualidade de serviço (QoS) do sistema solo, definidas na base de conhecimento.

As táticas para atingir os diferentes atributos de qualidade de serviço são independentes de arquiteturas e tecnologia. Desta maneira, o framework deve permitir que a adaptação seja implementada nesse contexto.

Como exemplo, agentes de adaptação para monitoramento de QoS, e execução das táticas, podem adotar estratégias de implementação como:

- Técnicas de Orientação a Aspectos;
- Uso de filtros e interceptadores para aplicações corporativas;
- Consultas e/ou Configurações em Banco de dados;
- Leitura e alteração de arquivos;
- Execução de comandos de sistema operacional;

O mecanismo de extensão do framework deve prover facilidades para a implemen-

tação dos agentes computacionais que atendam às restrições inerentes a cada arquitetura e tecnologia empregada nos sistemas.

O exemplo da abordagem FLAGS (BARESI et al., 2010), discutida na seção 3.2.5.5, prevê que a adaptação deve ser considerada no desenvolvimento de sistemas como requisitos que devem ser endereçados por eles. O problema dessa abordagem são as mudanças que ocorrem no ambiente do sistema após a entrada em operação, e que podem requerer ajustes no mecanismo de adaptação devido à degradação arquitetural, a alterações funcionais, etc. Desta maneira, os mecanismos de adaptação podem não ser efetivos.

Entende-se que, idealmente, uma abordagem dinâmica para adaptação de sistemas deve considerar as mudanças de contexto e as incertezas, de modo a se adequar a novas necessidades de adaptação, sem alteração do projeto do sistema. Nesse sentido, a adaptação "on-the-fly" deve ser característica predominante do *framework* Arctic Fox.

5.1.4 Suporte a Múltiplas Heurísticas

Diferentes necessidades de adaptação podem requerer diferentes algoritmos/técnicas para suporte ao ciclo de vida de adaptação, conforme bem observado em (OREIZY et al., 1999), mais precisamente na figura 3.4, explorada no capítulo 3.

Isto significa que o framework deve prover um mecanismo de extensão que permita ao desenvolvedor programar algoritmos diferentes, registrá-los no framework e recalibrar o mecanismo de adaptação, para que cada tipo de algoritmo possa ser utilizado da maneira mais eficaz possível.

Neste contexto, eficácia significa determinar qual é a heurística que melhor se aplica à análise de um conjunto de métricas de QoS, auto-propriedade que se pretende implementar e consequente seleção de táticas para viabilizar a adaptação autônoma.

Auto-propriedades de um sistema auto-adaptável se relacionam com os atributos de qualidade de serviço e táticas. De modo a exemplificar o relacionamento exposto, é apresentado um mapeamento sobre auto-propriedades e táticas aplicáveis a dois atributos de qualidade de serviço largamente utilizados, são eles Disponibilidade e Desempenho. Os mapeamentos podem ser vistos nas tabelas 5.1 e 5.2, respectivamente.

Tabela 5.1 - Auto-propriedades e táticas aplicadas ao Atributo de QoS Disponibilidade

Auto-propriedade	**Categoria QoS**	**Tática**
Autoconfiguração	Recuperação de Falha	Votação
Autoreparação	Recuperação de Falha	Redundância Ativa Redundância Passiva
Auto-otimização	Recuperação de Falha	Votação
Autoproteção	Recuperação de Falha	Redundância Ativa Redundância Passiva Spare
Autoestabilização	Recuperação de Falha	Spare Ressincronização de Estado
Autoescala	Prevenção de Falha	Spare Redundância Ativa
Autoimunidade	Prevenção de Falha	Remoção de Serviço Redundância Ativa Redundância Passiva Spare
Autoretenção	Recuperação de Falha	Spare Redundância Ativa Redundância Passiva

Fonte: Autor

Tabela 5.2 - Auto-propriedades e táticas aplicadas ao atributo de QoS Desempenho

Auto-propriedade	**Categoria QoS**	**Tática**
Autoconfiguração	Demanda de Recursos	Aumentar Eficiência Computacional
Autoreparação	Gestão de Recursos	Introduzir concorrência Manter múltiplas cópias de dados ou funcionalidades Aumentar disponibilidade de recursos
Auto-otimização	Demanda de Recursos Gestão de Recursos Arbitrar Recursos	Gerenciar Taxa de Eventos Controlar frequência de solicitações Política de Escalonamento Manter múltiplas cópias de dados ou funcionalidades Aumentar disponibilidade de recursos Reduzir sobrecarga computacional
Autoproteção	Demanda de Recursos	Política de Escalonamento
Autoestabilização	Gestão de Recursos	Introduzir concorrência Manter múltiplas cópias de dados ou funcionalidades Aumentar disponibilidade de recursos
Autoescala	Gestão de Recursos	Manter múltiplas cópias de dados ou funcionalidades Aumentar disponibilidade de recursos
Autoimunidade	Demanda de Recursos Gestão de Recursos Arbitrar Recursos	Introduzir concorrência Limitar Tempo de Execução Limitar tamanhos de fila Políticas de escalonamento
Autoretenção	Demanda de Recursos Gestão de Recursos Arbitrar Recursos	Limitar Tempo de Execução Limitar tamanhos de fila Políticas de escalonamento Aumentar disponibilidade de recursos

Fonte: Autor

5.2 Componentes Funcionais do Framework

Para atender às necessidades apontadas nas seções anteriores, foi definido um conjunto de componentes para a arquitetura do framework.

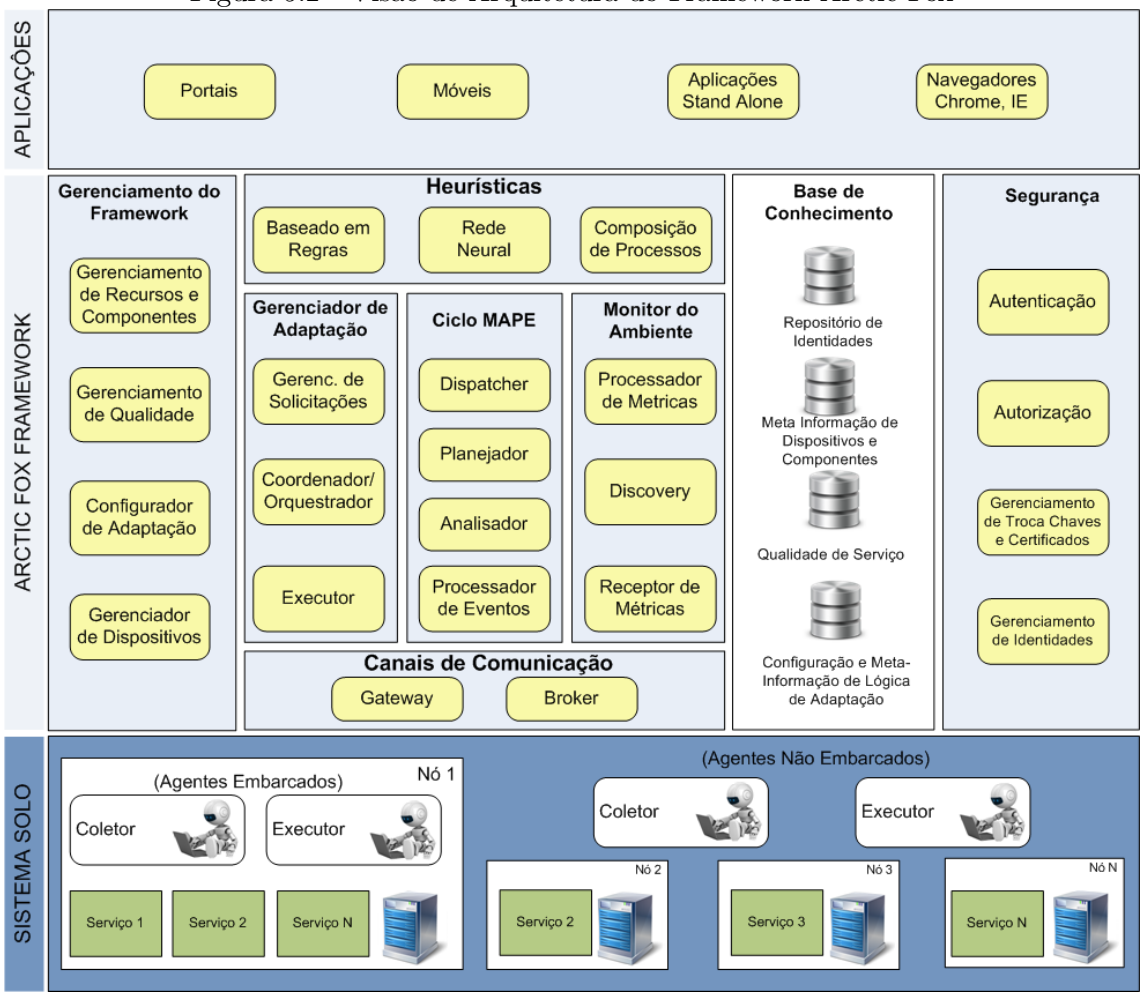
Como não há possibilidade de construção de uma solução compatível com todos os cenários possíveis de adaptação de uma missão espacial, a idéia central dessa abordagem está em modelar o domínio de aplicação de um sistema auto-adaptável para o segmento solo, e criar uma estrutura base reutilizável, que possa ser estendida para atender às necessidades específicas de cada missão.

Nesse sentido, foram definidos componentes funcionais responsáveis pelo Gerenciador Autônomo do framework, Desenvolvimento da Lógica de Adaptação, Componentes de Gerenciamento, Componentes para Segurança e Bases de Conhecimento. Na figura 5.2, é apresentada uma visão geral dos componentes do framework.

O framework proposto possui um conjunto de componentes distribuídos em três camadas, sendo a camada mais inferior o próprio sistema solo ou sistema alvo, ou seja, o elemento que fornece os recursos que serão monitorados e adaptados pelos agentes do framework. Na camada intermediária está o arctic fox framework, onde se encontram o gerenciador autônomo, mecanismos de extensão e configuração de parâmetros de adaptação. E a terceira e última camada é a de aplicações, a qual permite que outras aplicações possam utilizar facilidades encontradas no *framework*, para interagir com os componentes gerenciados e fazer consultas a métricas e a outras informações disponíveis. Uma breve descrição dos elementos principais é apresentada a seguir:

- **Sistema Solo**: É o sistema de software que será monitorado e alterado, e que fornece os recursos que serão monitorados e adaptados pelo framework. O sistema é conectado ao framework para monitoramento e adaptação. Essa conexão pode ser obtida por meio da implantação de agentes coletores de métricas de QoS e agentes executores de adaptação, através de dois formatos:
  - ***Agente Embarcado***: Pode ser utilizada uma API embarcada junto ao código-fonte de componentes do sistema alvo para monitoramento ou adaptação. A adaptação, neste caso, pode ser obtida de várias formas, como, por exemplo, através de técnicas de orientação a aspectos, captura de eventos, interceptadores, etc.

Figura 5.2 - Visão de Arquitetura do Framework Arctic Fox



Fonte: Autor

- ***Agente Não Embarcado***: Está fora dos componentes do sistema alvo. Neste caso, um elemento externo pode ser utilizado para monitoramento ou adaptação. A adaptação pode, então, ser obtida de várias formas, como, por exemplo, através de implementação de proxies web, aplicações convencionais que alteram parâmetros de banco de dados para fins de auto-configuração, interceptadores, etc.

- **Arctic Fox Framework**: Nesta camada, o framework é responsável por receber as métricas, calcular métricas derivadas, analisar os dados, planejar a execução e encaminhar o plano para execução no sistema alvo. Segue descrição dos componentes principais:

  - **Gerenciador de Adaptação**: componente de software responsável

por executar as Lógicas de Adaptação, definir e coletar métricas, e descobrir dados de recursos para registrá-los no framework (Discovery).

    * **Processador de Eventos**: Recepciona os eventos e executa um pré-processamento de adaptação (extração de parâmetros, métricas do evento, dados de ambiente, etc).
    * **Coordenador/Orquestrador**: Verifica se existe algum plano de execução ativo, aplicável ao componente e dispositivo informado no evento. No caso de mais de um plano disponível, determina qual a prioridade de execução desses planos. Também determina o momento para acionamento do Executor de plano de execução.
    * **Executor**: Executa o plano de adaptação e aplica as alterações no sistema;

– **Monitor do Ambiente**: componente de software responsável por coletar e calcular métricas de qualidade de serviço (QoS), além de descobrir implementações de lógica de adaptação e recursos para registrá-los no framework (Discovery).

    * **Receptor de métricas**: recebe as métricas coletadas pelos agentes;
    * **Processador de Métricas**: efetua cálculos para obtenção de métricas derivadas;
    * **Descoberta**: tem como função localizar e registrar, na base de conhecimento, dados de componentes, dispositivos e lógicas de implementação;

– **Ciclo MAPE**: Responsável pelo Gerenciamento do Ciclo de Adaptação do Framework; contém componentes para recepção dos eventos de monitoramento, análise de dados para determinação sobre quais táticas são aplicáveis, planejamento da execução do plano de adaptação e encaminhamento para execução.

    * **Analisador**: Tem como função analisar as métricas e dados de recursos gerenciados registrados no framework; este componente, no entanto, é composto por interfaces e classes abstratas; a tarefa de análise é delegada para uma implementação concreta de heurística.

    * **Planejador**: Planeja a execução do plano de implantação recebido do analisador, verifica qual agente executará o plano, bem como a ordem das alterações (táticas), e encaminha para execução.
    * **Publicador**: Publica o plano para execução, libera o plano para execução pelo Gerenciador de Adaptação através de um dos canais de comunicação.

- **Heurísticas**: Contém as implementações concretas de heurísticas.
    * **Rule Based Engine**: Implementação concreta de heurística de análise baseada em regras; contém uma base de regras e uma base de fatos para análise dos eventos.
    * **Demais heurísticas (Rede Neural e Composição de Processos)**: São algumas sugestões que aderem algum tipo necessidade de auto-propriedade, como autoimunidade, ou mesmo alguma arquitetura específica de sistemas de software. Não é escopo deste trabalho a implementação destas duas sugestões de heurísticas.
- **Base de Conhecimento**: Contém os repositórios com dados do sistema alvo. Este repositório mantém dados atualizados de dispositivos, componentes de software, qualidade de serviço, lógica de adaptação, identidades e permissões, além de parametrizações.
    * **Repositório de Identidades**: Mantém dados sobre identidades e permissões de elementos que precisam se autenticar para acessar recursos do framework.
    * **Meta-Informação de Dispositivos e Componentes**: Contém informações sobre todos os elementos de software (Componentes) que integram o sistema alvo, respectivos relacionamentos com dispositivos (instalações). Também contém informações sobre dispositivos, tais como: servidores, equipamentos de rede, instrumentos, etc.
    * **Qualidade de Serviço (QoS)**: Mantém informação referente a métricas, coletas, atributos de qualidade de serviço e relacionamentos com os componentes e dispositivos do sistema alvo, bem como relacionamento com os elementos do framework, tais como condições de disparo de táticas e lógicas de adaptação.
    * **Configuração e Meta-Informação de Lógica de Adapta-**

ção: Contém meta-informação da lógica de adaptação, incluindo regras para a execução da lógica de adaptação no sistema alvo. Também possui informação do framework, relacionada a heurísticas implementadas, transações, registros de eventos e canais de comunicação.

  - **Componentes de Segurança**: Conjunto de componentes responsáveis por autenticar usuários, dispositivos e componentes; autorizar acesso a recursos do framework; gerenciamento de chaves e/ou tokens, e certificados para protocolos de segurança.
  - **Componentes de Gerenciamento**: Conjunto de componentes responsáveis por manter/alterar dados de dispositivos, componentes, parâmetros de adaptação e qualidade de serviço.
  - **Canais de Comunicação**: Componentes de software que estabelecem a comunicação dos elementos do sistema solo com o framework. Para protocolos baseados no modelo publish/subscriber, um Broker assume este papel. Outros protocolos podem demandar um pondo de comunicação denominado Gateway, para a mesma finalidade. É importante que o framework forneça um mecanismo de extensão para permitir a inclusão de canais de comunicação, conforme cada necessidade de missão.

- **Aplicações**: Esta última camada é responsável por disponibilizar funções do framework para a utilização por aplicações externas.

5.3 Papéis do Arctic Fox

Esta seção visa descrever os papéis responsáveis pelas atividades de manutenção e evolução dos elementos do *framework* Arctic Fox.

São os papéis do *framework*:

- **Desenvolvedor de Heurística:** profissional responsável pelo desenvolvimento de heurísticas. A heurística é responsável pelas análises dos dados de QoS e definição das ações para implementação da adaptação. Ela é empacotada em um motor de análise que pode ser integrado ao framework. Para o desenvolvimento de um motor de análise, o desenvolvedor deve ter conhecimento das necessidades de adaptação de cada tipo de sistema,

da tecnologia empregada no algoritmo de análise ou, ainda, conhecimento específico de um domínio de conhecimento, como confiabilidade.

- **Especialista de Adaptação:** Tem conhecimento sobre o sistema alvo e variações do ambiente que permitem reconfigurar a base de conhecimento de modo a ajustar o framework para novas necessidades de adaptação. Pode ser um ser humano ou um elemento de software.

- **Desenvolvedor do Framework:** Tem por finalidade continuar a evoluir o framework para novas necessidades, tais como suporte a novos protocolos, otimizações, e outras ações que permitam evoluir a solução.

- **Desenvolvedor de Lógica de Adaptação:** Tem por finalidade desenvolver táticas para adaptação de sistemas.

5.4 Visão Comportamental do Framework

Nesta seção, é apresentada uma visão comportamental, com as interações entre os componentes do *framework* Arctic Fox, para os quatro cenários de maior relevância na sua utilização, sendo eles: adaptação não invasiva, adaptação invasiva, desenvolvimento da lógica de adaptação e desenvolvimento de heurística.

5.4.1 Cenário de Adaptação Não Invasiva

O *framework* pode ser utilizado de forma não invasiva para prover suporte autoadaptativo de sistemas. Nesse cenário, os agentes de coleta para monitoramento do sistema e de execução, que aplicam as adaptação no sistema alvo, estão instalados fora dos nós que compõem o sistema solo. Os agentes devem, no entanto, ter visibilidade dos componentes, isto é, acesso a eles via algum protocolo de comunicação, como HTTP por exemplo.

No contexto da adaptação não invasiva, o *framework* não conhece os estados internos dos componentes. Dessa maneira, o framework possui "consciência" do contexto, que é o ambiente no qual o componente se insere. Os agentes não estão embarcados no código-fonte do componente e, dessa maneira, apenas algumas métricas, como tempo de resposta, podem ser obtidas para monitoramento.

Este cenário é aplicado quando não há acesso ao código-fonte de algum componente. Também é indicado quando o momento selecionado para adaptação é o de "adaptar a cada solicitação". Quando a adaptação ocorre a cada solicitação do serviço, o

framework analisa o ambiente e produz um plano de adaptação, que é aplicado a inúmeras solicitações. São necessários dois componentes instalados no nó que reside no *framework*: o "Gerenciador de Solicitações" e o "Orquestrador", associados ao agente Executor, para viabilizar o cenário.

Para ilustrar o cenário de adaptação não invasiva, é apresentado, na figura 5.3, um exemplo de auto-otimização. Neste exemplo, uma aplicação cliente faz uma solicitação ao serviço 1 do sistema solo e o framework seleciona o serviço 2 para atender à solicitação. Esta decisão é a que indica a melhor opção para o desempenho do sistema, baseado nos dados monitorados.

Para materializar o cenário descrito no parágrafo anterior, são ilustrados dois processos na figura 5.3. No primeiro processo, responsável pela seleção do recurso, são executadas as atividades de monitoramento, análise e planejamento, que podem ser observadas nos círculos numerados na cor amarela. O resultado da execução do primeiro processo é um plano de adaptação pronto para execução. O segundo processo é o de execução da adaptação cujos passos também podem ser verificados nos círculos numerados na cor cinza. Este processo é o responsável pela execução do plano de adaptação.

Descrição dos passos do processo de monitoramento, análise e planejamento, no qual é produzido o plano de adaptação:

1) ArcticFox QoS: Métricas são coletadas dos componentes do Sistema Solo pelo "Agente Coletor".

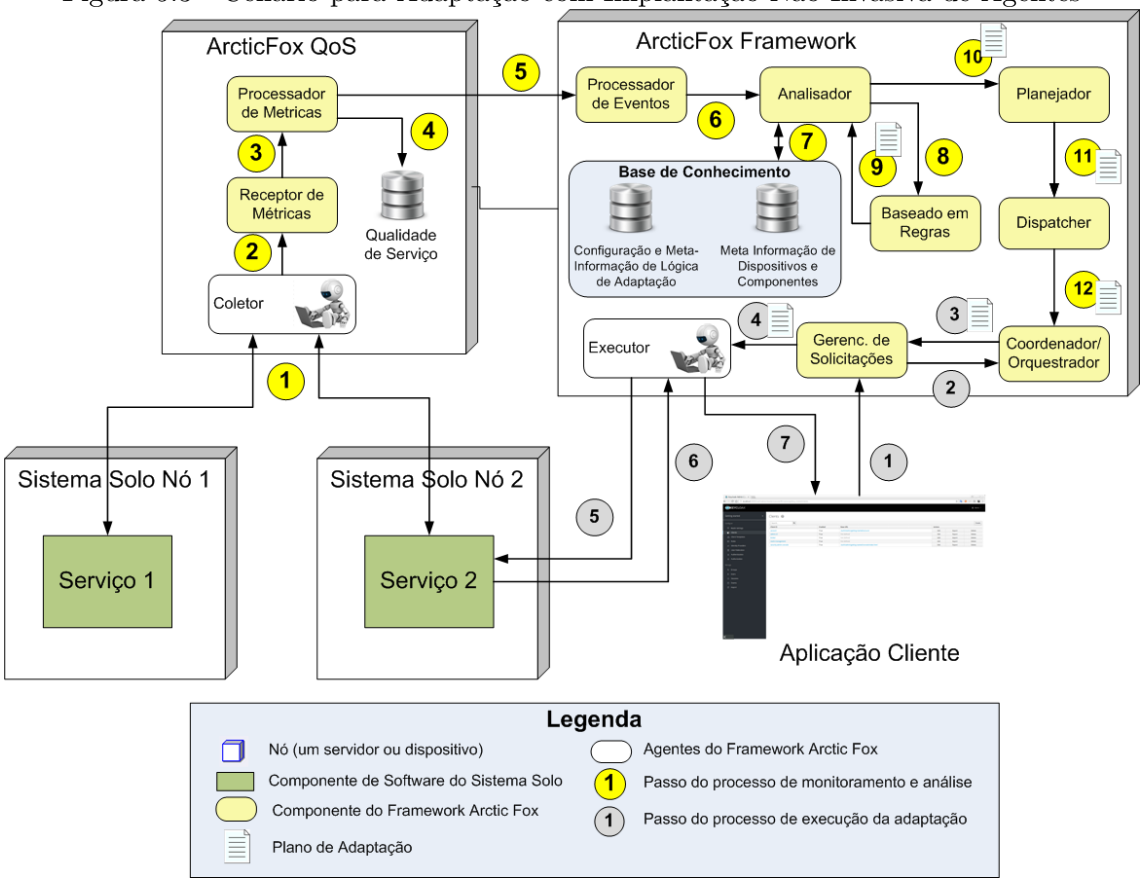
2) ArcticFox QoS: O "Agente Coletor" encaminha as métricas para o "Receptor de Métricas".

3) ArcticFox QoS: O "Receptor de Métricas" encaminha as métricas para o "Processador de Métricas", que agrega os dados e efetua cálculos de métricas derivadas.

4) ArcticFox QoS: O "Processador de Métricas" persiste a métrica na base de dados de qualidade de serviço.

5) ArcticFox Framework: O "Processador de Métricas" encaminha a métrica para o "Processador de Eventos", que avalia se aquela métrica, para aquele componente, é elegível para adaptação.

Figura 5.3 - Cenário para Adaptação com Implantação Não Invasiva de Agentes



Fonte: Autor

6) ArcticFox Framework: O “Processador de Eventos”, após a verificação, inicia o processo de análise pelo “Analisador”.

7) ArcticFox Framework: O “Analisador” consulta a base de dados para obter os critérios de adaptação definidos para a métrica e o componente informados.

8) ArcticFox Framework: O “Analisador” seleciona a heurística apropriada e inicia o processo de análise.

9) ArcticFox Framework: A “Heurística Baseada em Regras” produz um plano de adaptação com as alterações necessárias para execução e retorna o plano para o “Analisador”.

10) ArcticFox Framework: O “Analisador” encaminha o plano para o “Planejador”, que determina a ordem e a sequência de execução das táticas, além de selecionar qual a melhor forma de encaminhar o plano para execução.

11) ArcticFox Framework: O “Planejador” cria uma instância do “Dispatcher” e encaminha o plano para envio.

12) ArcticFox Framework: O “Dispatcher” encaminha o plano para o “Orquestrador”.

Descrição dos passos do processo de execução da adaptação:

1) A “Aplicação Cliente” efetua uma solicitação de serviço para o “Gerenciador de Solicitações”.

2) O “Gerenciador de Solicitações” aciona o “Orquestrador” e verifica se há um plano de adaptação disponível para execução.

3) O “Orquestrador” verifica se existe um plano de adaptação ativo e o encaminha para o “Gerenciador de Solicitações”.

4) O “Gerenciador de Solicitações” encaminha o plano para execução pelo “Executor”.

5) O “Executor” interpreta o plano e executa as ações que determinam a execução do Serviço 2 no nó 2 do sistema solo.

6) O “Executor” recebe o resultado da execução do Serviço.

7) O “Executor” encaminha o resultado para a “Aplicação Cliente”.

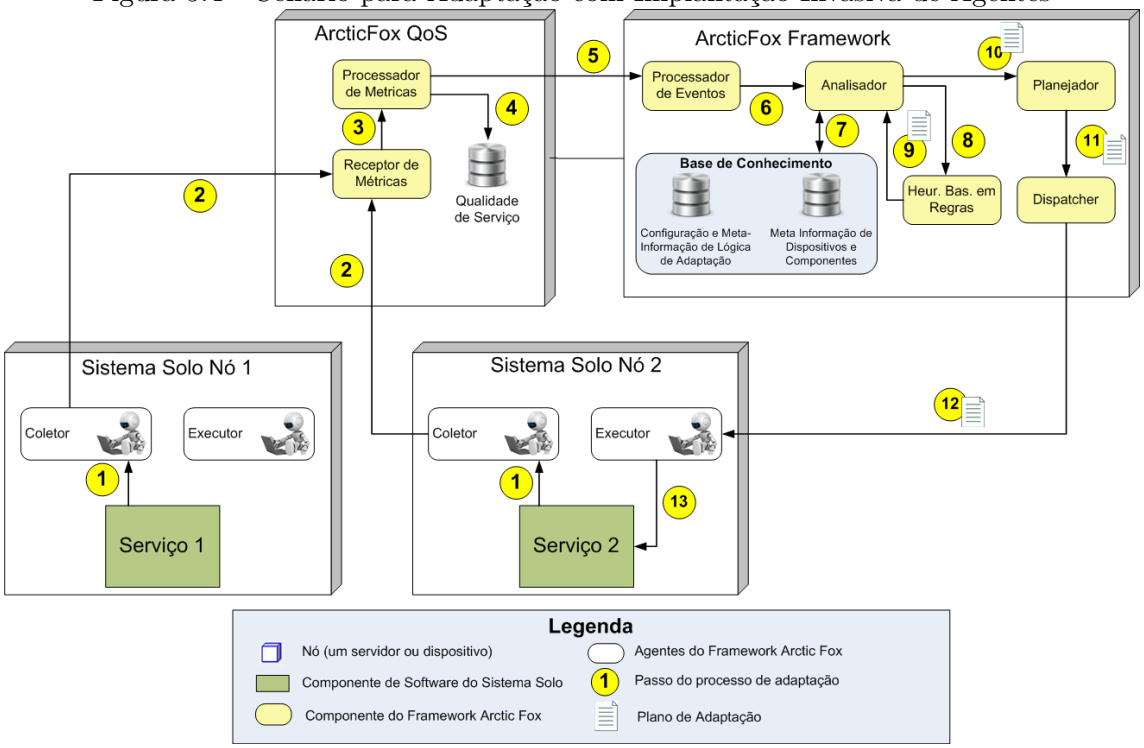
5.4.2 Cenário de Adaptação Invasiva

Neste cenário, os agentes de monitoramento e execução são embarcados junto ao código-fonte da aplicação, e permitem o conhecimento do estado interno do componente “auto-consciência”.

Em um cenário hipotético, onde uma aplicação cliente precisa reconfigurar parâmetros de recursos, o framework determina qual a parametrização a ser aplicada, baseada nos dados monitorados do sistema. O cenário é ilustrado na figura 5.4, onde as ações de adaptação são executadas diretamente pelo executor no nó do sistema solo.

Os passos do processo de adaptação, verificado nos círculos amarelos e numerados, são descritos na figura 5.4 a seguir:

Figura 5.4 - Cenário para Adaptação com Implantação Invasiva de Agentes



Fonte: Autor

1) ArcticFox QoS: Métricas são coletadas dos componentes do Sistema Solo pelo “Agente Coletor”.

2) ArcticFox QoS: O “Agente Coletor” encaminha as métricas para o “Receptor de Métricas”.

3) ArcticFox QoS: O “Receptor de Métricas”, encaminha as métricas para o “Processador de Métricas” que agrega os dados e efetua cálculos de métricas derivadas.

4) ArcticFox QoS: O “Processador de Métricas” persiste a métrica na base de dados de qualidade de serviço.

5) ArcticFox Framework: O “Processador de Métricas” encaminha a métrica para o “Processador de Eventos”, que avalia se aquela métrica, para aquele componente, é elegível para adaptação.

6) ArcticFox Framework: O “Processador de Eventos”, após a verificação, inicia o processo de análise pelo “Analisador”.

7) ArcticFox Framework: O “Analisador” consulta a base de dados para obter os critérios de adaptação definidos para a métrica e o componente informados.

8) ArcticFox Framework: O “Analisador” seleciona a heurística apropriada e inicia o processo de análise.

9) ArcticFox Framework: A “Heurística Baseada em Regras” produz um plano de adaptação com as alterações necessárias para execução e retorna o plano para o “Analisador”.

10) ArcticFox Framework: O “Analisador” encaminha o plano para o “Planejador”, que determina a ordem e a sequência de execução das táticas, além de selecionar qual a melhor forma de encaminhar o plano para execução.

11) ArcticFox Framework: O “Planejador” cria uma instância do “Dispatcher” e encaminha o plano para envio.

12) ArcticFox Framework: O “Dispatcher” encaminha o plano para o “Executor”.

13) ArcticFox Framework: O “Executor” efetiva as adaptações no componente do sistema Solo.

5.4.3 Cenário de Desenvolvimento de Lógica de Adaptação

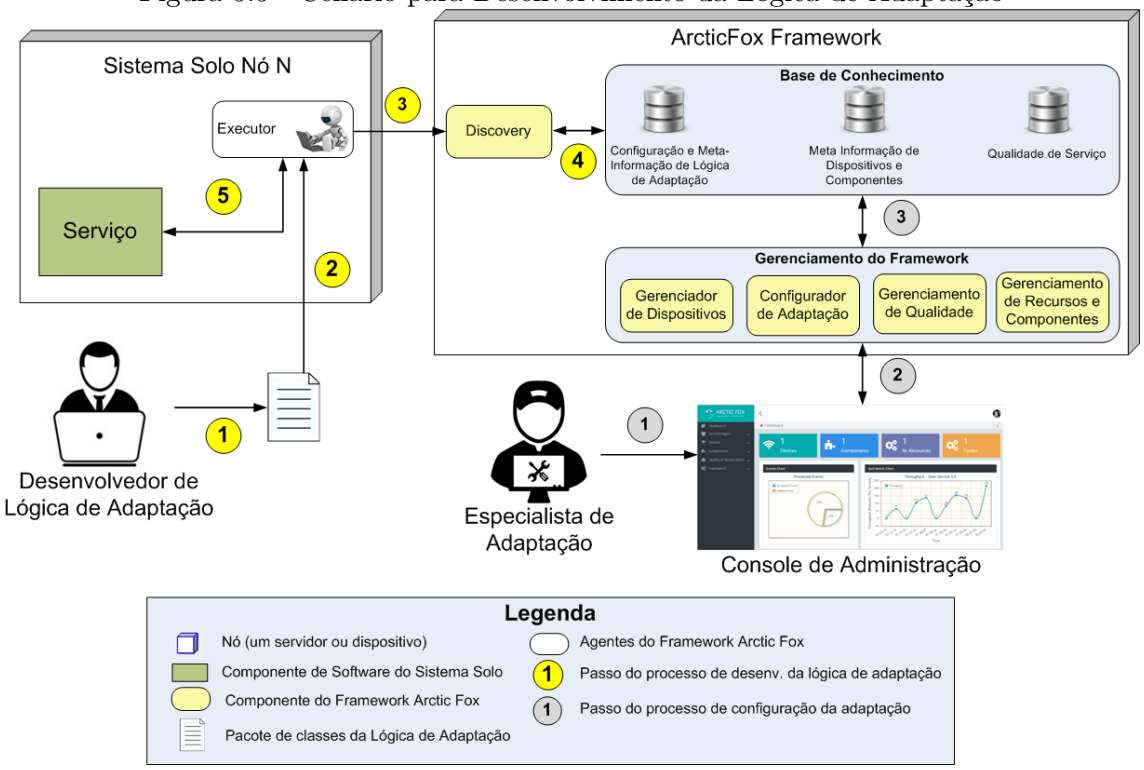
O *framework* Arctic Fox tem como característica o desacoplamento da Lógica de Adaptação da meta-informação dos componentes e instalação. Esta abordagem permite que lógicas de adaptação sejam implantadas em diversos nós do sistema solo, e que a meta-informação relacionada à localização da lógica de adaptação e condições de disparo sejam sincronizadas por meio de um mecanismo de descoberta “Discovery”.

O desacoplamento também permite que as configurações de meta-informação, como as condições de disparo, sejam alteradas em tempo de execução, possibilitando, assim a adaptação *on-the-fly*.

Na figura 5.5, é apresentado o cenário de desenvolvimento da lógica de adaptação.

O cenário de desenvolvimento é composto de dois processos, sendo o primeiro o de desenvolvimento da lógica, e o segundo o de configuração da base de conhecimento

Figura 5.5 - Cenário para Desenvolvimento da Lógica de Adaptação



Fonte: Autor

do framework para alteração dos critérios de adaptação. O processos são descritos a seguir:

Processo de desenvolvimento da lógica de adaptação:

1) O “Desenvolvedor da Lógica de Adaptação” programa as táticas de adaptação com o auxílio de uma API fornecida pelo *framework*.

2) Após testes unitários, a Lógica de Adaptação é implantada em um “Agente Executor”, em determinado nó, que pode ser do framework (não invasivo) ou do sistema solo (invasivo).

3) O “Executor” detecta que uma nova lógica de adaptação foi implantada e informa a meta-informação da lógica para o “Discovery”.

4) O “Discovery” sincroniza a informação na Base de Conhecimento.

5) O “Executor” está pronto para executar a lógica de adaptação mediante ordem do *framework* Arctic Fox.

Processo de configuração dos critérios de adaptação na base de conhecimento:

1) O "Especialista de Adaptação" pode alterar os critérios de disparo da lógica de adaptação na console de administração do *framework*.

2) Os componentes de "Gerenciamento do Framework" validam a informação e sincronizam a nova parametrização na base de conhecimento.

3) O *framework* passa a tomar decisões baseados na nova parametrização em tempo de execução.

5.4.4 Cenário de Desenvolvimento de Heurística

O framework provê suporte ao desenvolvimento de diversas heurísticas, o que permite explorar o espectro de abordagens de adaptação citados por Oreizy et al. (1999). O processo para desenvolvimento é apresentado na figura 5.6.

O cenário de desenvolvimento é composto de dois processos, sendo o primeiro o de desenvolvimento da heurística, e o segundo o de configuração da base de conhecimento do framework para alteração dos critérios de adaptação. O processos são descritos a seguir:

Processo de desenvolvimento da Heurística:

1) O "Desenvolvedor de Heurística" programa a heurística com o auxílio de uma API fornecida pelo *framework*.

2) Após testes unitários, a heurística é implantada no *framework*.

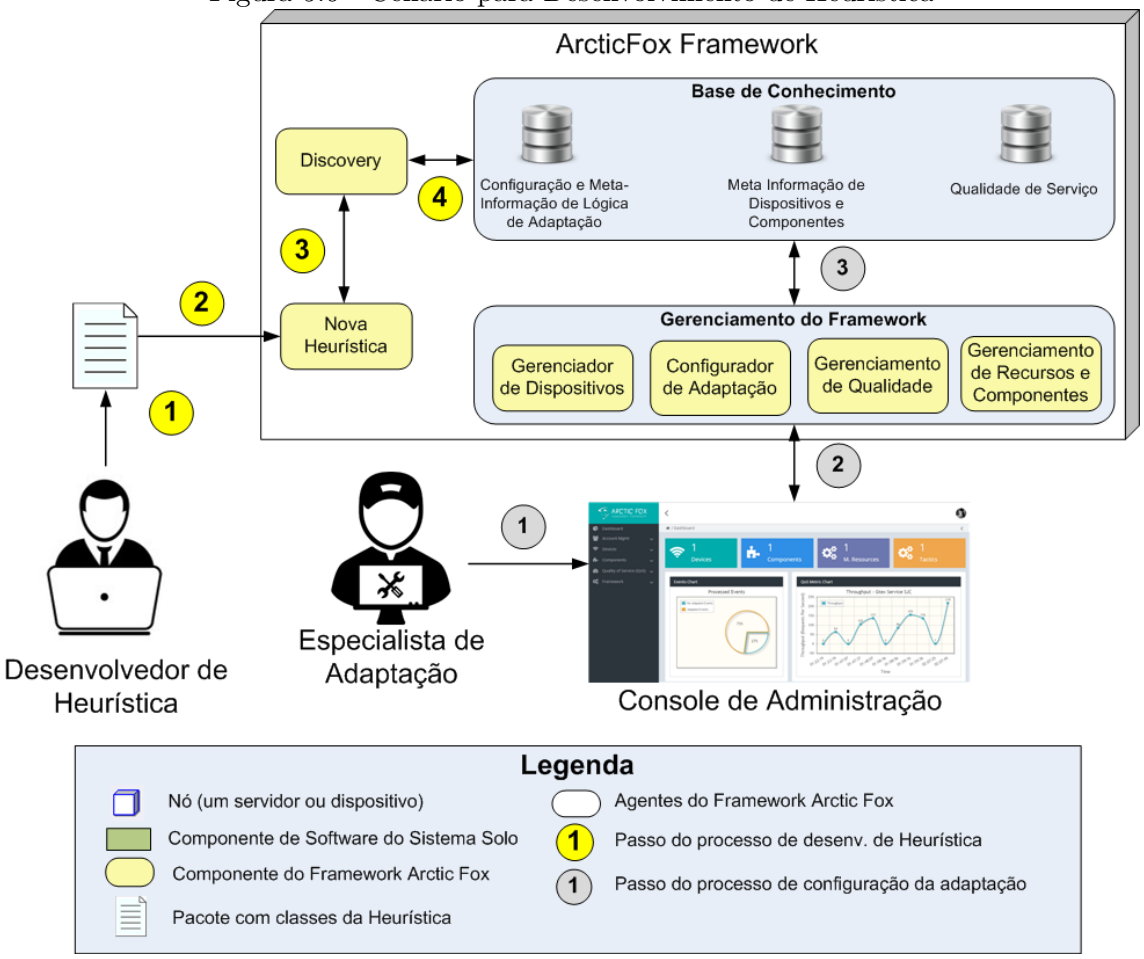
3) O "Discovery" detecta a implantação da nova heurística.

4) O "Discovery" sincroniza a informação na Base de Conhecimento.

Processo de configuração dos critérios de adaptação na base de conhecimento:

1) O "Especialista de Adaptação" pode, de maneira opcional, alterar os critérios de adaptação para utilizar a nova heurística implantada no framework. Esta configuração pode ser aplicada para que alguns componentes ou critérios de disparo de lógica de adaptação passem a ser analisados pela nova heurística.

Figura 5.6 - Cenário para Desenvolvimento de Heurística



Fonte: Autor

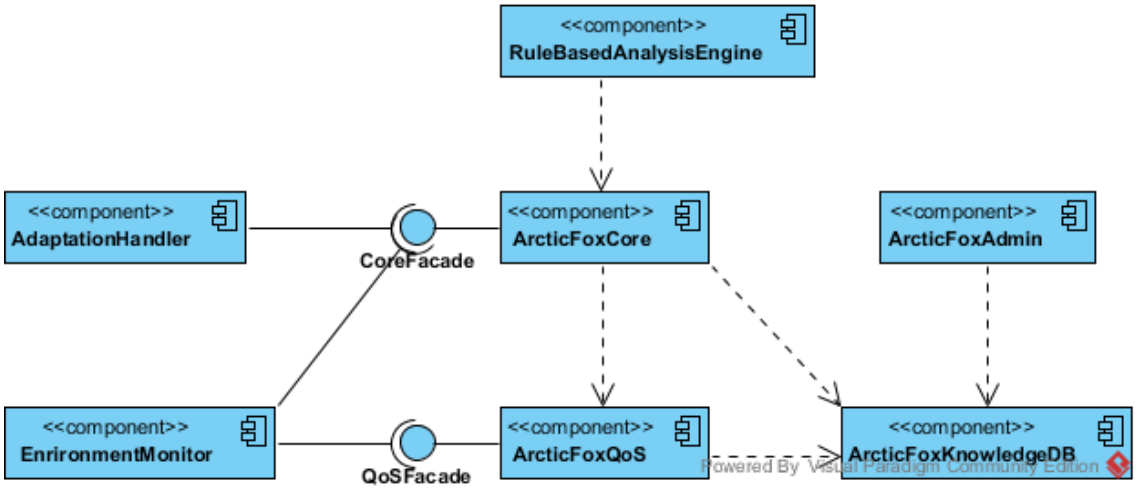
2) Os componentes de "Gerenciamento do Framework" validam a informação e sincronizam a nova parametrização na base de conhecimento.

3) O *framework* passa a tomar decisões baseadas na nova parametrização em tempo de execução.

5.5 Visão Geral da Implementação do Protótipo do Framework

Nesta seção, é apresentada uma visão de arquitetura de software, que implementa os componentes funcionais apresentados na seção 5.2. A visão em questão apresenta como as funções foram alocadas para cada componente, dependências e respectiva descrição. Na figura 5.7, é apresentado um diagrama de componentes UML, com os componentes de software e relações de dependência entre eles.

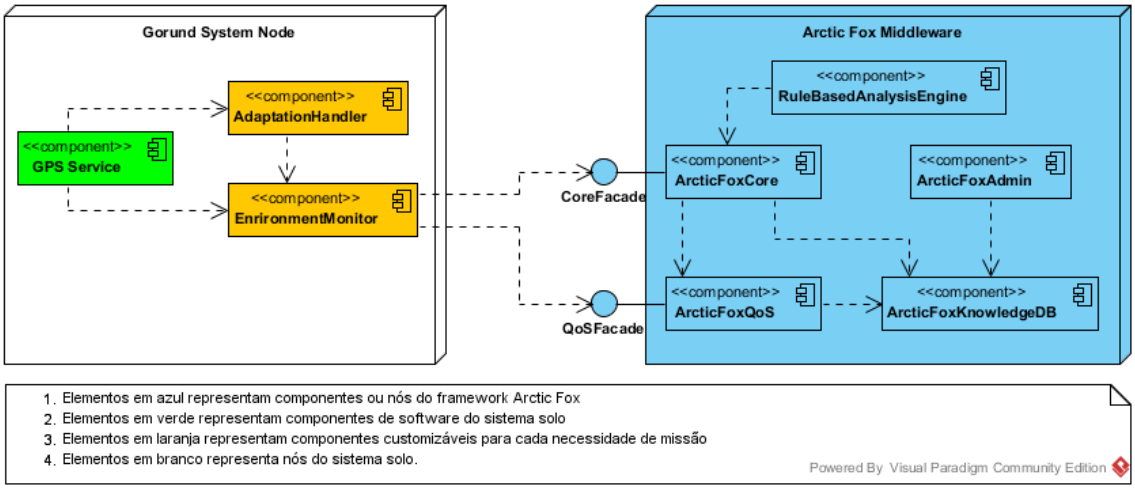
Figura 5.7 - Diagrama de Componentes Framework Arctic Fox



Fonte: Autor

Na figura 5.8, é apresentado um diagrama de implantação UML com a distribuição dos componentes de software em nós, considerando um cenário fictício de aplicação da solução para uma aplicação de clima espacial.

Figura 5.8 - Diagrama de Implantação Componentes Arctic Fox



Fonte: Autor

A tabela 5.3 apresenta os componentes de software do framework ArcticFox e respectiva alocação de grupos funcionais.

Tabela 5.3 - Componentes de Software do Framework Arctic Fox

Componente	**Descrição**	**Funções Alocadas**
ArcticFoxQoS	Responsável por gerenciar e manter informação de Qualidade de Serviço	Receptor Processador de Métricas Processador de Eventos
ArcticFoxCore	Gerenciador MAPE do framework, analisa, planeja e envia planos de adaptação, além de notificar motores de análise sobre alterações na base de conhecimento	Processador de Eventos Analisador Planejador Dispatcher
ArcticFoxKnowledgeDB	Contém os repositórios do framework	Meta-Informação de Componentes e Dispositivos Qualidade de Serviço Configuração e Meta-Informação de Lógica de Adaptação Repositório de Identidades
RuleBasedAnalysisEngine	Analisa eventos de componentes e dispositivos de modo a verificar se foi violado/atingido algum objetivo de adaptação. A implementação em questão utiliza um motor de inferência com uma base de fatos e uma case de regras que pode ser alterado quando ocorrem atualizações na base de conhecimento.	RuleBasedAnalysisEngine
ArcticFoxAdmin	Responsável pelo gerenciamento da Base de Conhecimento.	Gerenciamento de Dispositivos Gerenciamento de Componentes Gerenciamento de Qualidade de Serviço Gerenciamento de Parâmetros de Adaptação
AdaptationManager	Responsável por prover facilidades para implementação de lógica de adaptação, execução de adaptação, gerenciamento de planos de adaptação e execuções.	Executor Coordenador/Orquestrador Gerenciador de Solicitações
EnvironmentMonitor	Responsável por coletar métricas, descobrir e registrar componentes e dispositivos	Coletor Discovery

Fonte: Autor

A implementação de um protótipo do framework aderente a esta arquitetura é apresentada no próximo capítulo. Cada componente é detalhado com um nível de granularidade mais baixo, além de exemplos de código, APIs e detalhes do comportamento do protótipo.

6 UM PROTÓTIPO DO FRAMEWORK

Neste capítulo, são apresentados os artefatos de projeto de um protótipo do *framework* Arctic Fox. O objetivo do protótipo é validar os conceitos apresentados no capítulo 5, além de aplicá-lo em um estudo de caso. São detalhados os seus componentes, incluindo base de conhecimento, ferramenta para administração, criação de heurísticas e desenvolvimento da lógica de adaptação.

6.1 Plataforma de Desenvolvimento

Para a implementação do protótipo do *framework*, foi selecionada a plataforma JEE (Java Enterprise Edition) (SHANNON et al., 2000), devido à sua versatilidade para a construção de aplicações seguras e escaláveis com agilidade, além de diversas APIs que permitem o acesso a bancos de dados, acesso a sistemas legados, mensageria, processamento síncrono e assíncrono, transações, entre outros.

Há também vasta documentação disponível, o que aumenta o número de profissionais habilitados na plataforma JEE, além de causar uma diminuição na curva de aprendizado.

Para a heurística do protótipo, foi selecionada a arquitetura baseada em regras. Sistemas baseados em regras têm como característica o rápido processamento das regras, além de permitirem a compilação de regras em tempo de execução. Para a implementação do motor de inferência da heurística, foi selecionado o *framework* Jboss Drools (JBOSS, 2017), que pode ser integrado a um Application Server JEE.

A base de conhecimento foi implementada por meio de um gerenciador de banco de dados relacional PostgreSQL.

A característica de adaptação *on-the-fly*, definida para o *framework*, pode ser obtida por meio de 3 decisões de projeto, que são, respectivamente: o desacoplamento da Meta-Informação da Implementação da Lógica de Adaptação, a capacidade de recompilação de regras em tempo de execução, e a utilização do padrão de projeto Type Square (YODER et al., 2001) para a criação de entidades, também em tempo de execução.

As decisões de projeto e a tecnologia selecionada permitem adaptação *on-the-fly*. Isto é possível através de três decisões de projeto para a implementação do framework sendo elas:

- A adoção do *Design Pattern Type Square*, que permite definições de tipos e condições em tempo de execução;
- A adoção da API de reflexão do Java (FLANAGAN, 2005), que permite o desacoplamento entre Meta-informação e a implementação da lógica de adaptação. Com a API de reflexão Java, foi possível a execução de métodos de classes a partir da utilização de strings. Esta decisão de projeto materializou a implementação dos agentes executores e dos planos de execução.
- A utilização de anotações Java (FLANAGAN, 2005), que permitem a descoberta de classes, atributos e métodos e, desta maneira, permitem extrair a meta-informação para armazenamento em banco de dados.

Esta combinação de decisões de projeto e seleção de tecnologias permite ao framework tomar ações de imediato, assim que alterações de diversas configurações de adaptação, implantações em tempo de execução e recompilação de regras, sejam efetivadas na base de conhecimento e heurísticas.

6.2 Core: Gerenciador do Ciclo de Vida de Adaptação

O Gerenciador MAPE embarcado no framework ArcticFox é acionado pelo ArcticFoxQoS por meio do envio de eventos. O processo é disparado após a coleta e cálculos de métricas de qualidade de serviço dos recursos gerenciados do sistema solo.

O Gerenciador MAPE do ArcticFox framework foi modelado como uma máquina de estados que é responsável por duas das quatro etapas do modelo MAPE-K, isto quer dizer que, no âmbito do ciclo de vida de adaptação, o gerenciador é responsável por:

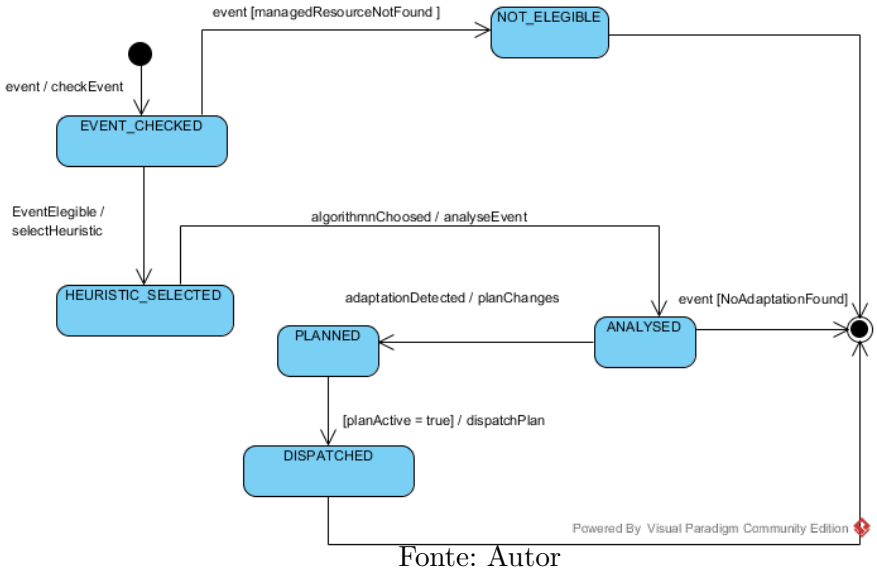
- Analisar os eventos gerados pelo ArcticFoxQoS por meio de uma heurística;
- Planejar a execução do plano de adaptação e encaminhá-lo para um agente de software.

Na figura 6.1, é apresentada a máquina de estados modelada como um diagrama de estados UML.

A descrição dos estados apresentados na figura 6.1 é fornecida na tabela 6.1.

Para a implementação da máquina de estados, foi selecionado o padrão de projeto *State Pattern* (GAMMA et al., 1994), que determina como deve ser implementado

Figura 6.1 - Máquina de Estados do Gerenciador Autônomo



Fonte: Autor

cada estado e controle das transições previstas na máquina . A figura 6.2 apresenta um diagrama de classes UML, que representa a estrutura da máquina de estados que utiliza o padrão de projeto mencionado.

Para produzir um plano de adaptação, o gerenciador autônomo do arcticfox executa uma máquina de estados, que realiza algumas etapas para gerar o plano de adaptação e, então, concluir a execução da máquina: a primeira etapa é a avaliação de elegibilidade, que verifica se os critérios de adaptação para a o componente e métricas de QoS estão previamente configurados na base de conhecimento; nessa etapa, ocorre uma transição do estado inicial (START) para o estado verificado (EVENT_-CHECKED). A segunda estapa do gerenciador autônomo do ArcticFox é selecionar a heurística adequada para análise; nesse passo, ocorre a transição do estado verificado (EVENT_CHECKED) para heurística selecionada (HEURISTIC_SELECTED). A terceira etapa é a execução da análise pela heurística selecionada e a geração de um plano de adaptação; nessa etapa, ocorre a transição do estado Algoritmo Selecionado (HEURISTIC_SELECTED) para Analisado (ANALYSED). A quarta etapa é o planejamento da execução do plano; a ordem das atividades pode ser alterada e o canal de encaminhamento é selecionado; nessa etapa, ocorre a transição de Analisado (ANALYSED) para Planejado (PLANNED). A quinta etapa é o encaminhamento do plano para execução por um agente de software; nessa etapa, ocorre a transição do estado Planejado (PLANNED) para o estado Encaminhado (DISPATCHED).

Tabela 6.1 - Descrição dos Estados do Gerenciador Autônomo

Estado	**Descrição**
START	Estado Inicial responsável por recepcionar eventos e verificar elegibilidade para adaptação.
EVENT_CHECKED	Caso o evento seja elegível para adaptação, o processo de seleção de heurística é acionado e o estado é transitado para HEURISTIC_SELECTED; do contrário, o estado é transitado para NOT_ELEGIBLE.
NOT_ELEGIBLE	Significa que o recurso gerenciado não está apto para adaptação, seja por inativação do recurso, ou alteração de dados do ambiente na Base de Conhecimento.
HEURISTIC_SELECTED	Aciona o analisador selecionado para geração do plano de adaptação; caso o plano selecione táticas para execução, o estado é transitado para ANALYSED; do contrário, o estado é transitado para STOP.
ANALYSED	Inicia processo de planejamento, onde é verificado qual o melhor formato para envio do plano, ordenação de táticas, sequência de execução, escolha de protocolo.
PLANNED	Encaminhamento do plano de adaptação para execução.
DISPATCHED	Plano de adaptação encaminhado para execução pelo agente Executor.
STOP	Finalização da máquina de estados.

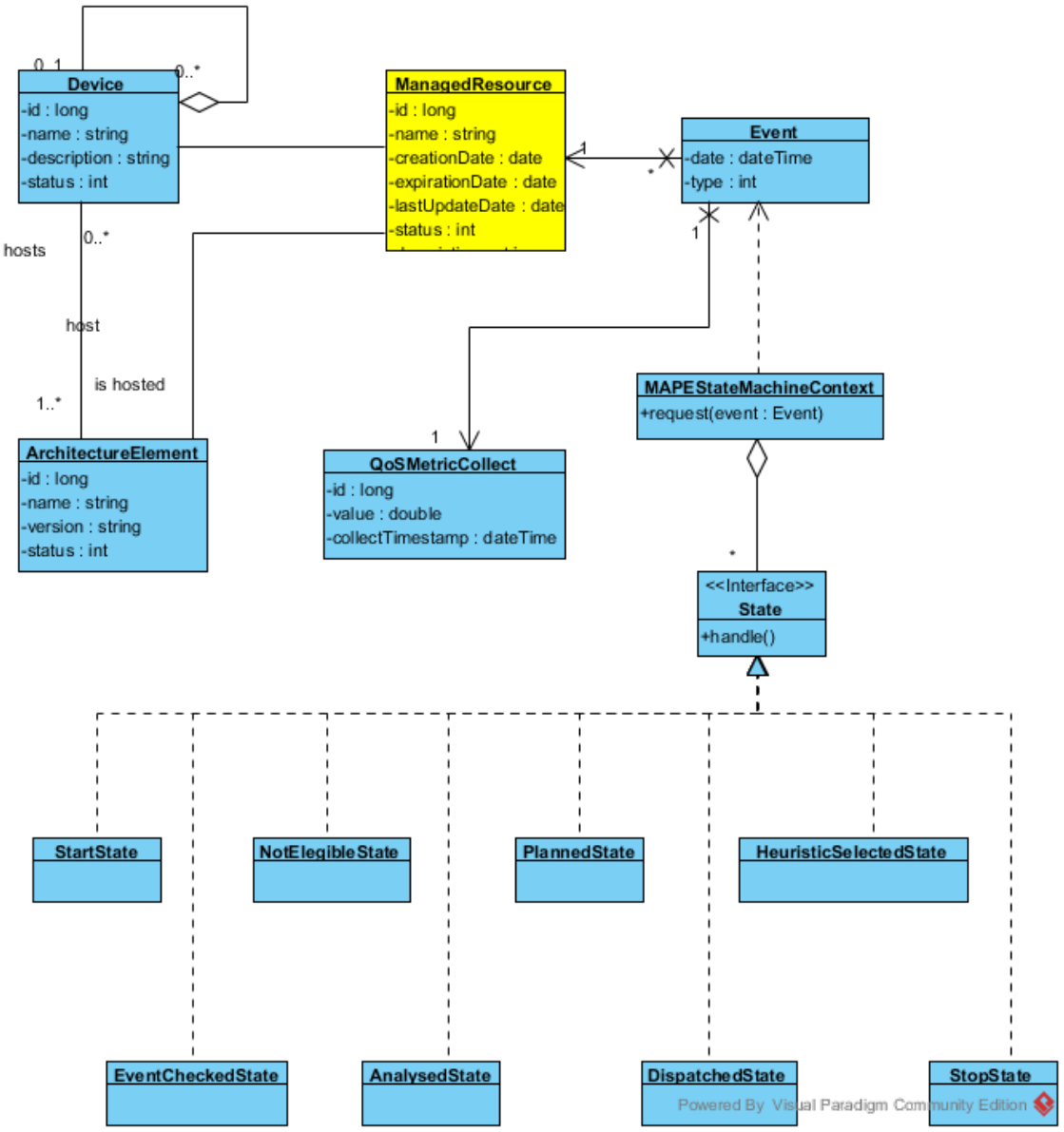
Fonte: Autor

Para ilustrar o processo de geração de um plano de adaptação, é apresentado, na figura 6.3, o log de execução do gerenciador autônomo, onde as etapas descritas no parágrafo anterior são destacadas e numeradas com círculos na cor amarela. O número quatro (4) da figura ilustra as duas últimas transições de estados, Analisado para Planejado e Planejado para Encaminhado.

6.2.1 Desenvolvimento de Heurísticas

Esta seção visa apresentar os recursos de extensão do framework para implementação de heurísticas. Neste sentido, são apresentados detalhes do processo de seleção de heurística, facilidades de desenvolvimento de observadores para recepção de notificações sobre alterações na Base de Conhecimento.

Figura 6.2 - Diagrama de Classes do Gerenciador Autônomo



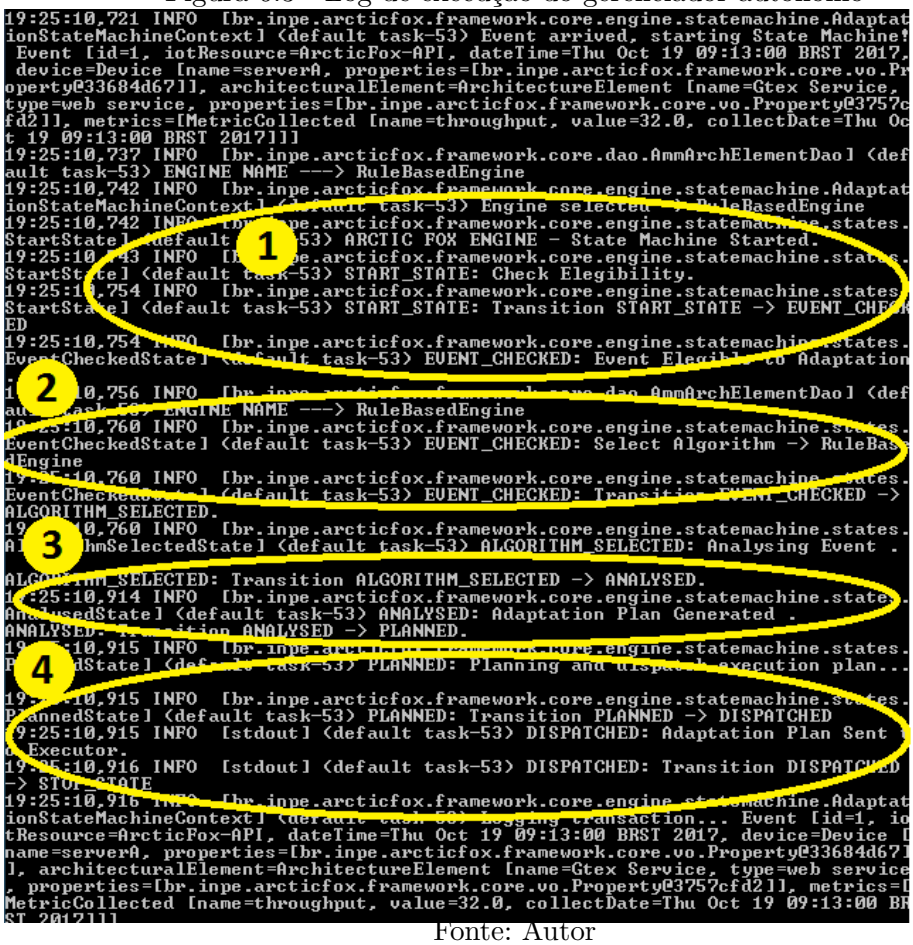
Fonte: Autor

6.2.2 Processo de Seleção de Heurísticas

É apresentado um recorte entre a transição dos estados EVENT_CHECKED e HEURISTIC_SELECTED, onde o processo é executado. A figura 6.4 mostra um diagrama de atividades UML que modela o comportamento do processo.

As classes que colaboram para o processo de seleção são apresentadas em um diagrama UML, que pode ser visto na figura 6.5.

Figura 6.3 - Log de execução do gerenciador autônomo



Fonte: Autor

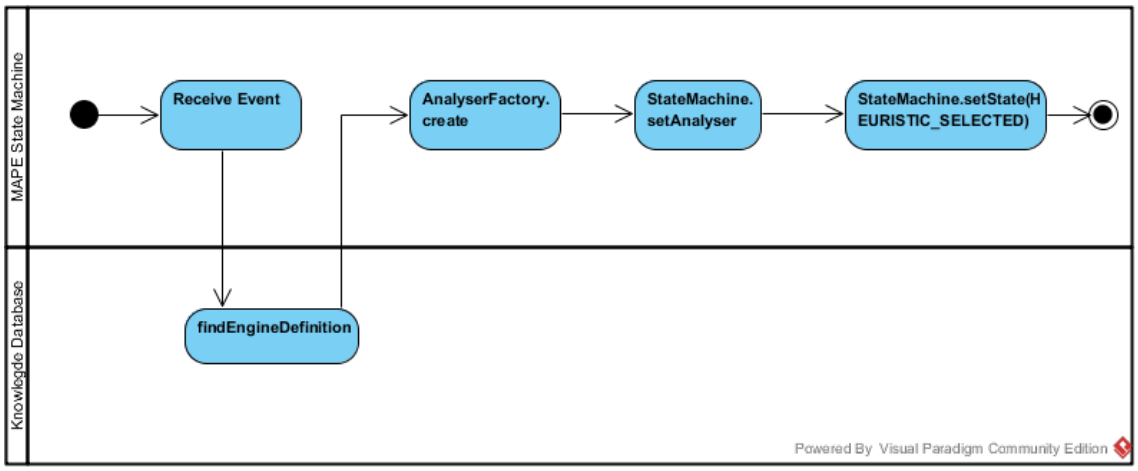
6.2.3 Pacotes de Classes, Interfaces e Anotações

O framework fornece um conjunto de classes, interfaces e anotações que permitem o desenvolvimento de Heurísticas, as quais podem ser registradas e incorporadas no framework. O elemento central pelo qual a heurística é implementada é o Analisador.

O Analisador é responsável por receber os eventos, acessar a Base de Conhecimento, analisar dados de Qualidade de Serviço e determinar se há um conjunto de táticas que podem ser executadas no sistema alvo.

Na tabela 6.2, é apresentada a descrição destes elementos fornecidos pelo mecanismo de extensão do framework Artic Fox.

Figura 6.4 - Processo de Seleção de Heurística



Fonte: Autor

Tabela 6.2 - Descrição dos Elementos do Mecanismo de Extensão do Framework

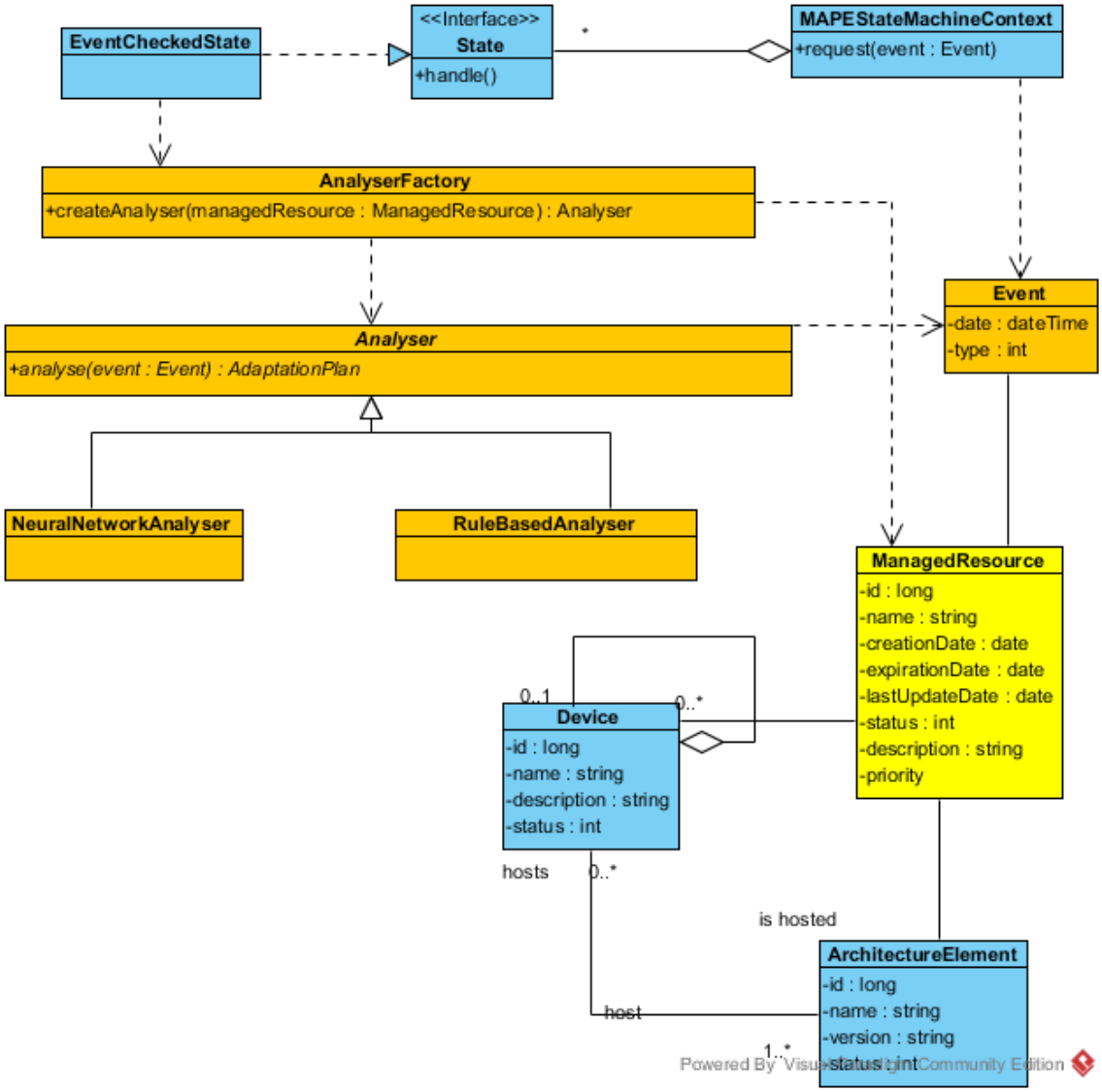
Elemento	**Tipo**	**Descrição**
Analyser	Anotação	Elemento pelo qual o Framework descobre implementações e registra na Base de Conhecimento.
Listener	Anotação	Informa uma implementação de um Observador para o Framework. O framework notifica observadores cada vez que a Base de Conhecimento é alterada.
AdaptationAnalyser	Interface	A implementação da heurística se dá por meio da criação de uma classe Java que implementa a interface AdaptationAnalyser.
ManagedResourceListener	Interface	Interface para desenvolvimento de Observadores, são notificados eventos de criação, exclusão e atualização de recursos.
Event	Classe	Representa um evento encaminhado pelo ArcticFoxQoS.
ManagedResource	Classe	Recurso Gerenciado pelo Framework.
Device	Classe	Representa um dispositivo como um servidor ou instrumento por exemplo.
ArchitectureElement	Classe	Elemento de software como um componente ou conector.
QoSMetricCollected	Classe	Representa um valor coletado ou cálculo de métrica de Qualidade de Serviço.
Tactic	Classe	Implementação de uma determinada lógica de adaptação.

Fonte: Autor

6.2.4 Implementação do Analisador

O Analisador é acionado pelo framework em tempo de execução. O framework toma a decisão sobre quando acionar determinada implementação baseada em parâmetros

Figura 6.5 - Diagrama de Classes do Processo de Seleção de Heurística



Fonte: Autor

configurados na base de conhecimento. O processo de desenvolvimento dos analisadores é relativamente simples e compreende os seguintes passos:

a) Criar um projeto java.

b) Criar uma classe java que implemente a interface AdaptationAnalyser.

c) Anotar a classe com a anotação @Analyser.

d) Empacotar as classes do projeto.

e) Instalar o pacote no Servidor de Aplicações em que está instalado o framework.

f) O framework reconhecerá o pacote (discovery) e irá procurar por classes anotadas com @Analyser; após encontradas essas classes, elas serão registradas na base de conhecimento para acionamento futuro.

g) O Analisador está pronto para uso.

Na figura 6.6, é apresentada a implementação em linguagem java do analisador padrão do framework. A implementação em questão utiliza uma estratégia baseada em regras e contém uma base de fatos para processamento dos eventos e uma base de regras que confronta os eventos com as regras para seleção de táticas dos elementos gerenciados.

A implementação em questão utiliza o framework Jboss Drools (JBOSS, 2017), que provê os mecanismos para criação do motor de inferência com as das bases de fatos e regras.

Figura 6.6 - Implementação do Analisador em Java

RuleBasedAnalyser.java

```
1  package br.inpe.arcticfox.engine.rulebased.analyser;
2
3  import org.kie.api.KieServices;
4  import org.kie.api.runtime.KieContainer;
5  import org.kie.api.runtime.KieSession;
6
7  import br.inpe.arcticfox.framework.core.engine.annotation.Analyser;
8  import br.inpe.arcticfox.framework.core.engine.interfaces.AdaptationAnalyser;
9  import br.inpe.arcticfox.framework.core.vo.AdaptationPlan;
10 import br.inpe.arcticfox.framework.core.vo.Event;
11
12 @Analyser(engineName="RuleBasedEngine")
13 public class RuleBasedAnalyser implements AdaptationAnalyser {
14
15     public AdaptationPlan analyse(Event event) {
16         KieServices kieServices = KieServices.Factory.get();
17         KieContainer kieContainer = kieServices
18             .newKieContainer(kieServices.getRepository().getDefaultReleaseId());
19         KieSession kSession = kieContainer.getKieBase().newKieSession();
20
21         AdaptationPlan plan = new AdaptationPlan();
22
23         kSession.insert(event);
24         kSession.insert(plan);
25         kSession.fireAllRules();
26
27         return plan;
28     }
29
30 }
31
```

Fonte: Autor

Na figura 6.7, são apresentados os logs do processo de descoberta de heurísticas do framework Arctic Fox. A área circulada na cor amarela destaca a descoberta de uma implementação de um analisador. O exemplo destacado é a heurística padrão do framework.

Figura 6.7 - Logs do processo de Descoberta de Heurísticas

Fonte: Autor

6.2.5 Implementação do Observador

O Observador é uma implementação opcional, utilizada para receber notificações do framework cada vez que ocorrem alterações na Base de Conhecimento. A implementação deste mecanismo é baseado no padrão de projeto *Observer Pattern* (GAMMA et al., 1994).

Em muitos casos, pode ser necessário sincronizar parâmetros utilizados pelo algoritmo implementado no Analisador com a base de conhecimento, dadas as mudanças ocorridas no ambiente.

Para solucionar esse problema, há a possibilidade de o Framework notificar o Motor de Análise por meio do desenvolvimento de uma classe java que implemente a interface AdaptationLogicListener.

A classe também precisa ser anotada com @Listener, anotação esta que permite ao framework registrar o observador e incluí-lo na lista de observadores para notificação quando ocorrer alguma alteração na base de conhecimento.

A implementação do observador pode ser verificada na figura 6.8.

Figura 6.8 - Implementação em linguagem Java do Observador

RuleBasedEngineListener.java

```
1  package br.inpe.arcticfox.engine.rulebased.listener;
2
3  import br.inpe.arcticfox.engine.rulebased.container.rulebuilder.RuleContainerBuilder;
4  import br.inpe.arcticfox.framework.core.engine.annotation.Listener;
5  import br.inpe.arcticfox.framework.core.engine.interfaces.AdaptationLogicListener;
6  import br.inpe.arcticfox.framework.core.vo.AdaptationLogic;
7
8  @Listener(engineName="RuleBasedEngine")
9  public class RuleBasedEngineListener implements AdaptationLogicListener {
10
11     public void notifyCreated(AdaptationLogic mr) {
12         try {
13             RuleContainerBuilder.getInstance().addToRuleBase(mr);
14         } catch (Exception e) {
15             // TODO Auto-generated catch block
16             e.printStackTrace();
17         }
18     }
19
20     public void notifyUpdated(AdaptationLogic mr) {
21         try {
22             RuleContainerBuilder.getInstance().updateRuleBase(mr);
23         } catch (Exception e) {
24             // TODO Auto-generated catch block
25             e.printStackTrace();
26         }
27     }
28
29     public void notifyRemoved(AdaptationLogic mr) {
30         RuleContainerBuilder.getInstance().removeFromRuleBase(mr);
31     }
32
33 }
34
```

Fonte: Autor

6.2.6 Suporte a outros Canais de Comunicação

No contexto de sistemas de solo de aplicações espaciais, em alguns casos, pode ser necessário suporte a múltiplos protocolos de comunicação. A arquitetura do framework prevê um mecanismo de extensão para adição de capacidades de comunicação.

Este mecanismo contacta o framework em 2 estados da máquina MAPE, sendo o primeiro ponto de contato o estado de recepção de eventos, enquanto o segundo consiste no encaminhamento do plano de execução para um dado agente de software.

A versão inicial do framework provê suporte inicial para 2 protocolos, e pode recepcionar eventos e entregar planos por HTTP e MQTT. O mecanismo não obriga a comunicação pelo mesmo protocolo, o que significa que o evento pode ser recebido por HTTP, e que o plano de adaptação pode ser entregue por MQTT dependendo da parametrização existente na Base de Conhecimento.

O mecanismo em questão possibilita a criação de uma classe java que implementa uma interface PlanDispatcher. O mecanismo foi baseado no padrão de projeto *Dispatcher* (ALUR et al., 2003). A implementação para o protocolo MQTT pode ser observada na figura 6.9.

Figura 6.9 - Dispatcher Java para Encaminhamento dos Planos de Adaptação

MqttDispatcher.java

```java
package br.inpe.arcticfox.framework.core.engine.dispatcher;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import br.inpe.arcticfox.framework.core.engine.interfaces.PlanDispatcher;
import br.inpe.arcticfox.framework.core.vo.AdaptationPlan;

public class MqttDispatcher implements PlanDispatcher {

    private static final Logger LOGGER = LoggerFactory.getLogger(MqttDispatcher.class);

    @Override
    public void dispatch(AdaptationPlan plan) {
        LOGGER.info("Dspatched to executors queues (MQTT)");
        Context ctx;
        try {
            ctx = new InitialContext();
            TopicConnectionFactory tConFactory = (TopicConnectionFactory) ctx.lookup("java:/ActiveMQConnectionFactory");
            Topic messageTopic = (Topic) ctx.lookup("java:jboss/activemq/topic/ArcticFoxAPITopic");
            TopicConnection tCon = tConFactory.createTopicConnection();
            TopicSession session = tCon.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
            TopicPublisher publisher = session.createPublisher(messageTopic);
            TextMessage msg = session.createTextMessage();

            msg.setText(plan.toString());
            publisher.publish(msg);
            publisher.close();
            tCon.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Fonte: Autor

6.3 Base de Conhecimento

A Base de Conhecimento é um componente de extrema importância na implementação do framework. Para a sua modelagem, foram utilizados alguns conceitos agrupados em quatro categorias, como apresentado no capítulo 5. Os agrupamentos são listados a seguir:

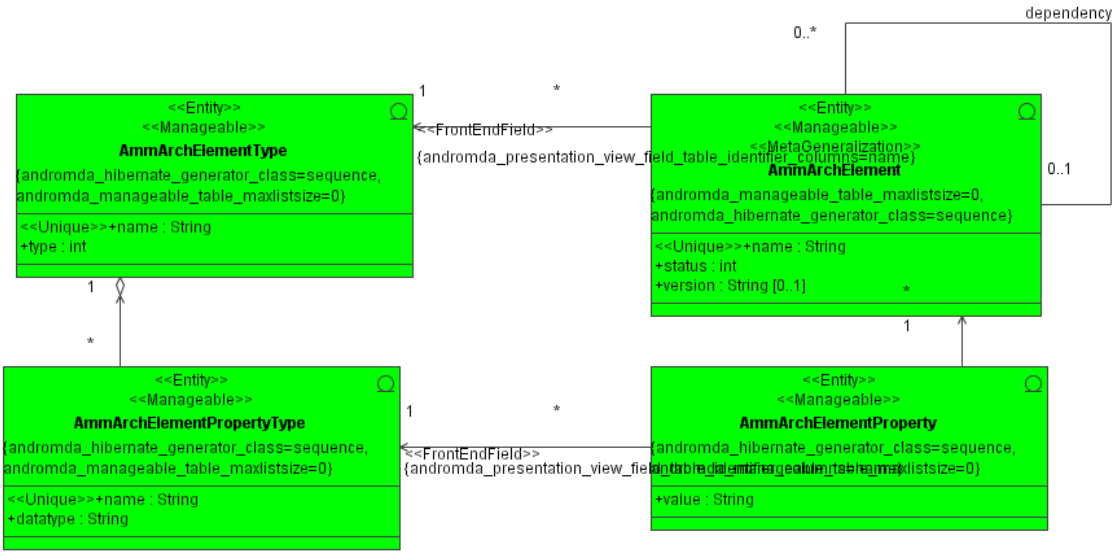
- Classes de objetos que representam Dispositivos.
- Classes de objetos que representam Elementos Arquiteturais (Componentes e Conectores).
- Classes de objetos que representam Qualidade de Serviço (QoS).
- Classes de objetos que representam Elementos do Framework.
- Classes de objetos que representam Usuário e Permissões do Framework.

Devido à natureza predominante *"on-the-fly"* do *framework*, onde alterações do ambiente preferencialmente não devem gerar alterações no sistema alvo, foi percebido que 4 entidades em particular podem variar de sistema alvo para sistema alvo e podem possuir atributos diferentes, o que sugere uma modelagem do tipo generalização/especialização. As entidades em questão são respectivamente: Dispositivos, Elementos Arquiteturais e Recursos (que representam os Gerenciadores de Adaptação).

Para resolver a questão, foi utilizado na modelagem da base de dados o padrão de projeto *Type Square* (YODER et al., 2001), técnica de Modelos de Objetos Dinâmicos que possibilita definir tipos de classes em tempo de execução. Na figura 6.10 é apresentado um diagrama de classes UML com a implementação do padrão *Type Square*.

Um conjunto completo dos diagramas da base de conhecimento pode ser observado no anexo A.

Figura 6.10 - Diagrama de Classes Elementos Arquiteturais



Fonte: Autor

6.4 Gerenciador de Adaptação

O Gerenciador de Adaptação é o componente responsável por receber eventos de componentes e dispositivos, verificar se há planos de adaptação disponíveis para execução e coordenar a execução dos planos.

Ele é composto de: facilidades para o desenvolvimento de lógicas de adaptação, executores e um coordenador/orquestrador de execuções.

A lógica de adaptação pode ser implementada por meio de uma classe java anotada com as Anotações apresentadas na tabela 6.3. A classe java determina uma lógica de adaptação aplicada a um dado recurso gerenciado. Os métodos da classe são o meio pelo qual as táticas são implementadas.

Tabela 6.3 - Descrição dos Elementos para implementação da Lógica de Adaptação

Elemento	**Tipo**	**Descrição**
@AdaptationLogic	Anotação	Identifica uma Lógica de Adaptação.
@ArchitectureElement	Anotação	Identifica um elemento de software como um componente ou conector.
@Property	Anotação	Propriedades de um elemento como url ou nome por exemplo.
@Tactic	Anotação	Identifica um método java como uma tática.
@Condition	Anotação	Informa uma condição inicial para execução da tática.
@ArcticFoxContext	Anotação	Aplicada em um atributo, marca o atributo para recepção do contexto da execução da adaptação, isto é, uma classe especializada que recebe parâmetros, medições, qualquer coisa que for necessária para execução da adaptação .
ArchitectureElementType	Enumeração	Agrupamento de constantes para definição de tipos de componentes suportados. São as constantes previstas para esta enumeração: WEB_SERVICE JAVA_APPLICATION BPEL_PROCESS AGENT INTERCEPTOR THIRD_PARTY_API
EventType	Enumeração	Agrupamento de constantes para definição de tipos de eventos suportados. São as constantes previstas para esta enumeração: SERVICE_REQUEST PLAN_EXPIRED CONDITION_VIOLATION
MatchPolicy	Enumeração	Agrupamento de constantes para definição de políticas de validação de condições. São as constantes previstas para esta enumeração: MATCH_ALL_CONDITIONS MATCH_ONLY_ONE_CONDITION
MetricOperator	Enumeração	Tipos de operadores booleanos que podem ser utilizados em uma condição. São as constantes previstas para esta enumeração: GREATER_THAN GREATER_THAN_OR_EQUAL_TO LESS_THAN LESS_THAN_OR_EQUAL_TO EQUAL NOT_EQUAL
Priority	Enumeração	Prioridade que uma tática pode ter sobre as demais táticas de uma classe. Ou prioridade entre classes que implementam Lógica de Adaptação. São as constantes previstas para esta enumeração: HIGHLY_IMPORTANT MEDIUM_IMPORTANT IMPORTANT LESS_IMPORTANT
AdaptationContext	Classe	Ao executar a tática, a mesma se dá em um determinado contexto, como por exemplo, evento que disparou a sua execução, parâmetros enviados para determinado componente, métricas e outras informações coletadas pelo agente de software que podem ser utilizados pela tática.

Fonte: Autor

A API do framework, embarcada no gerenciador de adaptação, invoca os métodos de cada classe de acordo com o que foi estabelecido no plano de adaptação enviado pelo framework.

Para implementar uma lógica de adaptação, é necessário criar uma classe java anotada com @AdaptationLogic, que marca a classe como lógica de adaptação; as anotações aninhadas @ArchitecturalElement e @Property identificam qual elemento de software se aplica à lógica.

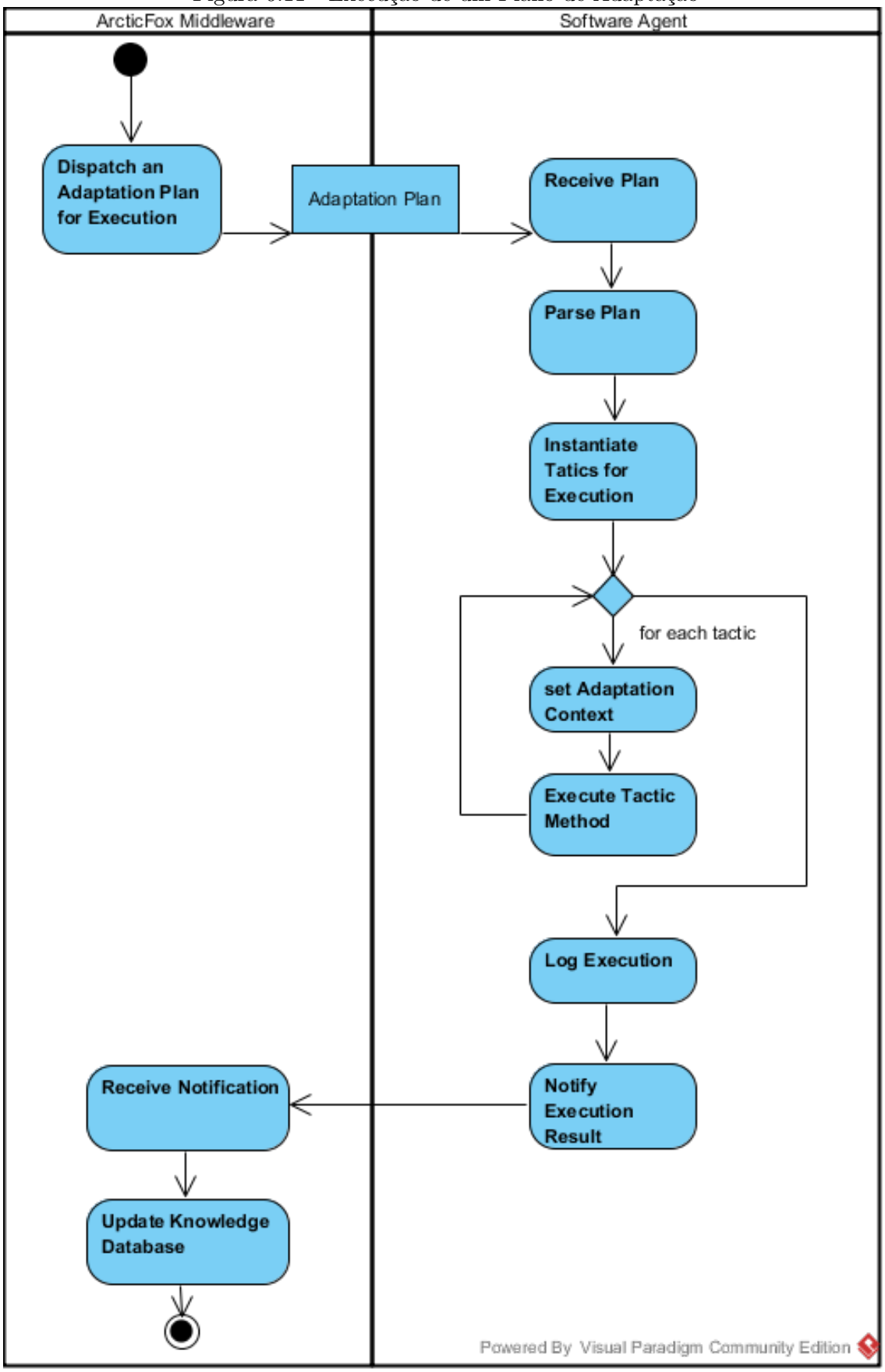
Os métodos são anotados com @Tactic, que informa que aquele método é uma tática que pode ser executada pelo framework. A anotação @Condition determina uma condição inicial para execução das táticas por meio de um valor de métrica de qualidade de serviço associado a uma condição de disparo.

Os valores fornecidos nas anotações das classes são utilizados para detecção das classes e para a configuração inicial na Base de Conhecimento. Pela proposta do framework, as configurações da Base de Conhecimento têm preferência sobre as configurações das anotações.

A classe AdaptationContext, que deve ser um atributo da classe java que contém a lógica de adaptação, deve ser anotada com @ArcticFoxContext. Esta classe representa o contexto da execução da classe, isto é, ela pode ser utilizada dentro dos métodos para acesso a parâmetros e demais informações extraídas e adicionadas a esta classe no momento em que o método é acionado. Como exemplo, pode-se citar um cenário de execução de um serviço Web: o agente, na forma de um proxy web, consegue capturar a solicitação do serviço, extrair parâmetros, valores de cabeçalho, métricas de qualidade de serviço, como tempo de resposta ou vazão em solicitações por segundo, e encapsular esta informação nesta classe.

O processo de execução das táticas, disparado a partir do framework, pode ser observado na figura 6.11. Por meio de um diagrama de atividades UML, é ilustrado o sequenciamento das ações para execução do plano de adaptação desde o encaminhamento pelo framework, a recepção do plano pelo gerenciador de adaptação, a execução das táticas e a notificação do resultado para fins de registro e estatística.

Figura 6.11 - Execução de um Plano de Adaptação



Fonte: Autor

6.5 Monitor de Ambiente

O Monitor de Ambiente é responsável por coletar as métricas de qualidade de serviço, efetuar cálculos e notificar o framework. O componente é parte integrante do ArcticFoxQoS, componente da arquitetura responsável pela coleta e calculo de métricas (MORAES, 2017).

Nesta seção, é apenas apresentado o processo de descoberta de dispositivos e componentes de importância vital para a manutenção da integridade da informação, de modo a evitar a degradação arquitetural.

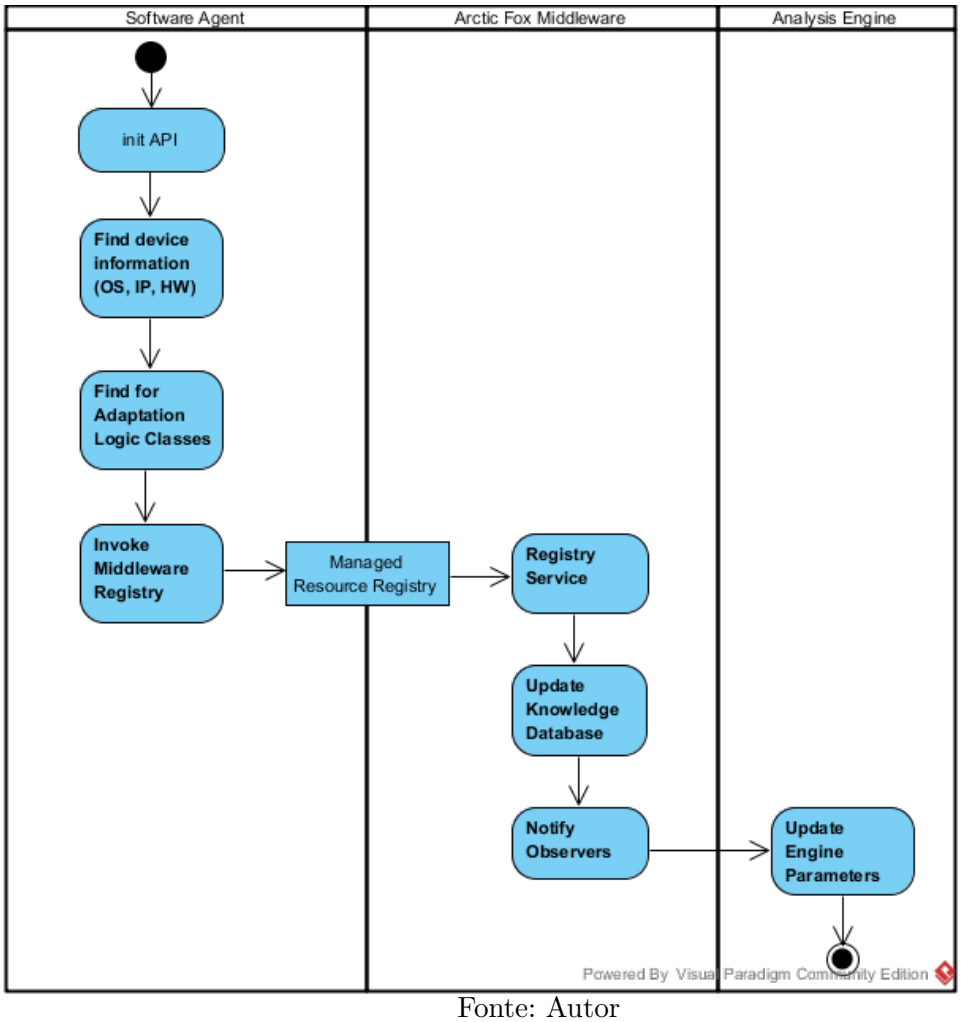
Para tanto, foi desenvolvida uma funcionalidade na API do framework para a descoberta de táticas e identificação de dispositivos e componentes nos quais o monitor de ambiente monitora ou atua.

Há dois tipos de informações que são coletadas no processo de descoberta: estáticas e dinâmicas. A explicação para cada um dos tipos é dada a seguir:

- **Estáticas**: informações são coletadas a partir dos valores dos atributos das anotações contidas nas classes e parâmetros informados no acionamento de métodos das classes disponíveis na API. Também podem ser extraídas a partir de arquivos de configuração.
- **Dinâmicas**: este tipo de informação é coletada em tempo de execução pelo agente, de forma dinâmica, como, por exemplo, informação sobre o tipo de dispositivo, endereço IP, MAC address, tipo de sistema operacional, versão de sistemas operacional, servidor de aplicações, portas, etc.

O processo de descoberta é ilustrado na figura 6.12, por meio de um diagrama de atividades UML.

Figura 6.12 - Processo de Descoberta de dispositivos, componentes e Lógica de Adaptação



Fonte: Autor

6.6 Ferramenta de Administração

De modo a facilitar as alterações de parâmetros da base de conhecimento, foi implementada uma ferramenta de administração com os seguintes grupos funcionais:

a) Dashboard: Painel de Instrumentos com gráficos e dados estatísticos do framework.

b) Gestão de Usuários: permite a criação de usuários e atribuição de permissão de acesso a determinados grupos funcionais (papéis).

c) Gestão de Dispositivos: Gestão de dados de dispositivos e itens de configuração.

d) Gestão de Componentes: Gestão de dados de componentes e itens de configuração.

e) Gestão da Qualidade de Serviço: Gestão de dados de métricas, coletas e configuração de parâmetros de qualidade de serviço.

f) Gestão do Framework: Gestão de itens do framework como lógicas de adaptação, táticas, condições de disparo e itens de configuração.

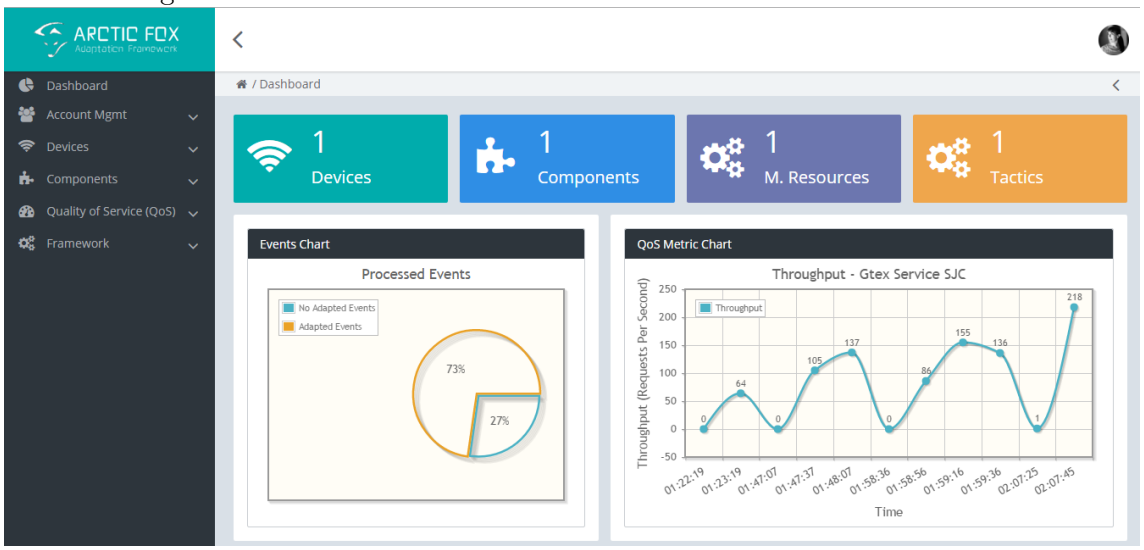
Na figura 6.13, é apresentada a tela de autenticação da interface de administração do framework. O mesmo mecanismo pode ser aplicado para autenticação de serviços do framework quando acessados por componentes ou dispositivos do sistema solo.

Figura 6.13 - Interface de Autenticação do Framework

Fonte: Autor

Na figura 6.14, é apresentado o painel de instrumentos com indicadores e estatísticas do framework.

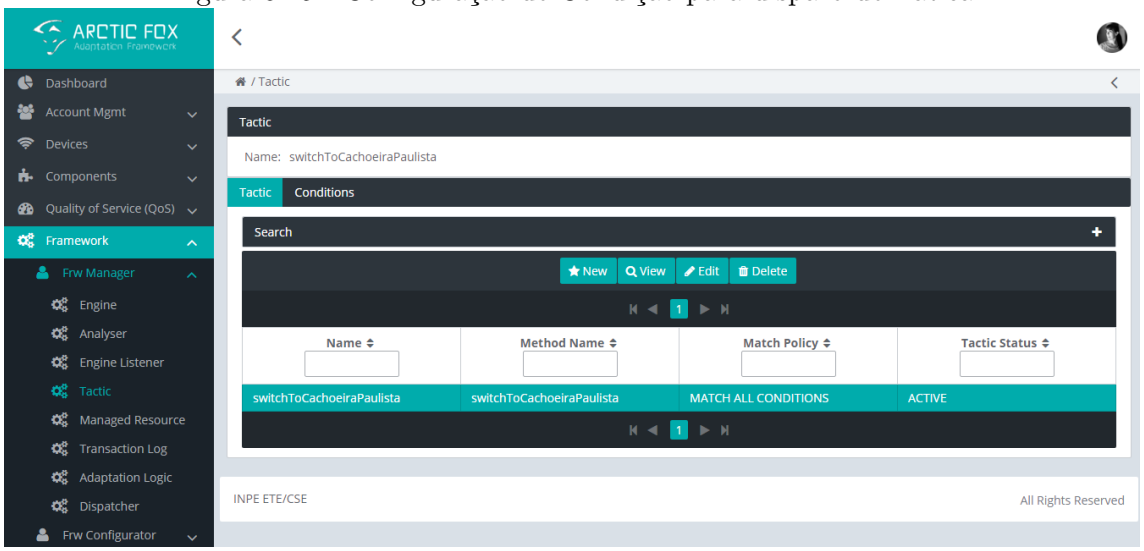
Figura 6.14 - Painel de Instrumentos com Estatísticas do Framework



Fonte: Autor

Na figura 6.15, é apresentada a interface para configuração das condições de disparo de determinadas táticas. Vale ressaltar que as configurações em base de dados sobrescrevem as configurações extraídas das anotações durante o processo de descoberta.

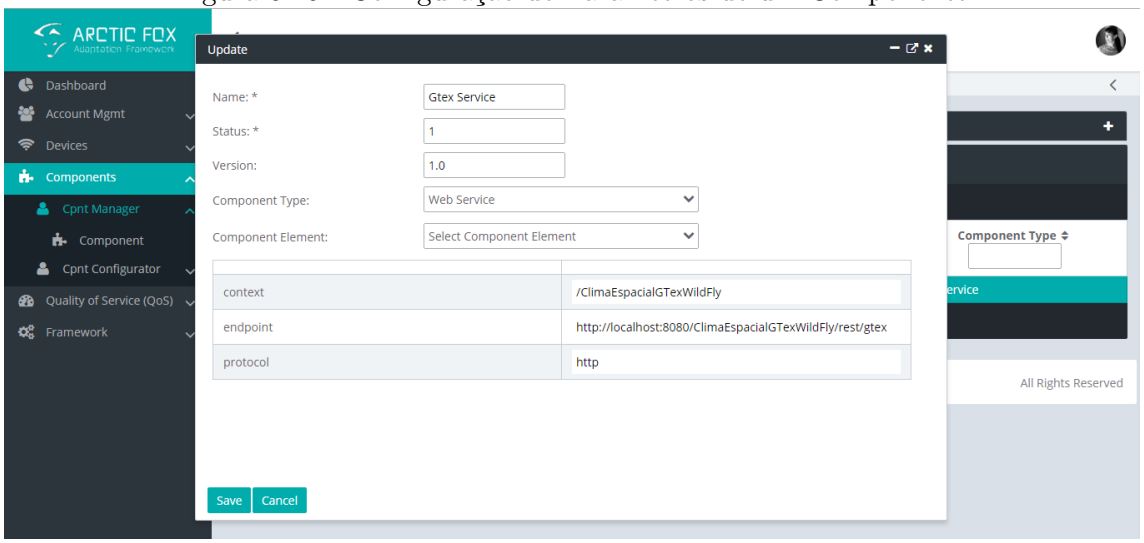
Figura 6.15 - Configuração de Condição para disparo de Tática



Fonte: Autor

Na figura 6.16, é apresentada a interface de configuração de um componente, como mencionado na seção 6.3. Dispositivo constitui uma das entidades modeladas com o padrão *Type Square*. Ao selecionar o tipo de dispositivo, são carregadas as propriedades relacionadas ao tipo selecionado.

Figura 6.16 - Configuração de Parâmetros de um Componente



Fonte: Autor

6.7 Utilização do Framework para Desenvolvimento de Sistemas Auto-Adaptáveis

Como verificado neste capítulo, o protótipo do framework possui diversos mecanismos para permitir o desenvolvimento de lógicas de adaptação, configuração de regras para o gerenciador autônomo, além de mecanismos de extensão.

De modo a possibilitar uma melhor compreensão sobre como utilizar os recursos ora apresentados, para permitir que um sistema existente se torne auto-adaptável, é apresentado, no próximo capítulo, um estudo de caso referente a uma aplicação de clima-espacial.

O estudo de caso proposto e detalhado no próximo capítulo fornece uma visão do ponto de vista do especialista a respeito do sistema solo, sobre como utilizar o framework para atender às necessidades de adaptação que venham a surgir durante a operação do sistema em questão.

7 ESTUDO DE CASO

Este capítulo tem como propósito demonstrar a utilização do framework prototipado em um sistema existente no INPE. Desta forma, são explorados os conceitos e a implementação realizada, através de um experimento controlado. Para a realização deste estudo de caso, são apresentadas neste capítulo as seguintes etapas para sua concretização: o planejamento do experimento, a execução e a coleta dos dados e, por fim, a análise dos resultados. Observa-se aqui que, como estudo de caso, procurou-se replicar as mesmas condições operacionais de um sistema real, porém em condições limitadas e controladas. Um exemplo destas condições limitadas aplica-se à métrica "Tempo de Resposta", onde entendemos que, em uma situação operacional real, os valores obtidos podem variar muito e são imensamente afetados pela infraestrutura computacional e de rede.

Neste experimento, foi explorado o efeito de aumentar a disponibilidade do sistema existente, representado pela métrica "taxa de sucesso das requisições", e o efeito da latência do sistema gerada, representada pela métrica de "tempo de resposta" do serviço ora em foco.

7.1 A Descrição do Sistema Existente Utilizado: O EMBRACE

Neste estudo de caso, foi utilizado o Sistema EMBRACE - Estudo e Monitoramento Brasileiro de Clima Espacial (http://www2.inpe.br/climaespacial/portal/pt/), que é responsável pelo armazenamento, processamento e disseminação de dados científicos para estudos do clima espacial (TAKAHASHI et al., 2008).

No contexto do EMBRACE, insere-se o serviço GTEX (GNSS-TEC Exchange), que é um serviço web de consulta de dados de GPS utilizado pelas aplicações do EMBRACE em operação. As aplicações podem utilizar o serviço GTEX para diversas finalidades científicas.

Uma das finalidades é a produção de um mapa do território brasileiro que informa uma área de cintilação ionosférica que pode causar impacto no funcionamento de receptores de GPS e, desta maneira, afetar usuários que necessitam de precisão para sincronizar posicionamento de equipamentos, por exemplo. A figura 7.1 apresenta um mapa de cintilação e as estações de recepção de dados de constelação GPS como produto do EMBRACE (TAKAHASHI et al., 2012).

Para produzir os mapas, a aplicação utiliza o serviço GTex, que é uma API web para consulta dos dados coletados de estações receptoras de GPS. O serviço foi

Figura 7.1 - Estações de recepção de GPS EMBRACE e aplicação de Mapas TEC (Total Electron Content).

Fonte: EMBRACE

desenvolvido para que os demais componentes da arquitetura PIPELINE possam processar a geração dos mapas de cintilação da ionosfera.

Esta seção apresenta a arquitetura das aplicações de clima espacial do INPE, onde foram levadas em consideração algumas premissas para solução dos problemas relativos ao contexto do clima espacial. As premissas endereçadas pela arquitetura do EMBRACE são:

- Produção de dados por muitos instrumentos e sensores em diferentes taxas e volumes.
- Diferentes aplicações e modelos necessitam de preparação específica de dados, de modo que a operação seja efetiva e eficiente.
- Uma vez que o dado é persistido, é necessário processar o dado bruto para produzir dados derivados por processamento assíncrono, isto é realizado por um subsistema de dados e metadados, que manipula, processa e filtra os dados.
- Fornecer dados de clima espacial por meio de serviços web para outras aplicações e usuários finais.
- Troca de informação por meio de padrões como observatórios virtuais; esta padronização deve ser levada em conta na arquitetura.
- Existe um grande volume de dados que chegam para processamento (aquisição) e um grande número de aplicações e usuário que utilizam os dados.

Para lidar com esta situação, o projeto de arquitetura prevê um bom modelo de persistência de dados (Modelo Lógico).

- Implantação de um conjunto de servidores instalados de maneira escalável, que pode aumentar ou diminuir de acordo com a demanda.

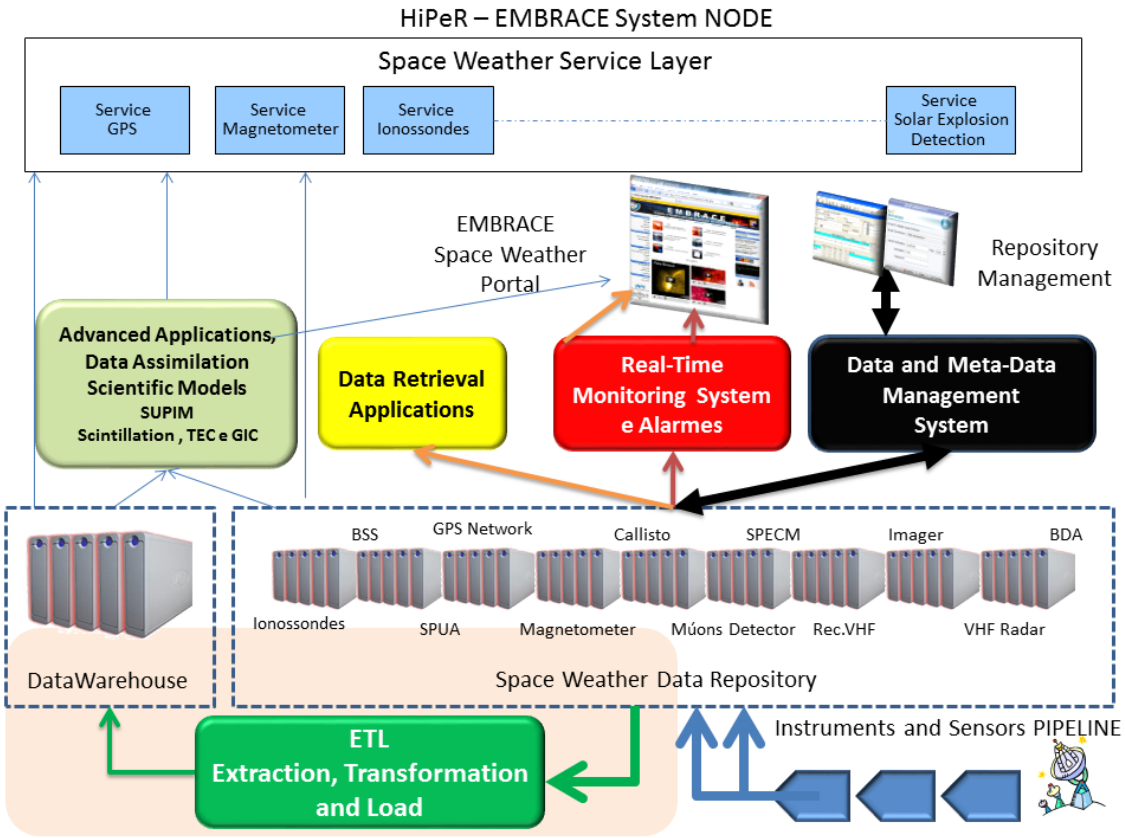
A arquitetura EMBRACE é estruturada em um PIPELINE (SANTANNA et al., 2014), para entregar os dados diretamente aos servidores do clima espacial. Esta solução é responsável por persistir os dados e foi implantada no clima espacial (EMBRACE) para gerenciar a recepção dos dados dos instrumentos.

A figura 7.2 apresenta a arquitetura do clima espacial. Pode ser observada, na arquitetura, a necessidade de utilização de diversos componentes para a execução do pipeline; entre eles estão: banco de dados com dados de instrumentos; componentes para coleta, transmissão e carga de dados de instrumentos, além de aplicações de monitoramento e alarmística, gerenciamento, recuperação de dados e aplicações científicas avançadas com aplicação de algoritmos especializados para análise de fenômenos, como, por exemplo, detecção de explosão solar.

A última premissa prevê requisitos, que devem ser atendidos pela arquitetura, relacionados à escalabilidade de recursos como servidores, componentes de software do pipeline ou a disponibilização de dados via serviços web. Atualmente, a forma selecionada para atender a estes requisitos é através de virtualização e balanceamento de carga com soluções de mercado. Apenas balanceamento de carga e virtualização não são suficientes para tratar redundância em locais diversos, implantação autônoma de pacotes de software, além de outras situações como a redistribuição de carga de uma localidade para outra. As situações descritas pressupõem um elemento de coordenação que faz medições dos elementos da arquitetura e seleciona as melhores opções para adaptar o sistema às novas condições impostas pelo seu contexto.

A próxima seção descreve o uso framework em um cenário simulado de adaptação para o serviço GTex.

Figura 7.2 - Arquitetura Clima Espacial.



Fonte: SantAnna et al. (2014)

7.2 Uso do framework Arctic Fox

Com o propósito de avaliar os componentes do framework Arctic Fox em um cenário de aumento de demanda, a qualidade do serviço será verificada em termos qualitativos entre a solução com a utilização do framework e sem a utilização do framework.

A avaliação proposta será realizada observando-se duas métricas, a *taxa de retorno com sucesso* e o *tempo médio de resposta do serviço* em um cenário de alta carga aplicada simultaneamente a um simulador do serviço Gtex.

O simulador do serviço Gtex tem como finalidade fornecer respostas similares ao serviço encontrado em ambiente de produção do EMBRACE. Atualmente, o serviço está implantado na infraestrutura do EMBRACE na cidade de São José dos Campos.

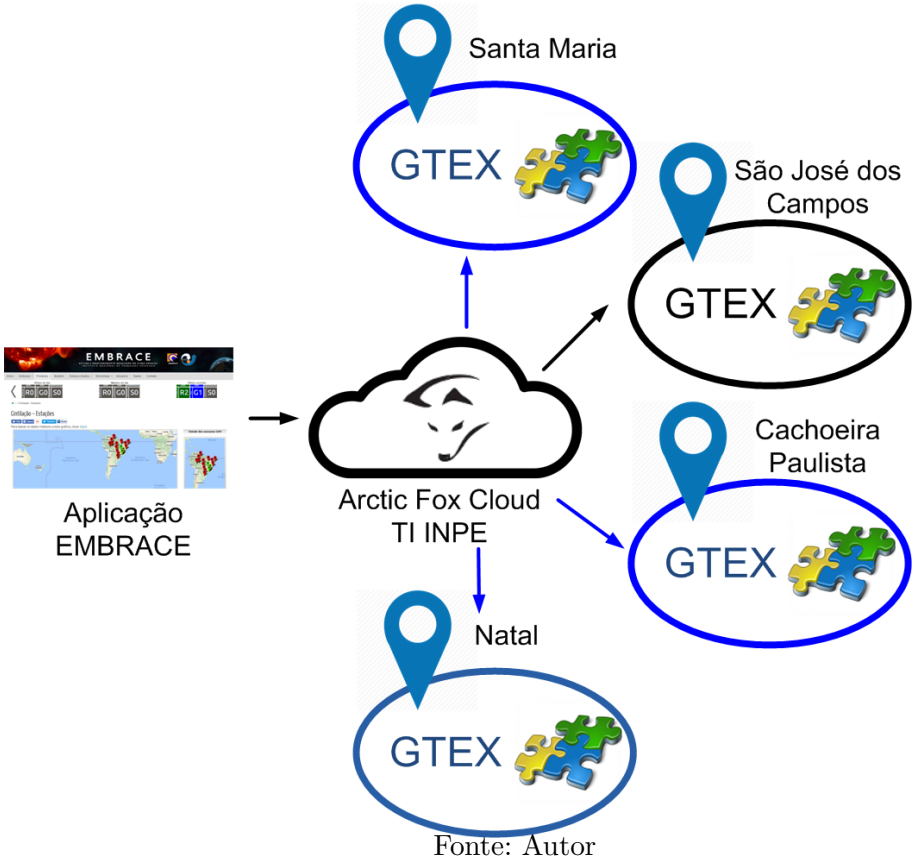
O *framework* Arctic Fox coletará as métricas de qualidade de serviço *tempo de*

resposta do serviço (SRT), e *vazão calculada do serviço (TP(SRV))*. As métricas informadas compõem o atributo de qualidade de serviço "desempenho" definido por Choi et al. (2007).

Quando os valores observados das métricas, atingirem as condições de adaptação definidas na base de conhecimento, o framework executará um plano de adaptação para ativar e redistribuir a carga para três novos nós. Os novos nós, emulam implantações na infraestrutura do EMBRACE nas seguintes cidades: Cachoeira Paulista, Santa Maria e Natal.

Os nós ativados simulam as unidades onde o INPE possui infraestrutura, para receber cópias do serviço GTex. A estratégia de adaptação, adotada para o estudo de caso, é apresentada na figura 7.3; os nós circulados em azul são novos recursos adicionados na infraestrutura de TI pelo *framework* Arctic Fox.

Figura 7.3 - Visão Geral da Estratégia de Adaptação do Estudo de Caso.



Fonte: Autor

As métricas utilizadas como condição de disparo, para as táticas de ativação de nós,

são as descritas abaixo:

Tempo de Resposta do Serviço, calculado pela fórmula definida em Choi et al. (2007):

$SRT = SCFSRT - SCSRR$

onde:

SRT = Tempo de Resposta do Serviço

$SCFSRT$ = Tempo utilizado pelo consumidor para enviar a solicitação

$SCSRR$ = Tempo no qual é iniciada a recepção da resposta pelo consumidor

Vazão de um Serviço, calculado pela fórmula definida em Choi et al. (2007):

$TP(SRV) = \frac{NCSR}{UT}$

onde:

$TP(SRV)$ = Vazão do Serviço

$NCSR$ = Número de Solicitações Completadas para o Serviço

UT = Unidade de Tempo

Os componentes prototipados do *framework*, descritos na seção 5.5, avaliados neste estudo de caso são, respectivamente:

- ArcticFoxQoS
- ArcticFoxCore
- RuleBasedEngine
- ArcticFoxKnowledgeDB
- AdaptationManager

7.3 Planejamento do Experimento

Esta seção visa apresentar a metodologia e as ferramentas empregadas no estudo de caso, para avaliar se há diferença significativa na adoção do framework Arctic Fox

em cenário de alta carga de dados do EMBRACE, para manter a disponibilidade do serviço.

7.3.1 Simulador Serviço GTEX

O simulador foi implementado para trazer dados reais coletados no EMBRACE para dez estações de GPS, o que pode ser verificado na figura 7.4. O simulador do serviço em questão acessa uma base de dados com os retornos das estações e, além disso, também foi implementado um atraso na resposta, de modo a refletir o tempo de resposta do serviço instalado atualmente em operação no EMBRACE.

Figura 7.4 - Saída do serviço Web GTEX com dados de GPS

```
     2            GTEX DATA          GPS                 GTEX VERSION / TYPE
RNX2GTEx V1.0       EMBRACE/INPE, BRAZIL                 PGM / RUN BY
       0                                                 EXPONENT OF TECU
       TEC values in 10^16 el/m^2 (1 TEC Unit)           COMMENT
       TEC Status Flag = 0 : Normal data                 COMMENT
                       = 1 : Lack of observables (TEC=999.) COMMENT
                       = 2 : Too large TEC (TEC=999.)    COMMENT
                       = 4 : Cycle slip (TEC discontinuity) COMMENT
                       = 5 : Cycle slip (LLI)            COMMENT
                       = 6 : Beginning of arc            COMMENT
       TYPES OF DATA   = R1 : Raw Slant TEC including bias COMMENT
                         A1 : Absolute slant TEC         COMMENT
                              R1 or A1 is necessary      COMMENT
                         1F : TEC status flag            COMMENT
                         1O : Observation data used for TEC COMMENT
                         ZN : Satellite zenith angle     COMMENT
                         AZ : Satellite azimuth angle    COMMENT
                                                         BIAS ESTIMATION PGM
ampr0301.16o                                             RINEX FILE NAME
ampr                                                     MARKER NAME
00000                  TPS NETG3     3.4 EG3 Jul, 02, 2010 REC # / TYPE / VERS
                     TRM29659.00       GSI               ANT # / TYPE
     3494.8889    -5327.4145     -290.8459               APPROX POSITION XYZ
       -2.6312      -56.7343        0.0097               POSITION LAT LON ALT
     6    L1    C1    L2    P2    S1    S2               # / TYPES OF OBSERV
     5    A1    1F    1O    ZN    AZ                     # / TYPES OF DATA
    30.0000                                              INTERVAL
  2016    1     30     0     0    0.0000000      GPS      TIME OF FIRST OBS
                                                         END OF HEADER
 16 01 30  0  0  0.0000000  0  2G27G 8
  41.8645  0  L1L2C1P2  15.27  180.80
  46.0581  0  L1L2C1P2  32.92  179.81
 16 01 30  0  0 30.0000000  0  2G 8G27
  46.2040  0  L1L2C1P2  32.78  179.81
  42.6619  0  L1L2C1P2  15.47  180.80
 16 01 30  0  1  0.0000000  0  2G27G 8
  43.1398  0  L1L2C1P2  15.68  180.80
  47.0971  0  L1L2C1P2  32.64  179.81
```

Fonte: Autor

Não foram implementadas rotinas de tratamento de exceções, isto quer dizer que todos os acionamentos ao simulador retornam sucesso. Sucesso nesta aplicação é um código de retorno 200 (Sucesso) do protocolo HTTP, qualquer retorno diferente de 200 indica falha do servidor de aplicação ou algum outro problema na rede como latência elevada, ou mesmo falha de conexão ao serviço.

Para estabelecer um parâmetro de comparação para o estudo de caso, foi executado um teste de carga no serviço para avaliar o tempo médio de resposta e e vazão calculada para as solicitações enviadas ao simulador do serviço GTex. O Teste foi executado em um computador local de modo a reduzir impacto de latência para fornecer uma base de comparação para o estudo de caso, os dados podem ser verificados na tabela 7.1.

Tabela 7.1 - Tempo médio de resposta do simulador em milisegundos

Parâmetro	**Valor**
Número de Amostras	1553 solicitações
(SRT) médio	194,10 ms
Desvio Padrão	58,93 ms
Vazão calculada TP(SRV)	33,8 solicitações/seg

Fonte: Autor

Além das métricas, tempo médio de resposta e vazão calculada, observadas no teste de carga aplicado no simulador, não foram observados erros no retorno do serviço. O teste de carga não teve como propósito provocar um cenário de degradação do serviço, apenas avaliar a operação em condições normais de utilização.

7.3.2 Descrição da Estratégia de Adaptação para o Estudo de Caso

O estudo de caso consiste em enviar um volume similar de solicitações ao serviço GTex em duas configurações: o primeiro cenário consiste no envio das solicitações diretamente para o simulador do serviço GTex, enquanto o segundo cenário consiste no envio das solicitações para o Framework Arctic Fox, que, inicialmente, acionará a mesma cópia do simulador.

O propósito desta abordagem está em analisar se há ganhos significativos na utilização do framework em um cenário de carga elevada ante uma configuração convencional sem uso de adaptação para a disponibilidade do sistema.

Para viabilizar o cenário proposto, foi implementado um conjunto de artefatos que possibilitam ao framework mudar a topologia atual do serviço GTex, de modo a incluir outros recursos computacionais para manter o serviço em operação para reduzir a sua degradação. Os artefatos produzidos são, respectivamente: Script de Teste, Simulador e Lógica de Adaptação.

Para a Lógica de Adaptação *GtexAdaptationLogic*, foram implementadas duas táticas para a execução da estratégia: *activateGtexAlternativeNodes*, responsável pela ativação dos nós; e *rebalanceTraffic*, responsável por redirecionar o tráfego para os nós ativados. As condições de disparo configuradas na base de conhecimento do framework são apresentadas na tabela 7.2.

Tabela 7.2 - Condições de disparo de táticas para a Lógica de Adaptação *GtexAdaptationLogic*

Métrica	**Condição de Disparo**	**Tática**
Tempo médio de resposta (SRT)	> 250	rebalanceTraffic
Vazão calculada do serviço TP(SRV)	< 10	activateGtexAlternativeNodes

Fonte: Autor

A estratégia adotada para o estudo de caso é descrita a seguir:

a) Inicialmente, o JMeter aciona o Script de Teste para geração das solicitações;

b) O JMeter encaminhas solicitações em simultâneo para o simulador GTex SJC e para o Framework Arctic Fox;

c) O Framework Arctic Fox calcula as métricas tempo de resposta e vazão calculada;

d) O Framework Arctic Fox analisa a métrica vazão calculada e verifica se atingiu o limite estipulado, acionando, desta forma, a tática para ativação dos nós em Cachoeira Paulista, Santa Maria e Natal, implementada na Lógica de Adaptação;

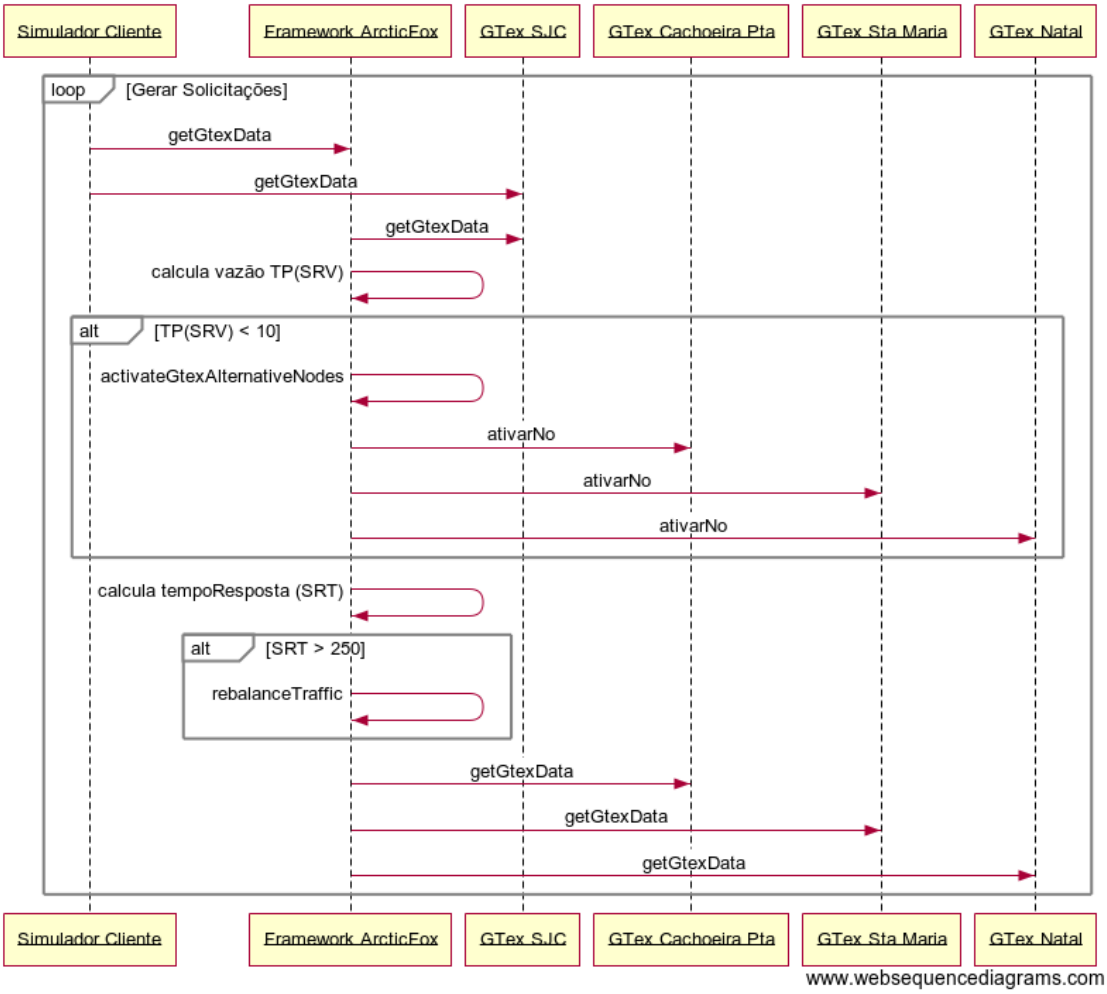
e) O Framework Arctic Fox analisa a métrica de tempo de resposta e aciona a tática para recalibrar os recursos disponíveis e passa a direcionar o tráfego para os três novos nós.

f) O Jmeter continua enviando solicitações em simultâneo para o GTex SJC e o Framework;

g) O Jmeter coleta dados de códigos de retorno HTTP, tempo de conexão, latência e tempo de resposta recebidos das respostas das solicitações enviadas.

A estratégia informada para o estudo de caso pode ser observada no diagrama de sequência UML apresentado na figura 7.5.

Figura 7.5 - Descrição do Estudo de Caso



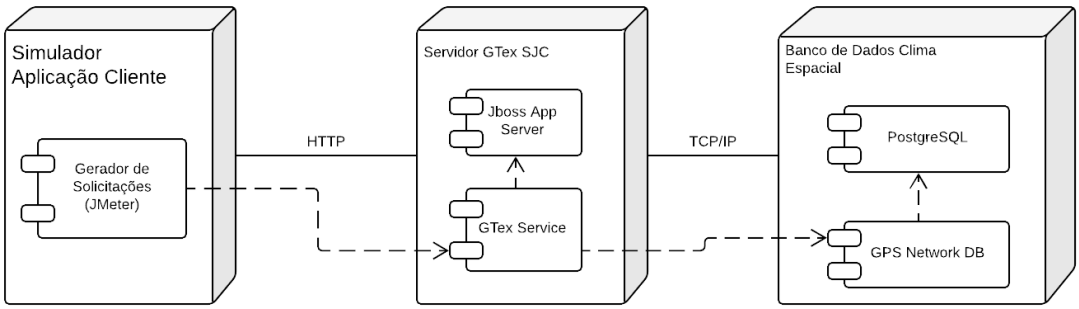
Fonte: Autor

7.3.3 Cenário de Implantação do Estudo de Caso

Foram necessárias duas implantações, sendo a primeira a implantação de um simulador GTex em um container docker, de modo a emular o cenário atual do EMBRACE com o serviço instalado apenas nas dependências da unidade de São José dos Campos. Já a segunda implantação consiste na ativação de imagens docker do serviço GTex, de maneira a emular a implantação dos serviços nas unidades de Cachoeira Paulista, Santa Maria e Natal.

A primeira implantação, que emula o cenário atual de implantação do serviço na localidade de São José dos Campos, pode ser observada na figura 7.6.

Figura 7.6 - Implantação de Cenário de Infra-Estrutura Atual do Serviço GTex



Fonte: Autor

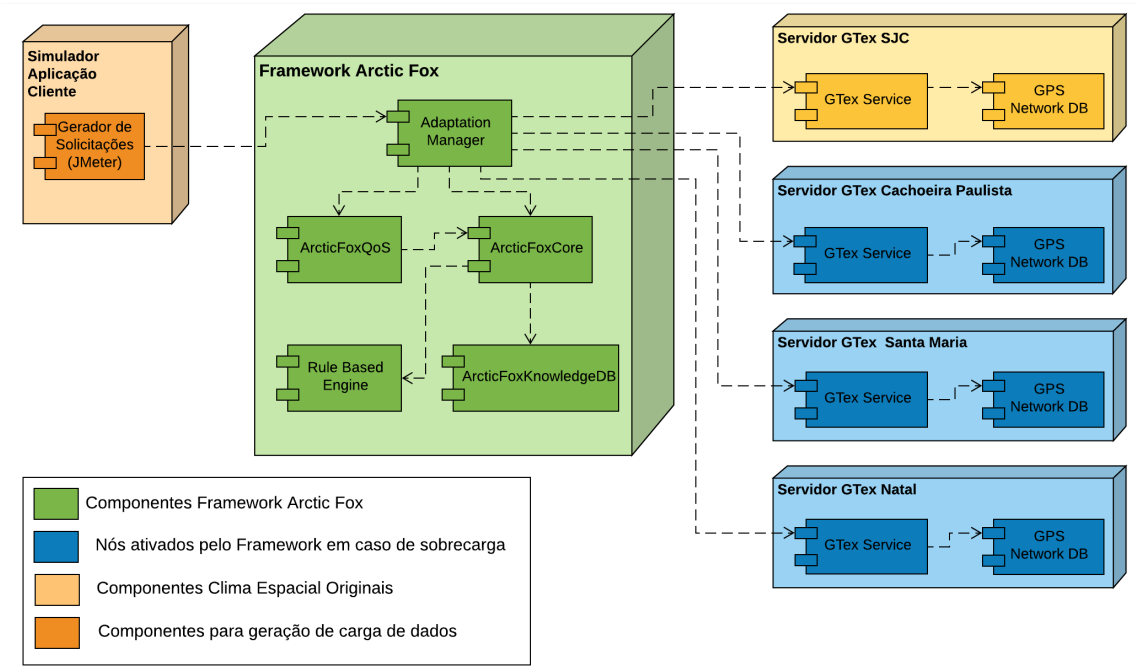
A segunda implementação, que considera um cenário de adaptação não invasiva, a exemplo do apresentado na seção 5.4.1, permite avaliar o funcionamento do framework em uma situação de carga elevada. O framework atua como um interceptador das solicitações da ferramenta JMeter, realiza os cálculos de métricas de qualidade de serviço, e aciona a Lógica de Adaptação implementada para o estudo de caso, conforme configurado na base de conhecimento. Os cálculos das métricas geram eventos para avaliação pelo gerenciador autônomo do *framework*.

Confirmada a ocorrência de evento que indique aumento de demanda, o framework executa a tática implementada para a ativação de três novos nós. Esta ação é necessária para aumentar a capacidade de resposta de modo a atender a nova demanda. Em seguida, o framework, depois de observada carga superior ao limite configurado na Base de Conhecimento para o serviço GTex, executa tática para redistribuição de carga para os novos nós ativados, que emulam as localidades de Cachoeira Paulista, Santa Maria e Natal. Esta nova topologia arquitetural, gerada após a aplicação das táticas pelo framework Arctic Fox, pode ser observada na figura 7.7.

7.3.4 Metodologia

Serão utilizadas duas métricas para avaliação do estudo de caso: tempo médio de resposta e taxa de sucesso. A avaliação consiste em determinado se existe diferença significativa para as duas métricas entre os cenários com e sem a utilização do framework.

Figura 7.7 - Implantação de Infra-Estrutura com Utilização do Framework Arctic Fox.



Fonte: Autor

O critério para avaliação desta métrica está em analisar os resultados de um teste de hipótese estatística, de modo a determinar se há diferença nas taxas de sucesso do sistema com e sem o uso do framework.

De outro lado, é necessário avaliar as consequências do uso do framework com relação ao desempenho do sistema, quando ocorre a execução da estratégia de adaptação. De maneira a avaliar este cenário, foram definidas hipóteses estatísticas avaliar também o tempo de resposta do sistema quando é utilizado o framework Arctic Fox.

7.3.4.1 Hipótese Estatística para Avaliação da Proporção da Taxa de Sucesso

De maneira a avaliar se a há ganho para o serviço GTex, para o atributo disponibilidade, com a utilização do framework é necessária a comparação da proporção de respostas com o código de retorno de sucesso, HTTP 200, para os cenários com e sem a utilização do framework.

Considerando uma distribuição normal, para verificação se há diferença significativa, foram definidas as hipóteses estatísticas para teste (tipo z) para comparação da

proporção de duas populações (GRAYBILL et al., 1997), com grau de significância de 5%, $\alpha = 0,05$, para os retornos com sucesso dos dois cenários.

As hipóteses nula e alternativa, definidas para avaliação estatística da proporção de retornos com sucesso, foram respectivamente:

H_0 = Não há diferença significativa entre as proporções de retorno com sucesso

H_1 = Há diferença significativa entre as proporções de retorno com sucesso

isto é:

$$\begin{cases} H_0 : p_1 - p_2 = 0 \\ H_1 : p_1 - p_2 \neq 0 \end{cases}$$

onde:

p_1 = proporção para a taxa de sucesso sem adaptação

p_2 = proporção para a taxa de sucesso com uso do framework

7.3.4.2 Hipótese Estatística para Tempo Médio de Resposta

Para avaliação se há diferença significativa no desempenho da utilização do framework Arctic Fox, em relação ao cenário sem utilização de adaptação, considerando uma distribuição normal, a métrica tempo de resposta foi submetido a um teste de hipótese estatística para média de duas populações com variância desconhecida t-Student unicaudal (GRAYBILL et al., 1997), com grau de significância de 5%, $\alpha = 0,05$, de modo a avaliar a equivalência do desempenho entre os cenários com e sem adaptação.

As hipóteses nula e alternativa definidas para avaliação estatistica da média de tempo de resposta foram respectivamente:

H_0 = Não há diferença significativa entre a média de tempo de resposta

H_1 = Há diferença significativa entre a média de tempo de resposta

isto é:

$$\begin{cases} H_0 : \mu_2 - \mu_1 = 0 \\ H_1 : \mu_2 - \mu_1 > 0 \end{cases}$$

onde,

μ_1 = tempo médio de resposta do serviço sem adaptação

μ_2 = tempo médio de resposta do serviço com uso do framework

7.4 Execução do Experimento

Nesta seção, são apresentados os recursos e resultados da execução do teste de carga aplicado aos cenários com e sem a utilização do framework.

7.4.1 Componentes de Software e Infraestrutura

Esta seção apresenta os recursos de software e hardware utilizados para a execução do estudo de caso, são listados os softwares proprietários e desenvolvidos para o estudo de caso, bem como a infraestrutura de máquinas físicas e virtualizadas utilizadas para a execução do estudo de caso. Ao final da seção, é apresentado um diagrama de implantação com a distribuição dos componentes de software instalados na infraestrutura de hardware.

7.4.1.1 Software Utilizados

Para implantação do simulador e framework para o estudo de caso, foram utilizados os seguintes softwares:

- Java Virtual Machine - JVM versão 1.6
- Apache Jmeter versão 4.0
- Script de teste de desempenho para acionamento dos simuladores Gtex e coleta de dados para análise dos resultados.
- WildFly Application Server versão 10 para implantação do simulador de Gtex e Framework Arctic Fox.
- PostgreSQL Database versão 10;
- Docker Container versão 1.13.1 - O banco de dados postgres e os simuladores foram instalados em containers docker.
- arcticfox-1.0.0.ear - Arquivo de instalação do *framework* Arctic Fox.
- ClimaEspacialGTex.war - Arquivo de instalação do simulador GTex.

7.4.1.2 Hardware Utilizado

Para execução do cenário, foram utilizados um computador pessoal e um servidor. O computador pessoal foi utilizado para a implantação do framework Arctic Fox e da ferramenta de teste de desempenho Apache JMeter. No servidor, estão implantados os containers docker para o Banco de Dados de Estações GPS GTex e os containers docker que emulam os sites de São José dos Campos, Cachoeira Paulista, Santa Maria e Natal.

Abaixo a descrição dos recursos de hardware utilizados no estudo de caso:

- Computador Pessoal 1 (*JMeter* e *Framework*)
  - CPU: Intel Core I5
  - RAM: 8GB
  - Capacidade de Armazenamento: 500GB
  - S.O.: Windows 7 Professional Edition
- Servidor 1 (*Containers Docker*)
  - CPU: Intel(R) Xeon(R) CPU 2.40GHz Dual Core
  - RAM: 16GB
  - Capacidade de Armazenamento: 160 GB
  - S.O.: Linux CentOS

7.4.1.3 Instalação dos Componentes

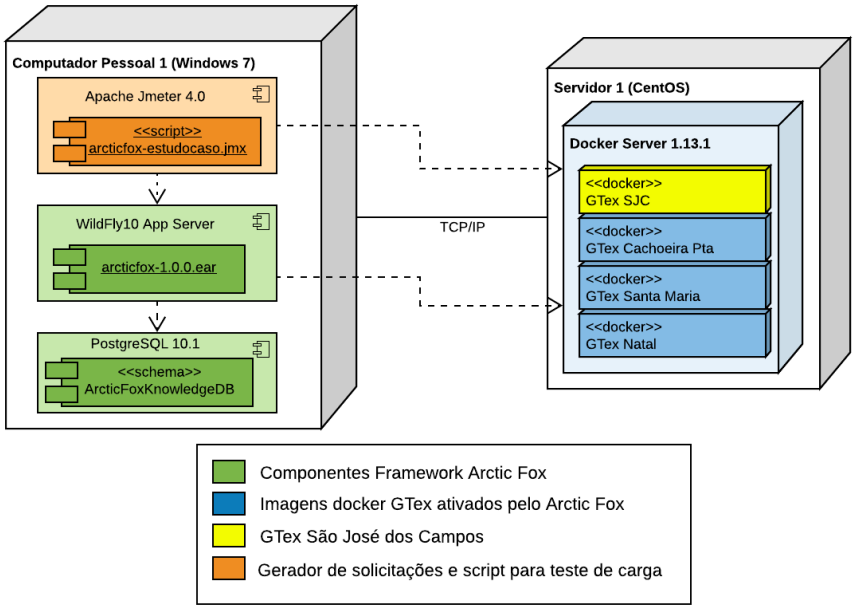
A instalação dos componentes de software na infraestrutura de hardware, pode ser visualizada na figura 7.8.

7.4.2 Execução do Estudo de Caso

O script de teste de carga utilizado na ferramenta Apache JMeter foi configurado para armazenar os dados coletados durante o teste em arquivo tipo texto, com campos separados por vírgula (CSV). Os dados armazenados são respectivamente: tempo de resposta, códigos de retorno HTTP, nome do serviço acionado, thread de execução.

A execução do teste teve uma duração total de 7min32seg. Verificou-se, de uma maneira geral, um quadro com variações que elevaram significativamente o tempo de

Figura 7.8 - Instalação de Componentes de Software e Infra-estrutura de Hardware



Fonte: Autor

resposta da solução, comparado ao tempo observado no teste aplicado ao simulador em máquina local.

A principal diferença observada está relacionada ao componente latência, observado no tempo de resposta, o qual se mostrou muito elevado.

O ArcticFoxQoS efetuou cálculos das métricas, tempo médio de resposta e vazão calculada a cada 10 segundos; cada cálculo gerou um evento para análise pelo framework Arctic Fox.

O script de carga foi programado para criar até 65 threads para envio de solicitações de forma progressiva em até 90 segundos. Neste período inicial de testes, todas as solicitações encaminhadas ao framework foram direcionadas ao serviço Gtex, que emula a localidade de São José dos Campos.

A condição de disparo foi atingida acerca de pouco mais de 1 minuto após o início da execução do teste de carga, o tempo total verificado para as ativações e atendimento das solicitações foi de 2 minutos, as ativações foram realizadas de forma incremental. O tempo de resposta das primeiras solicitações também se apresenta lento devido ao fato de a aplicação iniciar recursos (threads) para atendimento das solicitações.

O Framework Arctic Fox executou a tática para transferência de tráfego para os novos nós, logo em seguida à ativação. A partir deste momento, o tráfego foi balanceado entre os três nós que emulam as localidades Cachoeira Paulista, Santa Maria e Natal.

7.4.3 Resultados para a Métrica Taxa de Sucesso

Com relação aos códigos HTTP observados, pode-se verificar que houve degradação severa na qualidade do serviço prestado pelo simulador GTex que emula a cidade de São José dos Campos; foi observada uma taxa de sucesso de 94,10% das respostas da emulação do serviço GTex de São José dos Campos. Em contrapartida, 100% das respostas recebidas do framework obtiveram código de sucesso HTTP 200. Os dados podem ser observados na tabela 7.3 e no gráfico apresentado na figura 7.9.

Tabela 7.3 - Comparativo retorno com sucesso x erros

	Framework Arctic Fox	**Sem Adaptação**	**% Framework Arctic Fox**	**% Sem Adaptação**
Sucesso	26622	25163	100	94,10
Erro	0	1578	0	5,90
Totais	26622	26741	100	100

Fonte: Autor

Os dados calculados para o teste de proporção de taxa de sucesso, para as duas populações, podem ser observados na tabela 7.4.

Tabela 7.4 - Teste de Proporção dos Cenários com e sem Adaptação

Variável	**Valor Calculado**
p1	0,9409
p2	1,0000
p-valor (estimativa ponderada)	0,9704
Z observado	40,2663
z crítico superior	1,96
z crítico inferior	-1,96

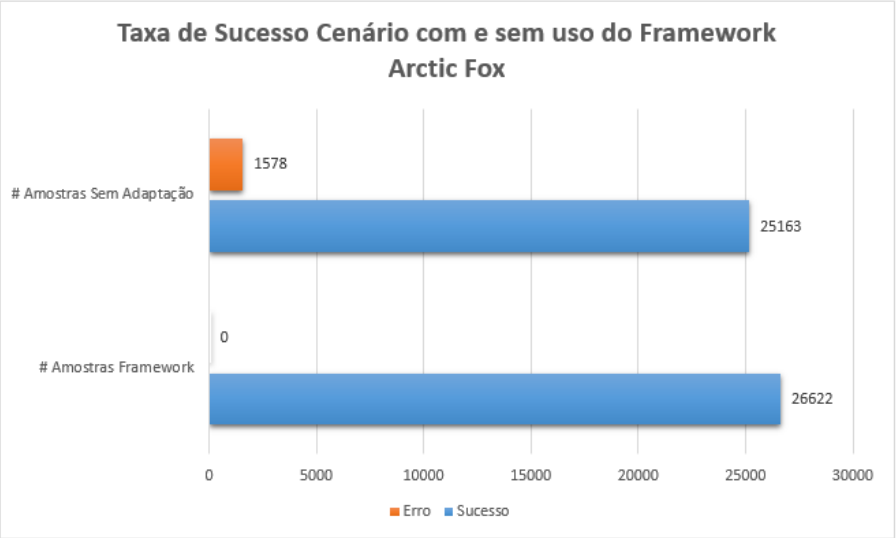
Fonte: Autor

Dada a área de aceitação de (H_0), $P(z_{\text{inferior}} <= z_{\text{observado}} <= z_{\text{superior}})$, $Z_{\text{observado}} = 40,2663$, maior que o limite superior definido para a área de aceitação $z_{\text{superior}} =$

1, 96, rejeita-se a hipótese nula. Desta maneira, ao nível de significância de 95%, fica evidenciada a diferença significativa entre as taxas de sucesso com o uso do framework em relação à taxa de sucesso sem o uso do framework.

Uma observação com relação aos erros: todos os erros registrados pela ferramenta JMeter se deram por exceder o tempo limite de resposta (connection timeout). O tempo padrão configurado na ferramenta JMeter para aguardar uma resposta de uma solicitação é de 500 milisegundos. Com uma taxa de erro próxima a 20% das solicitações no cenário sem adaptação, constata-se o impacto direto na diferença das médias de tempo de resposta, dados os problemas de latência registrados, observados na seção anterior.

Figura 7.9 - Comparativo Solicitações com Sucesso X Erros



Fonte: Autor

7.4.4 Resultados para a Métrica Tempo de Resposta

Para análise sobre a diferença de desempenho com e sem aplicação de adaptação, a diferença indica a hipótese de o tempo de resposta ser superior com o uso do framework em relação ao outro cenário sem uso. Para análise do quadro apresentado, foram calculados os valores das variáveis definidas no teste t-Student. De modo a comparar as médias de tempo de resposta das amostras extraídas, são apresentados os valores nas tabelas 7.5 e 7.6.

Dada a área de aceitaçãode (H_0), $P(T_{\text{observado}} <= t_{\text{crítico}})$, o resultado $t_{crítico} =$

Tabela 7.5 - Cálculos de Média, Variância dos cenários

	Framework Arctic Fox	**Gtex Sem Adaptação**
Média	1908,27	1499,14
Variância	2759459,98	2070292,58
Desvio Padrão (ms)	1661,16	1438,85
Observações	26622	26741

Fonte: Autor

Tabela 7.6 - Resultado Teste de Hipótese Tempo de Resposta

Variável	**Valor Calculado**
alfa (nível de significância)	0,05
Variância Agrupada	52231,00
Hipótese de diferença da média	0
gl (grau de liberdade)	1710
T observado	30,40
t crítico	1,65

Fonte: Autor

$1,65$ e $T_{observado} = 30,40$, conclui-se que deve ser rejeitada a hipótese nula. Desta maneira, ao nível de significância de 5%, fica evidenciada diferença significativa entre as médias de tempo de resposta, das duas populações de dados.

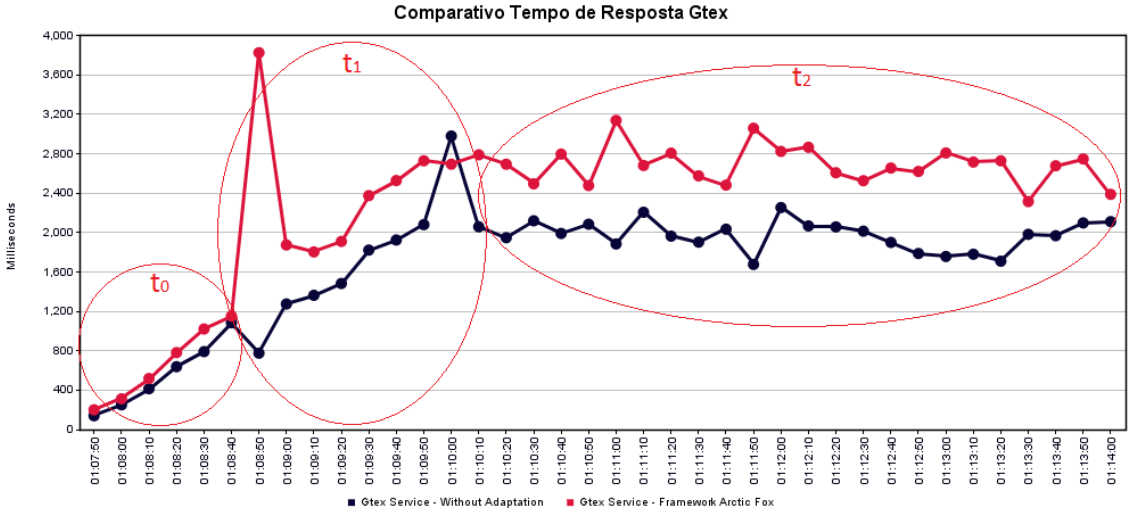
Este dado deve ser analisado com maior profundidade, e, para verificação das causas deste resultado, pode-se extrair algumas observações dado o cálculo parcial de tempos médios de resposta pela ferramenta JMeter, efetivados durante o período de duração dos testes, divididos em três períodos.

- t_0 - instante 01:07:50 a 01:08:40 - Período inicial dos testes, carga concentrada no simulador São José dos Campos.
- t_1 - Instante 01:08:50 a 01:10:00 - Período em que as condições de disparo das táticas foram atingidas, neste período a Lógica de adaptação para ativação de nós e redistribuição de carga foi executada pelo framework. Parte da grande diferença das médias pode ser observada neste período.
- t_2 - Instante 01:10:10 em diante - Período de estabilização, nós ativados e carga redistribuida pelo framework. Serviço GTex sob carga intensa, opera

em condições de degradação severa.

A variação dos tempos de resposta dos dois cenários, nos três períodos, pode ser observada no gráfico apresentado na figura 7.10.

Figura 7.10 - Média de Tempos de Resposta (calculadas a cada 10s)



Fonte: Autor

7.5 Análise dos Resultados

Foi verificado que, em um cenário de alta carga, há diferença significativa para a métrica taxa de sucesso com a utilização do framework, comparado ao cenário sem a utilização do framework. A taxa de sucesso verificada para o framework foi de 100% de sucesso ante a uma taxa de 94,10% sem a utilização do *framework*.

Desta maneira, conclui-se que, com um nível de confiança de 95%, há diferença significativa para a disponibilidade do serviço GTex. Neste sentido, para a estratégia de adaptação e as condições propostas no estudo de caso, houve um ganho de qualidade para as respostas com sucesso do serviço.

De outro lado, verificou-se, com um nível de confiança de 95%, que o tempo médio de resposta, para a estratégia de adaptação e as condições de ambiente aplicadas ao estudo de caso, foi maior para o cenário com a utilização do framework em relação ao cenário sem adaptação.

O ganho de disponibilidade do serviço GTex gerou um custo computacional relacio-

nado às atividades de adaptação. As atividades de ativação dos nós nas outras cidades e redistribuição de carga provocaram uma diferença entre as médias de tempo de resposta do serviço. Entretanto, não foi possível avaliar o impacto no tempo médio de resposta, de outras variáveis, como a degradação do serviço GTex da cidade de São José dos Campos e a latência, dadas as condições de tráfego de rede.

As condições observadas na execução dos testes podem não se repetir em cenários de produção como as encontradas no INPE, haja vista uma melhor infraestrutura de rede e recursos computacionais. Desta maneira, deve ser efetuada uma análise do tipo "tradeoff", isto é, um balanço sobre o ganho em termos qualitativos contra eventuais perdas de desempenho, dado o custo gerado pela adaptação.

8 CONSIDERAÇÕES FINAIS

Neste trabalho, foram pesquisados os componentes de um sistema espacial com foco nos sistemas do segmento solo. Uma análise criteriosa de diversas missões espaciais indica um impacto na arquitetura dos sistemas espaciais, onde é necessário o emprego de um grande número de instrumentos que podem ser embarcados em cargas úteis de satélites ou instalados em solo. As arquiteturas de sistemas espaciais, cada vez mais, baseiam-se na utilização de constelações e mega-constelações de satélites para atender aos requisitos de missão.

Esta nova situação é uma realidade no contexto do INPE, mais precisamente sistemas solo para disseminação de dados científicos como o EMBRACE - Estudo e Monitoramento BRasileiro de Clima Espacial, que coleta, processa e dissemina informações provenientes de diferentes instrumentos para a comunidade científica.

Observou-se, neste contexto, a necessidade de incluir, nos projetos de desenvolvimento de sistemas, requisitos relacionados à operação autônoma. Uma vasta pesquisa foi realizada no campo dos sistemas de software auto-adaptáveis propostos neste trabalho, para resolver a questão relacionada à operação autônoma dos sistemas de segmento solo. De uma maneira geral, algumas observações podem ser extraídas das pesquisas realizadas nesta área.

A primeira observação indica uma maior propensão à adoção do modelo MAPE-K (KEPHART; CHESS, 2003) para o desenvolvimento de sistemas auto-adaptáveis, isto implica que as abordagens, de uma maneira geral, possuem um ciclo de adaptação baseado em monitoramento, análise, planejamento e execução. Ainda neste contexto, são empregadas bases de conhecimento para manutenção de informação relativa ao ambiente no qual se inserem os sistemas auto-adaptáveis.

A segunda observação verificada nos trabalhos correlatos se refere a lacunas relacionadas ao desenvolvimento dos sistemas auto-adaptáveis. A primeira lacuna se refere a soluções específicas de adaptação, como, por exemplo, a abordagem RESIST (COORAY et al., 2010), que é um framework de adaptação especializado para o atributo de qualidade de serviço, confiabilidade, e o mais recente trabalho apresentado por Maimo et al. (2018), que é uma abordagem para auto-adaptação, que emprega a utilização de aprendizado de máquina para análise de tráfego de dados de redes móveis de quinta geração com gerenciamento dinâmico de recursos computacionais para atender a variações de demanda de análise de tráfego.

Outra lacuna verificada nos trabalhos correlatos, e que pode ter impacto na adoção de uma dessas abordagens no contexto de sistemas espaciais, é a impossibilidade da exploração de diferentes abordagens para auto-adaptação; esta lacuna, verificada nos trabalhos correlatos (BARESI; GUINEA, 2011) (ELKHODARY et al., 2010) (KRUPITZER et al., 2016) (GARLAN et al., 2004) (ESFAHANI et al., 2011) (COORAY et al., 2010) (POLADIAN et al., 2007) (MORIN et al., 2009) (ROSSI et al., 2017) (WANG; LI, 2016), pode limitar a sua adoção dependendo dos requisitos de operação autônoma definidos para cada missão.

Outra lacuna observada, de uma maneira geral, está no forte acoplamento entre os gerenciadores autônomos e a lógica de adaptação das abordagens, e este é outro fator que limita a sua adoção no contexto do EMBRACE, onde os instrumentos possuem capacidade de hardware limitada.

A última lacuna observada, que limita a aplicação de abordagens nos sistemas espaciais, é o suporte à adaptação "on-the-fly", característica esta muito importante, dada a possibilidade de modificação de elementos no contexto (ambiente) dos sistemas, como a inclusão de novos instrumentos, avaria de componentes, aumento de demanda por parte de usuários, entre outros aspectos que implicam em atualização dinâmica da base de conhecimento e reflexão destas alterações em tempo de execução, de modo que os sistemas continuem operando para manter os serviços prestados com qualidade.

Foram necessárias, ainda, pesquisas exploratórias sobre conceitos e tecnologias que podem ser aplicadas no desenvolvimento de sistemas auto-adaptáveis; essas pesquisas indicaram três caminhos para o desenvolvimento do trabalho: Qualidade de Serviço, Frameworks de Software e Inteligência Artificial.

De acordo com as pesquisas realizadas, foi proposto um framework de software, denominado Arctic Fox, para prover suporte auto-adaptativo aplicado às necessidades dos sistemas espaciais de segmento solo, com o intuito de resolver as lacunas supracitadas. Foi definida uma arquitetura de software cujos componentes estão distribuídos em três camadas. Para a arquitetura proposta, foram definidos componentes para coleta, recepção e cálculo de métricas de qualidade de serviço; componentes para gerenciamento do ciclo de adaptação, responsáveis pela análise, planejamento e execução de planos de adaptação nos componentes do sistema solo; além de componentes de apoio para gerenciamento da base de conhecimento do *framework* e segurança.

A arquitetura do *framework* Arctic Fox inova ao prover um mecanismo de extensão, que permite suporte ao desenvolvimento e execução de múltiplas heurísticas para análise de dados provenientes do contexto do sistema solo. Esta característica permite que o framework possa evoluir para atender a diferentes necessidades de adaptação verificadas nos requisitos das missões espaciais.

A arquitetura proposta também inova sob outro aspecto: o desacoplamento entre o Gerenciador Autônomo e a Lógica de Adaptação. Este desacoplamento funcional permite a alocação de recursos computacionais para atividades de análise que podem demandar a utilização de CPU e memória, ao mesmo tempo em que componentes de sistemas espaciais, como instrumentos com capacidade limitada de hardware, podem receber agentes de software com funções específicas de coleta de dados e execução de planos de adaptação.

Para validar os componentes especificados na arquitetura proposta para o framework Arctic Fox, foi desenvolvido um protótipo com os componentes necessários para a execução do ciclo de adaptação. O framework foi desenvolvido sob a plataforma JEE (SHANNON et al., 2000), e foi necessária a utilização do *framework* Jboss Drools (JBOSS, 2017) para desenvolvimento do motor de inferência, que analisa os dados de métricas de Qualidade de Serviço para a produção dos planos de adaptação. Este conjunto de tecnologias permitiu o suporte à adaptação *"on-the-fly"*, o desacoplamento lógico entre o gerenciador autônomo e a lógica de adaptação.

Com o propósito de demonstrar a capacidade de auto-adaptação do framework prototipado, foi elaborado um estudo de caso aplicado ao serviço GTex utilizado pelo EMBRACE, para a produção de mapas de cintilação da ionosfera, serviço este que fornece dados de estações de GPS para as aplicações de clima espacial. O estudo de caso proposto avaliou se houve ganho de disponibilidade, em um cenário simulado de aumento de volume de carga aplicado ao serviço Gtex com o uso do framework e sem o uso do framework. A lógica de adaptação implementada para o estudo de caso continha duas táticas: a primeira para ativar três novos nós, Cachoeira Paulista, Santa Maria e Natal, de maneira a atender à nova demanda e a segunda para re-distribuição da carga para os novos nós ativados.

Foi verificado ganho para disponibilidade do serviço GTex, haja vista o resultado de 100% de sucesso para retornos provenientes do framework Arctic Fox ante a uma taxa de sucesso próxima a 94,10% do simulador Gtex sem o uso do framework. No entanto, foram constatadas outras observação, como a diferença elevada de tempo médio de resposta pelo framework, degradação severa do serviço Gtex sem uso de

adaptação e alta latência.

Em resumo, o framework Arctic Fox apresentou-se adequado para as necessidades de adaptação verificadas na arquitetura do Clima Espacial; de uma maneira geral, o framework tem potencial para aplicação em outros sistemas e pode auxiliar o INPE a operar os sistemas num cenário de aumento de complexidade, dadas as novas missões baseadas no uso de nano satélites e parcerias com outras agência espaciais e instituições, a exemplo do ocorrido com o EMBRACE.

8.1 Comparativo entre Arctic Fox e outras abordagens

Conforme descrito nos parágrafos supracitados, o *framework* Arctic Fox foi concebido de maneira a resolver as lacunas observadas nos trabalhos correlatos. Para evidenciar esta assertiva, faz-se necessária a comparação do *framework* Arctic Fox com as outras abordagens, seguindo os critérios apresentados na tabela 8.1.

Tabela 8.1 - Descrição de Características de Sistema Auto-Adaptáveis

Característica	**Descrição**
Adaptação "On the Fly"	Capacidade do sistema auto-adaptável de alterar seu comportamento em tempo de execução, isto é, critérios de adaptação que não foram definidos em tempo de projeto e podem ser parametrizados durante a operação do sistema.
Descoberta de Componentes	Capacidade de detectar um novo componente ou a alteração de um existente e registrar essa informação em base de dados de conhecimento.
Descoberta de Nós	Capacidade de detectar um novo dispositivo ou a alteração de um existente e registrar essa informação em base de dados de conhecimento.
Detecta degradação arquitetural	Capacidade de manter informações sobre a topologia do sistema-alvo consistente, isto é, informação atualizada sobre componentes, versões e instalações em dispositivos.
MAPE + AL desacoplados	Gerenciador autonômo e lógica de adaptação estão desacoplados, podendo inclusive estar instalados em máquinas/dispositivos diferentes.
MAPE + AL acoplados	Gerenciador autonômico e Lógica de adaptação estão acoplados
Suporte a múltiplas heurísticas de adaptação	Permite alterar a estratégia de adaptação em tempo de execução, por exemplo, alterar a implementação do Gerenciador Autônomo, baseado em regras, para outro que utilize algoritmo evolutivo.
Processo de Desenvolvimento de Software Adaptável	Prevê um processo de desenvolvimento de sistemas auto-adaptáveis.
Aplicável a qualquer classe de sistemas?	Verifica se a abordagem informada pode ser replicada a qualquer domínio de aplicação e plataforma tecnológica.
Distribuição de gerenciadores de adaptação entre nós do sistema alvo?	Verifica se há necessidade de instalação de gerenciadores autônomos em diversos nós do sistema, necessidade esta que pode limitar a solução devido a requisitos de HW para execução dos gerenciadores.
Ajustável a todas as auto-propriedades?	Verifica se a abordagem informada pode ser ajustada para todas as auto-propriedades, mesmo que tenha suporte inicial para uma ou mais.

Fonte: Autor

Na tabela 8.2, é apresentado um comparativo entre o *framework* Arctic Fox, listado na segunda coluna, e as outras abordagem apresentadas na seção 3.2.5.

A intenção desta comparação está em procurar determinar as características do framework e confrontá-las com as características das outras abordagens, de modo a determinar os prós e contras e, desta maneira, apresentar as contribuições para o Estado da Arte e trabalhos futuros.

A tabela em questão procura apenas informar se determinada abordagem possui suporte à característica listada na coluna 1, utilizando-se das palavras sim ou não.

Prós em relação a maioria das abordagens:

- Adaptação "On the fly";
- Possibilidade de implementação de novas estratégias de adaptação;
- Descoberta de dispositivos e componentes de software;
- Pode ser ajustada a todas as auto-propriedades;
- Permite adaptação a elementos com pouco ou nenhum poder de processamento (caso de agente não embarcado).

Contras em relação à maioria das abordagens:

- Adaptação distribuída ainda não muito bem resolvida. O framework A3, por exemplo, possui a característica de se auto-organizar baseado nos dados de descoberta dos elementos de software, característica não suportada pelo Arctic Fox;
- Não há ainda definição de processo de desenvolvimento para sistemas auto-adaptáveis tendo o Arctic Fox como elemento central do processo.
- Não há mecanismos para re-calibragem da base de conhecimento baseado em aprendizado a exemplo do framework FUSION e do framework Multi-Agentes.

8.2 Contribuições

Como contribuições deste trabalho, destacam-se:

Tabela 8.2 - Comparativo ArcticFox x Outras Abordagens de Adaptação

Característica	**Arctic Fox**	**A3**	**FESAS**	**FUSION**	**Rainbow**	**FLAGS**	**RESIST**	**POISED**	**ADC**	**DH-SAS***	**SAA-NI5G****	**Multi-agent**
Adaptação "On the Fly"	Sim	Sim	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim
Descoberta de Componentes	Sim	Sim	Não	Não	Sim	Não	Não	Não	Não	Não	Não	Não
Descoberta de Nós	Sim	Sim	Não	Não	Sim	Não	Não	Não	Não	Não	Não	Não
Detecta degradação arquitetural	Sim	Sim	Não	Não	Não	Não	Sim	Não	Não	Não	Não	Não
MAPE + AL descentralizada	Sim	Não	Não	Não	Sim	Não	Sim	Não	Não	Não	Não	Não
MAPE + AL centralizada	Não	Sim	Sim	Sim	Sim	Sim	Não	Sim	Não	Sim	Sim	Sim
Suporte a múltiplas estratégias de adaptação	Sim	Não	Não	Não	Não	Não	Não	Não	Não	Não	Não	Não
Processo de Desenvolvimento de Sw Adaptável	Não	Não	Sim	Não	Não	Sim	Não	Sim	Não	Não	Não	Não
Aplicável a qualquer classe de sistemas?	Sim	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Não	Sim
Distribuição de gerenciadores de adaptação entre nós do sistema alvo?	Não	Sim	Sim	Não	Não	Não	Não	Não	Não	Não	Não	Não
Ajustável a todas as auto-propriedades?	Sim	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Não	Sim

Fonte: Autor

[0]Framework Multiagent-based Framework with Search-based Optimization (WANG; LI, 2016)
[0]*DH-SAS = Abreviação para Dynamic high-level in Self-Adaptive Systems (ROSSI et al., 2017)
[0]**SAA-NI5G = Self-Adaptive Architecture for Network Inspection in 5G (MAIMO et al., 2018)

a) Pesquisa e definições de conceitos sobre os sistemas espaciais, com atualização tecnológica para novas missões, que utilizam constelações e mega constelações.

b) Pesquisa e definições de conceitos sobre os sistemas auto adaptáveis, tais como elementos de arquitetura de software, modelos para auto adaptação, autopropriedades como autoconfiguração, autoreparação e autocontenção, trabalhos correlatos e contextualização para aplicação dos conceitos em sistemas de segmento solo.

c) Pesquisa sobre conceitos e tecnologias que podem ser empregadas no desenvolvimento de sistemas auto adaptaveis como: qualidade de serviço, acordos de nível de serviço, processamento de grandes volumes de dado, frameworks de software, internet das coisas (IoT) e inteligência artificial.

d) Uma arquitetura para um framework que provê suporte auto adaptável e que leva em consideração os aspectos do domínio de sistemas espaciais de segmento solo.

e) Arquitetura de um ambiente para monitoramento de métricas de qualidade de serviço (QoS).

f) Proposição de conceitos para estabelecimento de uma base de conhecimento, baseada no conhecimento da estrutura do sistema (Conhecimento Contextual) e do comportamento do sistema (Conhecimento Situacional). A base de conhecimento foi concebida para manter conhecimento de quatro domínios de conhecimento: Componentes e Dispositivos, Qualidade de Serviço, Segurança e Elementos Gerenciados pelo Framework.

g) Modificação do modelo MAPE-K para prover suporte a múltiplas heurísticas, desacoplamento funcional entre o gerenciador autônomo e a lógica de adaptação, suporte a adaptação on-the-fly e aquisição dinâmica de dados relacionados a estrutura e situação dos componentes do sistema solo.

h) Uma implementação do framework em JEE para validação dos componentes definidos na arquitetura proposta para o framework Arctic Fox.

i) Exercício do uso do framework em um experimento para o contexto do EMBRACE, com avaliação da qualidade do serviço prestado em um cenário de alta carga aplicada e também a avaliação da consequência para a daptação para o desempenho do serviço. Arquitetura de um ambiente para monitoramento de métricas de qualidade de serviço (QoS).

j) Proposição de conceitos para estabelecimento de uma base de conhecimento, baseada no conhecimento da estrutura do sistema (Conhecimento Contextual) e do comportamento do sistema (Conhecimento Situacional). A base de conhecimento foi concebida para manter conhecimento de quatro domínios de conhecimento: Componentes e Dispositivos, Qualidade de Serviço, Segurança e Elementos Gerenciados pelo Framework.

k) Modificação do modelo MAPE-K para prover suporte a múltiplas heurísticas, desacoplamento funcional entre o gerenciador autônomo e a lógica de adaptação, suporte a adaptação on-the-fly e aquisição dinâmica de dados relacionados a estrutura e situação dos componentes do sistema solo.

l) Uma implementação do framework em JEE para validação dos componentes definidos na arquitetura proposta para o framework Arctic Fox.

m) Exercício do uso do framework em um experimento para o contexto do EMBRACE, com avaliação da qualidade do serviço prestado em um cenário de alta carga aplicada e também a avaliação da consequência para a daptação para o desempenho do serviço.

8.3 Trabalhos Futuros

Como trabalhos futuros pode-se destacar o que segue:

- Exploração de Tecnologias de Inteligência Artificial para cenários mais complexos de adaptação, como uso de redes neurais para cenários de autocontenção, por exemplo;

- Aplicação do framework em projetos reais;

- Exploração de auto-propriedades que envolvam aprendizado como autoimunidade e autocontenção;

- Extensão do framework para suporte a outros protocolos de comunicação;

- Aprimoramento do Componente de Qualidade de Serviço (QoS) para coleta e processamento de volume massivo de dados e consequente exploração de tecnologias Big Data, Stream Analytics e Data Science.

8.4 Limitações do Trabalho

O protótipo foi avaliado em condições simuladas de um cenário hipotético do clima espacial e, desta maneira, não foi possível avaliar o protótipo em condições reais de operação, que possui diferentes realidades de infraestrutura e variáveis que podem produzir um comportamento diferente do observado no estudo de caso.

Outra limitação é a aplicação do framework em um sistema baseado em arquitetura orientado a serviços, caso do EMBRACE, onde os agentes de software implementados, que tratam de coleta de métricas e execução de adaptação, são especializados neste tipo de arquitetura, o que pode limitar a adoção do framework em outros tipos de sistemas de segmento solo.

Por último, não foram implementados todos os componentes da arquitetura proposta; como exemplo, citam-se os componentes responsáveis pela segurança das aplicações e o mecanismo de descoberta de componentes e dispositivos. Estes componentes devem ser explorados em outros trabalhos.

REFERÊNCIAS BIBLIOGRÁFICAS

ADVANCED OPEN STANDARDS FOR THE INFORMATION SOCIETY (OASIS). Reference model for service oriented architecture 1.0. 2006. Disponível em: <https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>. Acesso em: 05 Jul. 2018. 45

AIRBUS SPACE & DEFENSE. **OneWeb Satellites completes its industrial organisation**. 2016. Press Release. Disponível em: <http://www.airbus.com/space/telecommunications-satellites/oneweb-satellites-connection-for-people-all-over-the-globe.html>. Acesso em: 30 Jun. 2018. 1, 17

AKMEL, F.; BIRHANU, E.; SIRAJ, B.; SHIFA, S. A comparative analysis on software architecture styles. **International Journal in Foundations of Computer Science & Technology**, v. 7, n. 5/6, p. 11–22, nov 2017. 30

ALUR, D.; MALKS, D.; CRUPI, J. **Core J2EE patterns:**. best practices and design strategies, 2. Prentice Hall, 2003. ISBN 0133807460. Disponível em: <https://www.amazon.com/Core-J2EE-Patterns-paperback-Strategies/dp/0133807460?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0133807460>. Acesso em: 30 Jun. 2018. 110

APACHE FOUNDATION. **Apache Flink open source platform for distributed stream and batch data processing**. 2018. Disponível em: <https://flink.apache.org/>. Acesso em: 30 Jun. 2018. 67

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **Sistemas de gestão de qualidade-requisitos. Quality management systems-requirements.** Rio de Janeiro, 2008. 59

BAKALOS, N.; KYRIAZIS, D.; PROTONOTARIOS, E.; VARVARIGOU, T.; BARRETO, O. **SLA specification and reference model-a D3.2**. 2015. European Commission: Slalom. Disponível em: <https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5a07549af&appId=PPGMS>. Acesso em: 02 Jul. 2018. 61

BARESI, L.; GUINEA, S. A3: Self-adaptation capabilities through groups and coordination. In: INDIA SOFTWARE ENGINEERING CONFERENCE ON - ISEC '11, 4. **Proceedings...** [S.l.]: ACM Press, 2011. 47, 77, 144

_____. Architectural styles for adaptive systems: a tutorial. In: IEEE INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS, 6. **Proceedings...** [S.l.]: IEEE, 2012. 39, 41

BARESI, L.; NITTO, E. D.; GHEZZI, C. Toward open-world software: issues and challenges. **Computer**, v. 39, n. 10, p. 36–43, oct 2006. 37

BARESI, L.; PASQUALE, L.; SPOLETINI, P. Fuzzy goals for requirements-driven adaptation. In: IEEE INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE, 18. **Proceedings...** [S.l.]: IEEE, 2010. 52, 80

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice**. Addison-Wesley Professional, 2003. Disponível em: <https://www.amazon.com/Software-Architecture-Practice-Len-Bass-ebook/dp/B002L9MZ1U?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B002L9MZ1U>. Acesso em: 30 Jun. 2018. 2, 32, 33, 34, 60

BASSI, A.; BAUER, M.; FIEDLER, M.; KRAMP, T.; KRANENBURG, R. van; LANGE, S.; MEISSNER, S. **Enabling things to talk**: designing iot solutions with the iot architectural reference model. [S.l.]: Springer, 2016. ISBN 3662524945, 9783662524947. 69

BERNS, A.; GHOSH, S. Dissecting self properties. In: IEEE INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS, 3. **Proceedings...** [S.l.]: IEEE, 2009. 3, 36

CAMPOS, V. F. **TQC: controle da qualidade total**. Belo Horizonte: [s.n.], 1992. 44, 59

CARVALHO, M. J. M. d.; LIMA, J. S. d. S.; JOTHA, L. d. S.; AQUINO, P. S. d. Conasat - constelação de nano satélites para coleta de dados ambientais. In: EPIPHANIO, J. C. N.; GALVÃO, L. S. (Ed.). **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE), 2013. p. 9108–9115. ISBN 978-85-17-00066-9 (Internet) and 978-85-17-00065-2 (DVD). Disponível em: <http://urlib.net/dpi.inpe.br/marte2/2013/05.29.00.50.13>. Acesso em: 11 maio 2018. 1

CERVANTES, H.; KAZMAN, R. **Designing software architectures**: a practical approach. Upper Saddle River: Pearson, 2016. ISBN 0134390784. Disponível em: <http://www.ebook.de/de/product/25420109/designing_software_arch_practice_approac.html>. Acesso em: 30 Jun. 2018. 27

CHOI, S. W.; HER, J. S.; KIM, S. D. Qos metrics for evaluating services from the perspective of service providers. In: IEEE INTERNATIONAL CONFERENCE ON E-BUSINESS ENGINEERING ICEBE. **Proceedings...** [S.l.]: IEEE, 2007. p. 622–625. 32, 59, 60, 125, 126

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS (CCSDS). **Space packet protocol**. 2003. Disponível em: <https://public.ccsds.org/Pubs/133x0b1c2.pdf>. Acesso em: 30 Jun. 2018. 76

COORAY, D.; MALEK, S.; ROSHANDEL, R.; KILGORE, D. RESISTing reliability degradation through proactive reconfiguration. In: IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING - ASE, 10. **Proceedings...** [S.l.]: ACM Press, 2010. 53, 54, 57, 143, 144

ELKHODARY, A.; ESFAHANI, N.; MALEK, S. Fusion: a framework for engineering self-tuning self-adaptive software systems. In: ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 18. **Proceedings...** ACM, 2010. Disponível em: <http://dl.acm.org/citation.cfm?id=1882296>. Acesso em: 30 Jun. 2018. 41, 49, 50, 144

ESFAHANI, N.; KOUROSHFAR, E.; MALEK, S. Taming uncertainty in self-adaptive software. In: ACM SIGSOFT SYMPOSIUM AND THE 13TH EUROPEAN CONFERENCE ON FOUNDATIONS OF SOFTWARE ENGINEERING, 19. **Proceedings...** ACM, 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=2025147>. Acesso em: 30 Jun. 2018. 53, 144

EUROPEAN COMMISSION. **Cloud computing service level agreements**. 2013. Disponível em: <http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=2496>. Acesso em: 30 Jun. 2018. 61

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **Space engineering: policy and principles**. Noordwijk, The Netherlands: ECSS, 1996. 8, 9, 10

_____. **Space engineering: ground systems and operations**. Noordwijk, The Netherlands: ECSS, 2008. 1, 10, 11, 15

EUROPEAN SPACE AGENCY (ESA). **ESOC ESA Spacecraft Operations Centre (BR-265)**. 2006. Disponível em: <http://www.esa.int/esapub/br/br265/br265.pdf>. Acesso em: 30 Jun. 2018. 11, 12, 14

FLANAGAN, D. **Java in a nutshell**. 5. ed. O'Reilly Media, 2005. ISBN 0596007736. Disponível em: <https://www.amazon.com/Java-Nutshell-5th-David-Flanagan/dp/0596007736?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0596007736>. Acesso em: 30 Jun. 2018. 100

GAMMA, E.; HELM, R.; JOHNSON, R. **Design patterns**. Pearson Technology Group, 1994. Disponível em: <http://www.ebook.de/de/product/24740039/erich_gamma_richard_helm_ralph_johnson_design_patterns.html>. Acesso em: 30 Jun. 2018. 100, 109

GARLAN, D.; CHENG, S.-W.; HUANG, A.-C.; SCHMERL, B.; STEENKISTE, P. Rainbow: architecture-based self-adaptation with reusable infrastructure. **Computer**, v. 37, n. 10, p. 46–54, oct 2004. 51, 52, 144

GARLAN, D.; MONROE, R. T.; WILE, D. Acme: architectural description of component-based systems. In: LEAVENS G. T.; SITARAMAN, M. (Ed.). **Foundations of component-based systems**. New York, NY, USA: Cambridge University Press, 2000. p. 47–67. ISBN 0-521-77164-1. Disponível em: <http://dl.acm.org/citation.cfm?id=336431.336437>. Acesso em: 30 Jun. 2018. 2, 31, 32

GARTNER INC. **Gartner internet of things prediction**. 2015. Disponível em: <http://www.gartner.com/newsroom/id/3165317#>. Acesso em: 30 Jun. 2018. 65

GOUSIOS, G.; SAFARIC, D.; VISSER, J. Streaming software analytics. In: INTERNATIONAL WORKSHOP ON BIG DATA SOFTWARE ENGINEERING, 2. **Proceedings...** [S.l.], 2016. 67

GRAYBILL, F. A.; IYER, H. K.; BURDICK, R. K. **Applied statistics**: a first course in inference. Prentice Hall, 1997. ISBN 0136214673. Disponível em: <https://www.amazon.com/Applied-Statistics-First-Course-Inference/dp/0136214673?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0136214673>. Acesso em: 30 Jun. 2018. 133

GROSAN, C.; ABRAHAM, A. **Intelligent system**. Springer-Verlag GmbH, 2011. ISBN 3642210031. Disponível em: <http://www.ebook.de/de/product/14922018/crina_grosan_ajith_abraham_intelligent_systems.html>. Acesso em: 30 Jun. 2018. 70, 71

GUINEA, S.; SAEEDI, P. Coordination of distributed systems through self-organizing group topologies. In: ICSE WORKSHOP ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS (SEAMS). **Proceedings...** [S.l.]: IEEE, 2012. 40, 47, 48, 49, 77

HILL, E. F. **Jess in action**: java rule-based systems. Manning pubn, 2003. ISBN 1930110898. Disponível em: <http://www.ebook.de/de/product/4013611/ernest_friedman_hill_ernest_friedman_hill_jess_in_action_java_rule_based_systems.html>. Acesso em: 30 Jun. 2018. 71

HINCHEY, M. G.; RASH, J. L.; TRUSZKOWSKI, W. F.; ROUFF, C. A. Autonomous and autonomic swarms. In: AUTONOMIC & AUTONOMOUS SPACE EXPLORATION SYSTEMS (A&A-SES-1) AT 2005 INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING RESEARCH AND PRACTICE (SERP'05). **Proceedings...** Las Vegas, NV: CREA Press, 2005. p. 27–30. 1, 23, 24

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **IEEE. Recommended practice for architectural description of software-intensive systems**. [S.l.]: IEEE, 2000. 27

ITIL. **Glossary of terms definitions and acronyms**. 2007. Disponível em: <https://www.axelos.com/Corporate/media/Files/Glossaries/ITIL_2011_Glossary_BR-PT-v1-0.pdf>. Acesso em: 19 Jan. 2017. 60

JBOSS, D. T. **Drools business rules management system reference manual**. [s.n.], 2017. Disponível em: <https://docs.jboss.org/drools/

release/7.0.0.Final/drools-docs/html_single/index.html>. Acesso em: 30 Jun. 2018. 71, 99, 107, 145

KEPHART, J.; CHESS, D. The vision of autonomic computing. **Computer**, v. 36, n. 1, p. 41–50, jan 2003. 3, 42, 43, 44, 77, 143

KITAJIMA, S.; ABE, S.; HANADA, T.; KAWAMOTO, S. Influences of mega constellations on the orbital environment. In: INTERNATIONAL ASTRONAUTICAL CONGRESS (IAC), 67. **Proceedings...** Guadalajara, Mexico: International Astronautical Federation, 2016. Disponível em: <https://iafastro.directory/iac/proceedings/IAC-16>. Acesso em: 30 Jun. 2018. 1, 17, 18

KLEIN, J.; BUGLAK, R.; BLOCKOW, D.; WUTTKE, T.; COOPER, B. A reference architecture for big data systems in the national security domain. In: INTERNATIONAL WORKSHOP ON BIG DATA SOFTWARE ENGINEERING, 2. **Proceedings...** [S.l.], 2016. 66

KRUPITZER, C.; ROTH, F. M.; BECKER, C.; WECKESSER, M.; LOCHAU, M.; SCHURR, A. Fesas ide: an integrated development environment for autonomic computing. In: IEEE INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING (ICAC). **Proceedings...** [S.l.]: Institute of Electrical and Electronics Engineers (IEEE), 2016. 3, 50, 144

KRUPITZER, C.; VANSYCKEL, S.; BECKER, C. FESAS: Towards a framework for engineering self-adaptive systems. In: INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS, 7. **Proceedings...** [S.l.]: IEEE, 2013. 44, 50, 57

LARSON, W. J.; WERTZ, J. R. **Space mission analysis and design**. 3. ed. [S.l.: s.n.], 1999. 1, 8, 10

LI, C. S. Z. Autonomic computing for spacecraft ground systems. In: IEEE INTERNATIONAL CONFERENCE ON SPACE MISSION CHALLENGES FOR INFORMATION TECHNOLOGY (SMC-IT'06), 2. **Proceedings...** [S.l.]: IEEE, 2006. 22

LUDWIG, H.; KELLER, A.; DAN, A.; KING, R. P.; FRANCK, R. **Web service level agreement (WSLA) language specification**. [S.l.: s.n.], 2003. 815–824 p. 61, 62

MAGEE, J.; KRAMER, J. Dynamic structure in software architectures. In: ACM SIGSOFT SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING - SIGSOFT '96, 4. **Proceedings...** [S.l.]: ACM Press, 1996. 31, 32

MAIMO, L. F.; GOMEZ, A. L. P.; CLEMENTE, F. J. G.; PEREZ, M. G.; PEREZ, G. M. A self-adaptive deep learning-based system for anomaly detection in 5g networks. **IEEE Access**, v. 6, p. 7700–7712, 2018. 55, 143, 148

MATIELLO-FRANCISCO, M. d. F. **Sistemas computacionais em aplicações espaciais**. INPE, 2003. Disponível em: <http://urlib.net/sid.inpe.br/jeferson/2003/10.13.15.25>. Acesso em: 30 Jun. 2018. 14

MITCHELL, M. **An introduction to genetic algorithms**. MIT PR, 1998. ISBN 0262631857. Disponível em: <http://www.ebook.de/de/product/3240124/melanie_mitchell_an_introduction_to_genetic_algorithms.html>. Acesso em: 30 Jun. 2018. 70

MORAES, M. P. S. **Uma arquitetura para avaliação de métricas de qualidade de serviço de sistema de segmento solo e de seus componentes**. 157 p. Dissertação (Mestrado em Engenharia e Gerenciamento de Sistemas Espaciais) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 03 2017. 116

MORIN, B.; BARAIS, O.; JEZEQUEL, J.-M.; FLEUREY, F.; SOLBERG, A. Models@ run.time to support dynamic adaptation. **Computer**, Institute of Electrical and Electronics Engineers (IEEE), v. 42, n. 10, p. 44–51, oct 2009. 55, 144

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). **NASA Systems Engineering Handbook**. [S.l.]: National Aeronautics and Space Administration (NASA), 2007. 7

OREIZY, P.; GORLICK, M.; TAYLOR, R.; HEIMHIGNER, D.; JOHNSON, G.; MEDVIDOVIC, N.; QUILICI, A.; ROSENBLUM, D.; WOLF, A. An architecture-based approach to self-adaptive software. **IEEE Intelligent Systems**, v. 14, n. 3, p. 54–62, may 1999. 3, 34, 38, 40, 57, 80, 94

ORLANDO, V.; KUGA, H. K. Os satélites scd1 e scd2 da missão espacial completa brasileira - mecb. In: FÍSICA, L. da (Ed.). **A Conquista do Espaço do Sputnik a Missão Centenário**. [S.l.]: Agência Espacial Brasileira, 2007. p. 151–176. ISBN 978-85-88325-89-0. 1

PALERMO, G.; GAUDENZI, P. Progressive deployment of a leo constellation providing support services to leo client satellites: a trade-off analysis. In: IAC 67th I. A. C. (Ed.). **Proceedings...** Guadalajara, Mexico: International Astronautical Federation, 2016. Disponível em: <https://iafastro.directory/iac/proceedings/IAC-16>. Acesso em: 30 Jun. 2018. 19

PIRES, P. F.; DELICATO, F. C.; BATISTA, T.; BARROS, T.; CAVALCANTE, E.; PITANGA, M. Plataformas para a internet das coisas. In: SBRC-SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS. **Anais...** [S.l.], 2015. 68

POLADIAN, V.; GARLAN, D.; SHAW, M.; SATYANARAYANAN, M.; SCHMERL, B.; SOUSA, J. Leveraging resource prediction for anticipatory dynamic configuration. In: INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS (SASO 2007). **Proceedings...** [S.l.]: IEEE, 2007. 53, 144

RIEHLE, D. **Framework design: a role modeling approach**. 229 p. Tese (Doutorado em Ciência da Computação) — Swiss Federal Institute of Technology Zurich, Zurique, Suiça, 2000. 67, 73

ROSSI, D.; POGGI, F.; CIANCARINI, P. Dynamic high-level in self-adaptive systems. In: INTERNATIONAL CONFERENCE ON RELIABILITY, INFOCOM TECHNOLOGIES AND OPTIMIZATION (TRENDS AND FUTURE DIRECTIONS) (ICRITO), 6. **Proceedings...** [S.l.]: IEEE, 2017. 54, 144, 148

RUSSELL, S.; NORVIG, P. **Artificial intelligence**: a modern approach. Prentice Hall, 2009. ISBN 0136042597. Disponível em: <http://www.ebook.de/de/product/9052581/stuart_russell_peter_norvig_artificial_intelligence_a_modern_approach.html>. Acesso em: 30 Jun. 2018. 69

SABRY, A. E. Decision model for software architectural tactics selection based on quality attributes requirements. **Procedia Computer Science**, v. 65, p. 422–431, 2015. 32, 60

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: landscape and research challenges. **ACM Transactions on Autonomous and Adaptive Systems**, p. 14–55, 2009. 3, 34, 35, 42, 44

SANTANNA, N.; GUERRA, E.; IVO, A.; PEREIRA, F.; MORAES, M.; GOMES, V.; VERAS, L. G. Modelo arquitetural para coleta, processamento e visualização de informações de clima espacial. In: AO, S. B. de Sistemas de I. (Ed.). **Anais...** Londrina, PR: [s.n.], 2014. p. 125–136. 2, 15, 123, 124

SAUNDERS, C.; LOBB, D.; SWEETING, M.; GAO, Y. Building large telescopes in orbit using small satellites. In: INTERNATIONAL ASTRONAUTICAL CONGRESS (IAC), 67. **Proceedings...** Guadalajara, Mexico: International Astronautical Federation, 2016. Disponível em: <https://iafastro.directory/iac/proceedings/IAC-16>. Acesso em: 30 Jun. 2018. 1, 20, 21

SHADE, O. K.; FRANK, I.; AWODELE, O.; SAMUEL, O. O. Quality of service (qos) issues in web services. **IJCSNS International Journal of Computer Science and Network Security**, v. 12, n. 1, p. 94–97, 2012. 32, 59

SHANNON, B.; HAPNER, M.; MATENA, V.; DAVIDSON, J.; PELEGRI-LLOPART, E.; CABLE, L. et al. **Java 2 platform, enterprise edition:**. platform and component specifications. [S.l.]: Addison-Wesley Reading, MA, 2000. 99, 145

SOMMERVILLE, I. **Engenharia de software**. [S.l.]: Pearson, 2011. ISBN 9788579361081. 59

STEEL, R.; HOFFMANN, A.; NIEZETTE, M.; CIMATTI, A.; ROVERI, M.; KAPELLOS, K.; DONATI, A.; POLICELLA, N. Innovative rover operations concepts - autonomous planner (ironcap) - supporting rover operations planning on ground. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS, SPACEOPS, 12. **Proceedings...** [S.l.], 2012. 23

TAKAHASHI, H.; COSTA, J. E. R.; LAGO, A. D.; NARDIN, C. M. D.; PAULA, E. D.; SOUZA, J. R. D.; PADILHA, A. L.; SANTANNA, N.; GATTO, R. C.; MIRANDA, O. D. Space weather program in brazil. In: INTERNATIONAL SYMPOSIUM ON EQUATORIAL AERONOMY, 13. (ISEA)., 12 a 17/03/2012, Paracas, Peru. **Proceedings...** [S.l.], 2012. Acesso em: 14 jul. 2016. 2, 121

TAKAHASHI, H.; PADILHA, A. L.; SAWANT, H. S.; COSTA, J. E.; CECATTO, J. R.; GONZALEZ, W. D.; PAULA, E. de; VITORELLO, I.; VELHO, H. d. C.; SILVA, J. D. The first landmark of the brazilian space weather program. In: BRAZILIAN DECIMETRIC ARRAY WORKSHOP. **Proceedings...** São José dos Campos, 2008. 2, 15, 121

TANEJA, M.; DAVY, A. Poster abstract: resource aware placement of data stream analytics operators on fog infrastructure for internet of things applications. In: IEEE/ACM SYMPOSIUM ON EDGE COMPUTING (SEC). **Proceedings...** IEEE, 2016. Disponível em: <https://doi.org/10.1109/sec.2016.44>. Acesso em: 30 Jun. 2018. 65

TANENBAUM, A. S.; WETHERALL, D. J. **Computer networks**. 5. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN 0132126958, 9780132126953. 76

TAYLOR, R. N.; MEDVIDOVIC, N.; DASHOFY, E. M. **Software architecture:**. foundations, theory and practice, 1. John Wiley & Sons, 2009. ISBN 978-0470-16774-8. Disponível em: <http://www.ebook.de/de/product/7349169/r_n_taylor_nenad_medvidovic_eric_dashofy_software_architecture.html>. Acesso em: 02 Jul. 2018. 2, 27, 30, 31, 32, 34

TESSENYI, M.; TINETTI, G.; TENNYSON, J.; SAVINI, G.; PASCALE, E.; JASON, S.; CURIEL, A. da S.; WILLIAMS, J.; JOHNSON, G.; PRASAD, S.; VORA, A.; SAUNDERS, C.; FRIEND, J. Twinkle - a mission to unravel the story of planets in our galaxy. In: INTERNATIONAL ASTRONAUTICAL CONGRESS (IAC), 67. **Proceedings...** Guadalajara, Mexico: International Astronautical Federation, 2016. Disponível em: <https://iafastro.directory/iac/proceedings/IAC-16>. Acesso em: 30 Jun. 2018. 1, 19

THE OPEN GROUP. **Soa source book**. Van Haren Publishing, 2009. ISBN 9087535031. Disponível em: <http://www.ebook.de/de/product/8528338/van_haren_publishing_soa_source_book.html>. Acesso em: 30 Jun. 2018. 44

THIO, N.; KARUNASEKERA, S. Automatic measurement of a qos metric for web service recommendation. In: AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE. **Proceedings...** [S.l.], 2005. p. 202–211. 59

TU BERLIN AND HUMBOLDT UNIVERSITY AND THE HASSO PLATTNER INSTITUTE. **Stratosphere Big Data looks tiny from here**. 2018. Disponível em: <http://stratosphere.eu/>. Acesso em: 02 Jul. 2018. 67

VARGHESE, B.; MCKEE, G. Building reliable systems for space applications using swarm-array computing. In: COMPUTATION WORLD: FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS. **Proceedings...** [S.l.]: IEEE, 2009. 23

VIRGILIA, B. B.; KRAG, H.; LEWIS, H.; RADTKE, J.; ROSSIE, A. Mega-constellations, small satellites and their impact on the space debris environment. In: INTERNATIONAL ASTRONAUTICAL CONGRESS (IAC), 67. **Proceedings...** Guadalajara, Mexico: International Astronautical Federation, 2016. Disponível em: <https://iafastro.directory/iac/proceedings/IAC-16>. Acesso em: 30 Jun. 2018. 1, 17, 18

WANG, L.; LI, Q. A multiagent-based framework for self-adaptive software with search-based optimization. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME). **Proceedings...** [S.l.]: IEEE, 2016. 56, 144, 148

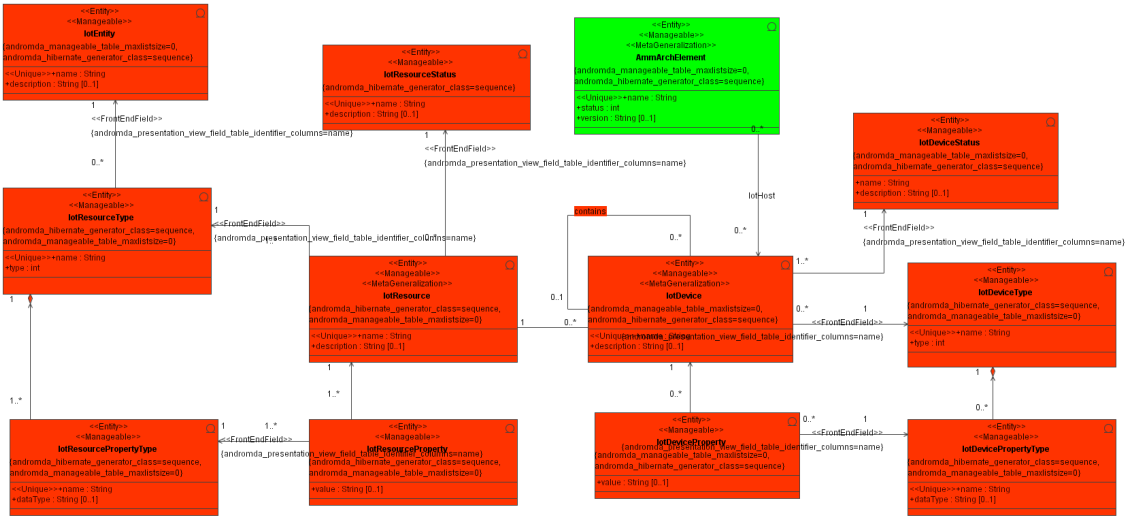
YODER, J. W.; BALAGUER, F.; JOHNSON, R. Architecture and design of adaptive object-models. **ACM SIGPLAN Notices**, v. 36, n. 12, p. 50–60, dez. 2001. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/583960.583966>. Acesso em: 30 Jun. 2018. 99, 111

APÊNDICE A - BASE DE CONHECIMENTO

Neste anexo são apresentados os diagramas de classes UML para os quatro domínios de conhecimento representados na Base de Conhecimento.

Na figura A.4 é apresentado um diagrama de classes UML que representa o domínio de informação de Dispositivos.

Figura A.1 - Diagrama de Classes UML: Dispositivos



Na figura A.4 é apresentado um diagrama de classes UML que representa o domínio de informação de Qualidade de Serviço (QoS).

Na figura A.4 é apresentado um diagrama de classes UML que representa o domínio de informação de Elementos de Arquitetura (Componentes).

Na figura A.4 é apresentado um diagrama de classes UML que representa o domínio de informação do Framework.

Figura A.2 - Diagrama de Classes UML: Qualidade de Serviço (QoS)

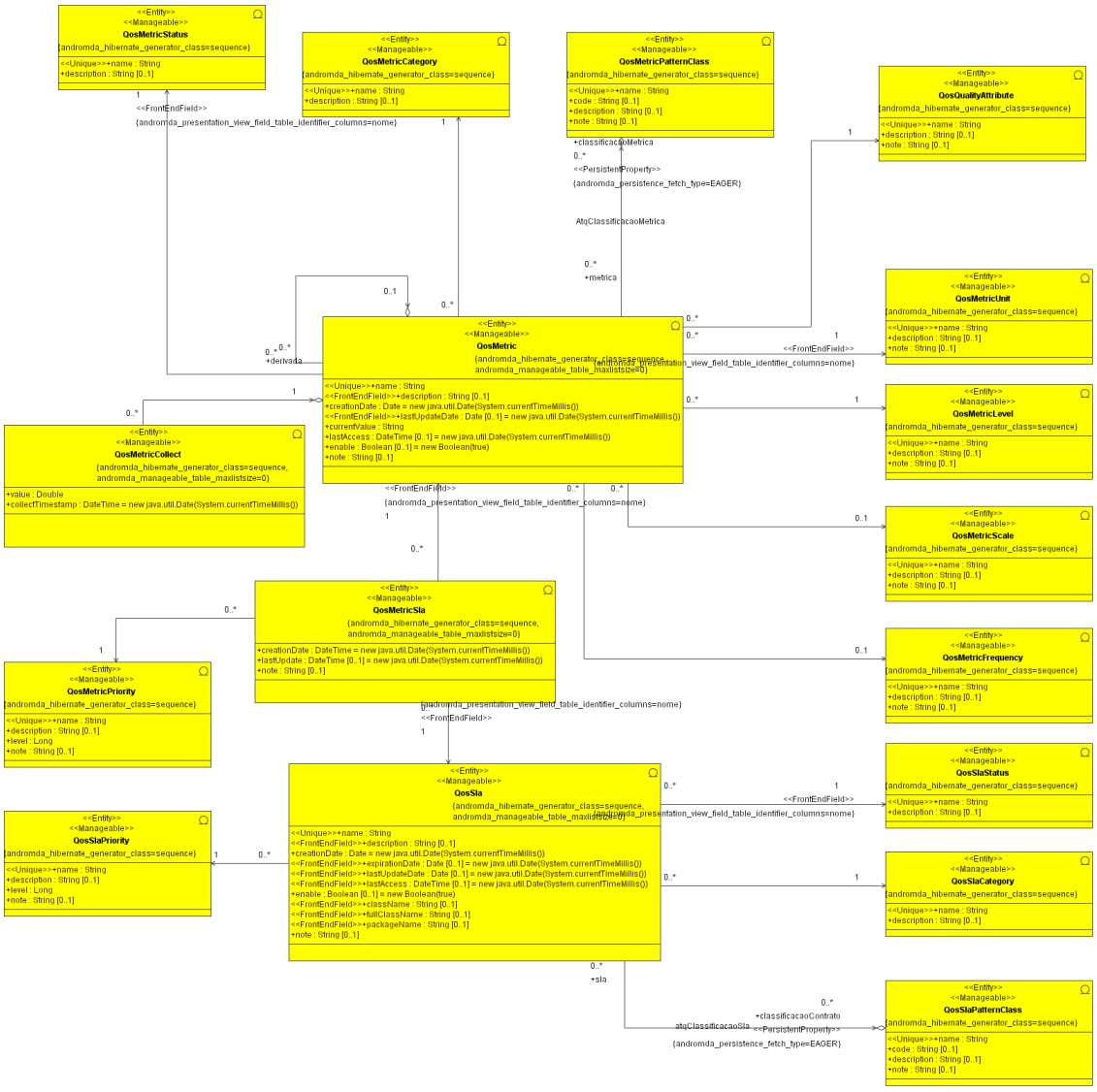


Figura A.3 - Diagrama de Classes UML: Elementos de Arquitetura (Componentes)

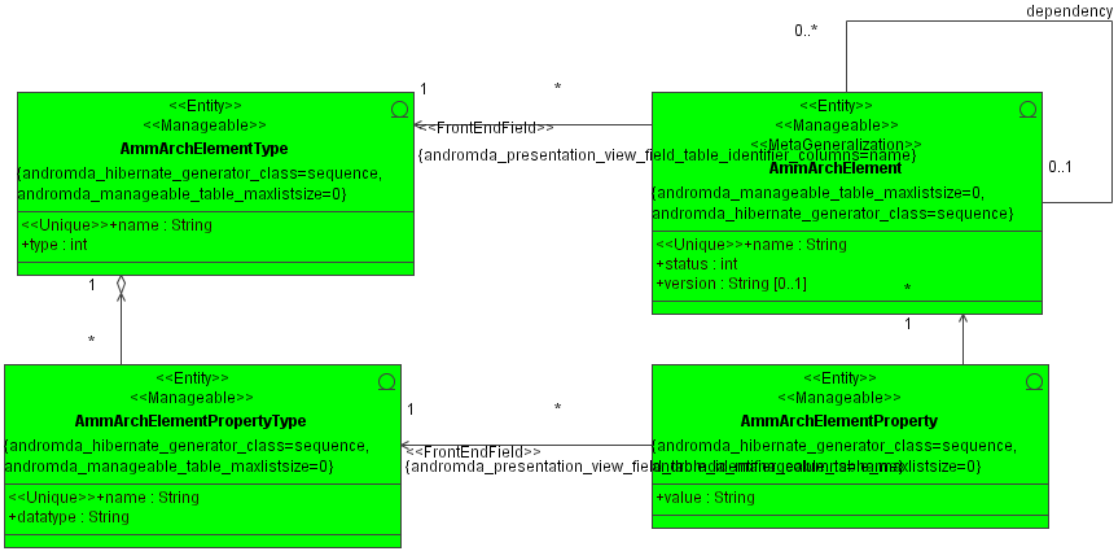
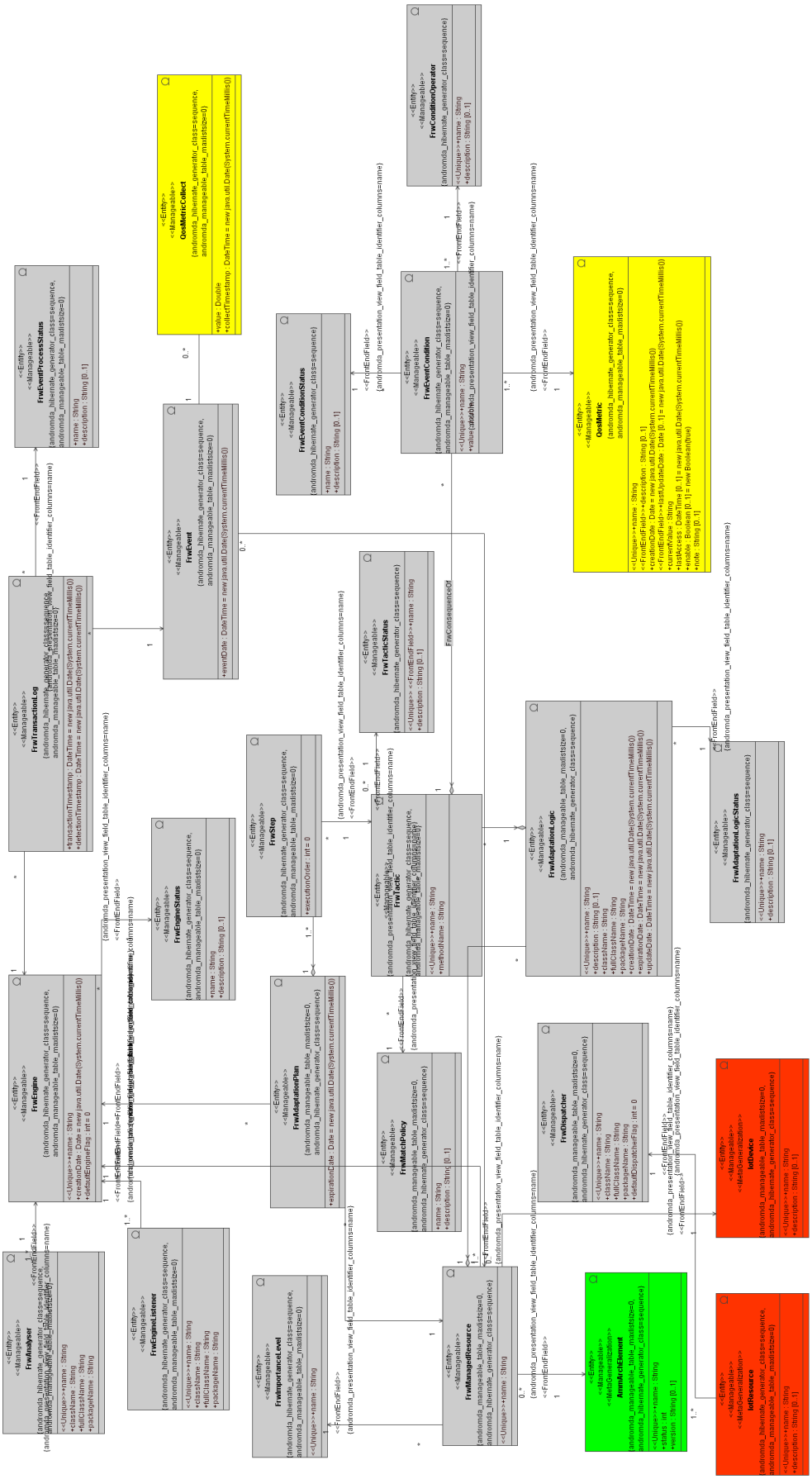


Figura A.4 - Diagrama de Classes UML: Framework



PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.