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

sid.inpe.br/mtc-m21b/2017/05.03.20.18-TDI

UMA ARQUITETURA PARA AVALIAÇÃO DE MÉTRICAS DE QUALIDADE DE SERVIÇO DE SISTEMA DE SEGMENTO SOLO E DE SEUS COMPONENTES

Marcos Paulo Salgueiro de Moraes

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Nilson Sant'Anna, aprovada em 28 de março de 2017.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3NQU9CL>

INPE
São José dos Campos
2017

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
E-mail: pubtc@inpe.br

COMISSÃO DO CONSELHO DE EDITORAÇÃO E PRESERVAÇÃO DA PRODUÇÃO INTELECTUAL DO INPE (DE/DIR-544):
Presidente:
Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação (CPG)
Membros:
Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)
Dr. André de Castro Milone - Coordenação de Ciências Espaciais e Atmosféricas (CEA)
Dra. Carina de Barros Melo - Coordenação de Laboratórios Associados (CTE)
Dr. Evandro Marconi Rocco - Coordenação de Engenharia e Tecnologia Espacial (ETE)
Dr. Hermann Johann Heinrich Kux - Coordenação de Observação da Terra (OBT)
Dr. Marley Cavalcante de Lima Moscati - Centro de Previsão de Tempo e Estudos Climáticos (CPT)
Silvia Castro Marcelino - Serviço de Informação e Documentação (SID)
BIBLIOTECA DIGITAL:
Dr. Gerald Jean Francis Banon
Clayton Martins Pereira - Serviço de Informação e Documentação (SID)
REVISÃO E NORMALIZAÇÃO DOCUMENTÁRIA:
Simone Angélica Del Ducca Barbedo - Serviço de Informação e Documentação (SID)
Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)
EDITORAÇÃO ELETRÔNICA:
Marcelo de Castro Pazos - Serviço de Informação e Documentação (SID)
André Luis Dias Fernandes - Serviço de Informação e Documentação (SID)

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

sid.inpe.br/mtc-m21b/2017/05.03.20.18-TDI

UMA ARQUITETURA PARA AVALIAÇÃO DE MÉTRICAS DE QUALIDADE DE SERVIÇO DE SISTEMA DE SEGMENTO SOLO E DE SEUS COMPONENTES

Marcos Paulo Salgueiro de Moraes

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Nilson Sant'Anna, aprovada em 28 de março de 2017.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3NQU9CL>

INPE
São José dos Campos
2017

Dados Internacionais de Catalogação na Publicação (CIP)

Moraes, Marcos Paulo Salgueiro de.

M791a Uma arquitetura para avaliação de métricas de qualidade de serviço de sistema de segmento solo e de seus componentes / Marcos Paulo Salgueiro de Moraes. – São José dos Campos : INPE, 2017.

xxvi + 157 p. ; (sid.inpe.br/mtc-m21b/2017/05.03.20.18-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2017.

Orientador : Dr. Nilson Sant'Anna.

1. Arquitetura. 2. Qualidade de serviço. 3. Atributos e métricas. I.Título.

CDU 658.56:629.7

Esta obra foi licenciada sob uma Licença Creative Commons Atribuição-NãoComercial 3.0 Não Adaptada.

This work is licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License.

Aluno (a): ***Marcos Paulo Salgueiro de Moraes***
"UMA ARQUITETURA PARA AVALIAÇÃO DE MÉTRICAS DE QUALIDADE DE SERVIÇO DE SISTEMA DE SEGMENTO SOLO E DE SEUS COMPONENTES"

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

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

Dr. Maurício Gonçalves Vieira Ferreira

Presidente / INPE / SJCampos - SP

Dr. Nilson Sant'Anna

Orientador(a) / INPE / SJCampos - SP

Dra. Maria de Fátima Mattiello-Francisco

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

Dr. Roberto Affonso da Costa Junior

Convidado(a) / UNIFEI / Itajubá - MG

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 28 de Março de 2017

"Nunca deixe que lhe digam que não vale a pena acreditar no sonho que se tem. Quem acredita sempre alcança!".

Renato Russo
em *"Música Mais Uma Vez"*

A meus pais ***Paulo Roberto*** *e* ***Eunice Salgueiro****,*
minha esposa ***Camile Horta*** *e meu filho querido*
Paulinho

AGRADECIMENTOS

Primeiramente a Deus por me dar força, saúde, sabedoria e tranquilidade para realizar este trabalho.

A minha esposa Camile Horta por todo amor, paciência e compreensão principalmente nos momentos mais difíceis, pois sem ela não teria conseguido finalizar esta jornada.

A meus pais Paulo e Eunice, pelo incentivo, carinho, amor e por tantas vezes que ficaram de “guarda“ com o netinho deles para que eu pudesse dar continuidade ao trabalho.

A meu filho Paulo amor da minha vida, fonte de inspiração e transpiração.

A meu orientador Nilson Sant'Anna pela confiança, boa vontade e dedicação.

À família Lean Survey por todo incentivo e apoio nos momentos difíceis.

Aos amigos Moacyr Gonçalves Cereja Junior, Luiz Gustavo Diniz, Elisabeth Andrade, Raffael Carvalho da Costa e Luiz Tadeu da Silva que me ajudaram nas correções.

Por fim, aos meus amigos e familiares pelo incentivo a concluir esta etapa em minha vida.

RESUMO

Aplicações científicas relacionadas a sistemas de segmento solo tradicionalmente agregam dados capturados por instrumentos distribuídos geograficamente. As sequências de informações percorridas desde a coleta até à disponibilização dos dados estão sujeitas a falhas, sendo que a indisponibilidade desses serviços pode comprometer a qualidade da prestação dos mesmos. Um dos sistemas de segmento solo que podemos observar no Instituto Nacional de Pesquisas Espacias (INPE) é o programa de Clima Espacial, onde atualmente não é possível conhecer a qualidade de serviço (QoS) de suas aplicações. Ainda, percebemos que o INPE em sua maioria não conhece a QoS de seus sistemas. Portanto, essa dissertação apresenta uma arquitetura que permite avaliar a QoS de sistemas de segmento solo e de seus componentes através das seguintes características: (i) definir; (ii) coletar; (iii) processar; (iv) armazenar; e (v) disponibilizar dados de QoS. Além disso, criar uma interface a fim de fornecer dados de QoS para arquitetura de sistemas adaptáveis que necessitam da coleta e monitoramento da QoS para fins de adaptação. O estudo de caso apresenta a implementação de um sistema denominado *ArcticFoxQoS* orientado pela arquitetura proposta, que realiza um fluxo desde a coleta até a disseminação dos dados de QoS de um *web service* que disponibiliza dados espaciais. Por fim, uma análise das limitações encontradas, as contribuições da solução proposta, bem como as considerações para os trabalhos futuros fazem parte desta dissertação.

Palavras-chave: Arquitetura; Qualidade de Serviço; Atributos e Métricas.

A ARCHITECTURE FOR EVALUATION THE QUALITY OF SERVICE GROUND SYSTEMS AND ITS COMPONENTS

ABSTRACT

Scientific applications related to soil segment systems traditionally aggregate data captured by geographically distributed instruments. The sequences of information collected from the collection to the availability of the data are subject to failures, and the unavailability of these services may compromise the quality of the services provided. One of the soil segment systems that we can observe at the National Institute of Space Research (INPE) is the Space Weather program, where it is currently not possible to know the quality of service (QoS) of its applications. Still, we realize that INPE mostly does not know the QoS of their systems. Therefore, this dissertation presents an architecture that allows to evaluate the QoS of soil segment systems and their components through the following characteristics: (i) define; (Ii) collect; (Iii) processing; (Iv) storing; And (v) to provide QoS data. In addition, to create an interface to provide QoS data for adaptive systems architecture that require the collection and monitoring of QoS for adaptation purposes. The case study presents the implementation of a system called textit ArcticFoxQoS oriented by the proposed architecture, which performs a flow from the collection to the dissemination of the QoS data of a web service that provides spatial data. Finally, an analysis of the limitations found, the contributions of the proposed solution, as well as the considerations for the future works are part of this dissertation.

Key words:s Architecture; Quality of Service; Attributes and Metrics.

LISTA DE ABREVIATURAS E SIGLAS

API	–	Application Programming Interface
CBERS	–	China-Brazil Earth Resources Satellite
CONASAT	–	Constelação Nano Satélites para Coleta de Dados Ambientais
COSMIC	–	Constellation Observing System for Meteorology, Ionosphere and Climate
DGI -	–	Divisão de Geração de Imagens
EMBRACE	–	Estudo e Monitoramento BRAsileiro de Clima Espacial
GNSS	–	Global Navigation Satellite System
GPS	–	Global Positioning System
GTex	–	GNSS-TEC Exchange
HTTP	–	HyperText Transfer Protocol
INPE	–	Instituto Nacional de Pesquisa Espaciais
JMS	–	Java Message Service
QoS	–	Quality of Service
SATCS	–	SATellite Control System
SLA	–	Service Level Agreement
SOAP	–	Simple Object Access Protocol
REST	–	Representational State Transfer
UML	–	Unified Modeling Language
JSON	–	JavaScript Object Notation
GQM	–	Goal, Question, Metric
ISO	–	International Organization for Standardization
W3C	–	World Wide Web Consortium
XML	–	eXtensible Markup Language

LISTA DE FIGURAS

Pág.

2.1 Elementos básicos do segmento solo. 13
2.2 Arquitetura completa do SATCS. 17
2.3 Componentes de software do COSMOS. 18
2.4 Visão Geral do Painel de Monitoramento do MOST. 19
2.5 Painel de Monitoramento do CEO. 20
2.6 Aplicação web exibindo os índices de Ksa do magnetômetro. 22
2.7 Modelo do fluxo de dados do Pipeline. 23
2.8 Disponibilização dos dados históricos da rede PCD. 24
2.9 Sistema de Arquivos relacionado com o banco de metadados. 25

3.1 Severidade dos requisitos de QoS para cada tipo de aplicação. 28
3.2 Atributos e métricas de QoS para os provedores de serviço. 30
3.3 SLA XML Schema. 37
3.4 Gerenciamento em tempo de execução de um WSLA. 38
3.5 Monitoramento dos servidores usando o Zabbix. 40

4.1 Visão conceitual de um sistema de robô de controle de empacotamento. . 48
4.2 Fluxo de comunicação do padrão MVC. 51
4.3 Exemplo de uma arquitetura genérica em camadas. 52
4.4 Fluxo de uma arquitetura cliente-servidor. 53
4.5 Comunicação entre cliente e servidor utilizando o protocolo HTTP. . . . 55
4.6 Dois exemplos da configuração do *publish/subscriber*. 58
4.7 Estrutura de um documento XML de mensagem SOAP. 59
4.8 Modelo de Referência IoT. 60

5.1 Modelo Arquitetural do ArcticFox. 66
5.2 Modelo Arquitetural do ArcticFoxQoS. 67
5.3 Número de identificação dos serviços. 68
5.4 Níveis de importância das métricas. 73
5.5 Agentes *Loader* implementando os protocolos HTTP e MQTT. 77
5.6 Documento de descrição de serviços para clientes REST. 78
5.7 Descrição em formato XML de um *Web Service* para clientes SOAP. . . . 79
5.8 Responsável pelo fornecimento da métrica e as formas de implantação. . 83
5.9 Modelo de implantação geral dos componentes do AFQoS. 83
5.10 Tipo de coleta não invasiva das métricas MENE, MONE e EXT. 84
5.11 Tipo de coleta invasiva das métricas MEAC usando coletor embarcado. . 85

5.12 Tipo de coleta invasiva das métricas MEAC usando publicador embarcado. 85
5.13 Tipo de coleta invasiva das métricas MEAT embarcadas pelo AFQoS terceiros. 86
5.14 Tipo de coleta invasiva das métricas MEAEFS embarcada e fora do sítio ao mesmo tempo usando um coletor embarcado. 87
5.15 Tipo de coleta invasiva das métricas MEAEFS embarcada e fora do sítio ao mesmo tempo usando um publicador embarcado. 87

6.1 Redundância do serviço GTex WS do EMBRACE. 90
6.2 Registro do cadastro da metainformação das métricas coletadas. 95
6.3 Pipeline da coleta realizada pelo AFQoSCollector. 96
6.4 Arquitetura do ArcticFoxQoS coletando métricas do tipo MONE. 96
6.5 Comparação do Tempo de Resposta do serviço GTex WS. 97
6.6 Configurações da conta do usuário. 98
6.7 Arquivo JSON usado para modelar as métricas de entrada. 99
6.8 Pipeline de coleta e envio dos dados coletados pelo AFQoS. 100
6.9 Arquitetura do AFQoS coletando métricas do tipo Embarcada e Fora do Sítio ao mesmo tempo. 101
6.10 Tabela de comparação entre os sítios de Cachoeira Paulista X Santa Maria. 101
6.11 Arquitetura do AFQoS coletando métricas do tipo Terceiro. 102
6.12 Tópico criado para gerenciar os dados coletados do servidor wildfly. . . . 103

A.1 Caso de uso do engenheiro da qualidade. 123
A.2 Caso de uso de um gerenciamento de SLA. 124
A.3 Caso de uso de um agente de coleta. 125
A.4 Caso de uso de um serviço de entrega. 126
A.5 Caso de uso de um cliente requisitando e recebendo os dados solicitados. 126
A.6 Estrutura principal dos pacotes do ArcticFoxQoS. 127
A.7 Estrutura do pacote config e sua classe de configuração AppConfig. . . . 127
A.8 Estrutura do pacote data e suas respectivas classes. 128
A.9 Estrutura do pacote filter e suas respectivas classes. 129
A.10 Estrutura do pacote service e suas respectivas classes. 129
A.11 Estrutura do pacote resource e suas respectivas classes. 130
A.12 Estrutura do pacote sessionbean e suas respectivas classes. 131
A.13 Estrutura do pacote sessionbeanlocal e suas respectivas interfaces. 132
A.14 Painel de monitoramento do AFQoS. 133
A.15 Menu lateral para o cadastro dos parâmetros. 134
A.16 Cadastro do atributo de qualidade. 135
A.17 Cadastro das métricas dependentes. 135
A.18 Cadastro da unidade de medida da métrica. 136

A.19 Cadastro dos tipos de escala da métrica. 137
A.20 Definição da parâmetro da frequência de coleta da métrica. 137
A.21 Acordo de nível de servico por métricas. 138
A.22 Código fonte do cálculo do atributo de qualidade *Performance*. 139
A.23 Código fonte do método responsável por entregar a métrica coletada. . . 140
A.24 Código fonte do método responsável por enfileirar a métrica coletada. . . 140
A.25 Código fonte do agente assinante. 141
A.26 Código fonte do serviço de entrega das métricas coletadas por contrato. . 143
A.27 Resposta do serviço *getMetricCollectedByServiceContractAndPeriod* no formato Json. 144
A.28 Código fonte da camada de serviço do Sistema de Gerenciamento do AFQoS. 145

B.1 Diagrama Entidade Relacionamento do AFQoS. 148
B.2 Diagrama Entidade Relacionamento do AFQoS - Parte Iot. 149
B.3 Diagrama Entidade Relacionamento do AFQoS - Parte Metric. 151
B.4 Diagrama Entidade Relacionamento do AFQoS - Parte Architecture Element. 153
B.5 Diagrama Entidade Relacionamento do AFQoS - Parte Security. 155
B.6 Diagrama Entidade Relacionamento do AFQoS - Parte Sla. 157

LISTA DE TABELAS

Pág.

3.1 Atributos de QoS para *web services*. 31

5.1 Definição da métrica 71
5.2 Informações básicas do acordo de nível de serviço 72
5.3 Informações sobre os possíveis status dos SLA´s, métricas e contratos de serviços. 74
5.4 Exemplo de uma metainformação sobre um Resource. 75
5.5 Agrupamento das informações de um elemento arquitetural. 75
5.6 Informações cadastrais de um dispositivo. 76
5.7 Entidade e suas especializações. 76
5.8 Dados de acesso do usuário 80

SUMÁRIO

Pág.

1 INTRODUÇÃO 1
1.1 Objetivos Específicos 4
1.2 Metodologia de Pesquisa 4
1.3 Trabalhos Correlatos 6
1.4 Organização da Dissertação 9

2 SISTEMAS ESPACIAIS 11
2.1 Segmento Solo 12
2.1.1 Definição 12
2.1.2 Comunicação 13
2.1.3 Sistemas de Segmento Solo 14
2.1.3.1 Sistemas de Controle e Rastreio de Satélites 14
2.1.3.2 Aplicações de Clima Espacial 20
2.1.3.3 Sistema de Dados de Satélites Ambientais 23
2.1.3.4 Repositório de Metadados Espaciais 25

3 QUALIDADE DE SERVIÇO 27
3.1 Atributos de Qualidade de Serviço 29
3.2 Acordo de Nível de Serviço 32
3.3 Ferramentas de avaliação da QoS 39

4 ARQUITETURAS DE SISTEMAS 43
4.1 Definições 45
4.2 Visões Arquiteturais 47
4.3 Padrões Arquiteturais 49
4.3.1 Model-View-Controller (MCV) 50
4.3.2 Arquitetura em camadas 51
4.3.3 Duto (*pipe*) e Filtro (*filter*) 52
4.3.4 Repositório 52
4.3.5 Cliente-Servidor 53
4.3.6 REST (A Representational State Transfer) 53
4.4 Protocolos de Comunicação 54
4.5 Sistemas IOT (Internet Of Things) 59

4.6 Arquitetura Adaptável 60

5 ARQUITETURA DO ARCTICFOXQoS 65
5.1 Requisitos 65
5.2 ArcticFox 65
5.3 ArcticFoxQoS 66
5.3.1 Camadas da Arquitetura 67
5.3.1.1 Sistema Alvo 67
5.3.1.2 ArcticFoxQoS 70
5.4 Categorização das métricas 81
5.5 Formas de implantação 82
5.5.1 Responsabilidade de Fornecimento X Formas de implantação 82
5.5.2 Engenheiro da Qualidade 88

6 ESTUDO DE CASO 89
6.1 Cadastro dos parâmetros de QoS 91
6.2 Coleta da métricas do tipo embarcadas pelo ArcticFoxQoS Agente Coletor 94
6.3 Coleta das métricas do tipo embarcada e fora do sítio ao mesmo tempo . 97
6.4 Coleta das métricas do tipo embarcadas pelo ArcticFoxQoS - Terceiros . 102
6.4.1 ArcticFoxQoS Implementação 103
6.4.1.1 Estrutura do Projeto 103

7 CONCLUSÃO 107

REFERÊNCIAS BIBLIOGRÁFICAS 111

APÊNDICE A: Documentação da implementação do sistema ArcticFoxQoS 119
A.1 Requisitos 119
A.1.1 Requisitos Arquiteturais - RA 119
A.1.2 Requisitos Funcionais do ArcticFoxQoS - RFA 120
A.2 Casos de uso do ArcticFoxQoS 123
A.2.1 Gerenciamento da Qualidade 123
A.2.2 Gerenciamento de SLA 124
A.2.3 Agentes de Coleta 125
A.2.4 Deliver 126
A.2.5 Cliente 126
A.3 Diagramas de Pacotes do ArcticFoxQoS 127
A.3.1 Pacote Principal 127

A.3.2 Pacote Config 127
A.3.3 Pacote Data 128
A.3.4 Pacote Filter 129
A.3.5 Pacote Services 129
A.3.6 Pacote Resource 130
A.3.7 Pacote SessionBean 131
A.3.8 Pacote SessionBeanLocal 132
A.4 Telas do sistema de cadastro 132
A.5 Código fonte do CollectorQoSAgentWS 138
A.6 Código fonte do ReceptorService 140
A.7 Código fonte do SubscriberQoSAgentWS 141
A.8 API do Sistema de Entregas 141
A.9 Código fonte do Service Client do *Front-End* 145

APÊNDICE B: Diagramas de Entidade e Relacionamento relacionados à base de dados do ArcticFoxQoS 147
B.1 Diagrama Entidade Relacionamento 147
B.1.1 DER - IoT 149
B.1.2 DER - Metric 151
B.1.3 DER - Architecture Element 153
B.1.4 DER - Security 155
B.1.5 DER - SLA 157

1 INTRODUÇÃO

Os satélites artificiais são de extrema importância para o desenvolvimento do ramo da ciência tecnológica espacial, podendo ser utilizados em diversas categorias de missões espaciais, como forças armadas, sensoriamento remoto, meteorologia, oceanografia, espionagem, alarme e reconhecimento, comunicação e muitos outros. No contexto espacial a prestação de serviço está contida em diversos segmentos, sendo o solo um deles. Sua aplicabilidade visa desenvolver produtos no estado da arte, como sistemas de telecomunicações, rastreio e controle de satélites, recepção, processamento e disponibilização de dados espaciais.

Barbosa (2009) ressalta que "a implantação de uma infraestrutura de solo, seja para a coleta e manuseio de dados científicos ou para o desenvolvimento de produtos e processos derivados, passou a ser prioridade do programa espacial brasileiro a partir da existência de recursos humanos capacitados."

Normalmente, as missões espaciais possuem vários objetivos. Alguns são secundários e podem ser atendidos por um conjunto definido de equipamentos, enquanto outros são objetivos adicionais que podem exigir mais equipamentos (LARSON; WERTZ, 1999). Os sistemas de software espaciais possuem inúmeros requisitos com o propósito de garantir a extração máxima de informações durante a vida útil de um satélite. O fluxo de informações que ocorre desde a coleta até à disponibilização dos dados percorre por diversos elementos do segmento de solo. Exemplos destes são os centros de controle e rastreio de satélites (determinação de órbita, plano de voo), Estação solo (recebimento de telemetrias e envio de telecomandos) e centro de missão (armazenamento de dados e disseminação de dados da carga útil).

No sistema de coleta de dados do Instituto Nacional de Pesquisas Espaciais (INPE), segundo (ORLANDO; KUGA, 2007), os satélites funcionam como retransmissores de mensagens das PCD´s (Plataformas de Coleta de dados ambientais). Quando um dos satélites do sistema entra na região de alcance (visibilidade) de uma das estações de recepção (Cuiabá ou Alcântara), a comunicação satélite solo é estabelecida.

No entanto, esses elementos estão expostos a situações severas durante a transferência de informações até os centros de processamento e armazenamento de dados. Além disso, os componentes de software que atuam em cada fase deste processo são passíveis de falhas. O não funcionamento desses sistemas pode resultar em situações críticas, uma vez que podem inviabilizar a execução de análises e a predição de eventos de interesse econômico, tecnológico, entre outros.

Considerando que esses componentes são explorados através de serviços, comumente os sistemas de segmento de solo e seus componentes não possuem indicadores que possibilitam identificar a qualidade da prestação dos seus serviços, seja para o auxílio em uma missão espacial ou para disponibilização de dados da carga útil. Consequentemente é muito difícil propor melhorias nesses sistemas, o que dificulta a redução de custos em projetos futuros. Inclusive abordagens como monitoramento de métricas de qualidade de serviço e arquiteturas autoadaptáveis aplicadas nos sistemas segmento solo podem ser implementados nas missões correntes, resolvendo problemas existentes que usualmente necessitam de intervenção humana para resolvê-los.

No INPE existem diversas aplicações de segmento solo que prestam serviços diretamente à comunidade acadêmica, empresas estatais, governamentais e usuários comuns. A Divisão de Geração de Imagens (DGI) do INPE, é responsável pela recepção, processamento e distribuição de imagens de sensoriamento remoto adquiridas pelas estações terrenas de Cuiabá e Alcântara. Os satélites de sensoriamento remoto CBERS (*China-Brazil Earth Resources Satellite, Satélite Sino-Brasileiro de Recursos Terrestres*) angariaram significativos avanços ao Brasil. As imagens de satélite disponibilizadas pela DGI são utilizadas em diferentes aplicações, como cartografia, degradação ambiental, agricultura, geografia e outros. Uma pesquisa realizada em 2009 sobre o perfil dos usuários das imagens de satélite CBERS avaliou o tempo entre a solicitação e o recebimento das imagens recebidas eletronicamente (SILVA, 2009). Todavia essa pesquisa foi realizada apenas em 2007 e 2009 respectivamente e tratava-se apenas do catálogo CBERS. Outros sistemas de segmento solo dentro do INPE não possuem avaliações em relação à qualidade dos serviços prestados, sua utilização e principalmente sobre seus requisitos não funcionais como a disponibilidade e o desempenho dos sistemas. Outros projetos do mesmo segmento podem ser vistos em COSMIC (SOUZA et al., 2015) e CONASAT (CARVALHO et al., 2013).

O Centro de Monitoramento de Clima Espacial localizado em São José dos Campos é responsável por disseminar dados de diversos tipos de equipamentos. Um deles é o magnetômetro responsável pela medição do campo magnético da terra (COSTA L. L.; DURÃO, 2011). A coleta dos dados é realizada nas cidades de Alta Floresta (MT), Cachoeira Paulista (SP), Eusébio (CE), Jataí (GO), Rio Grande (Argentina), São José dos Campos (SP), São Luís (MA), São Martinho da Serra (RS) e Vassouras (RJ) e disponibilizados através de um portal web hospedado no centro de monitoramento em São José dos Campos. Entretanto a demora na entrega de um dado, uma intermitência ou até mesmo a indisponibilidade desse portal, certamente comprometeriam a qualidade do serviço (QoS - *Quality Of Service*) prestado aos seus

usuários.

Outro exemplo em relação à medição da qualidade de serviço pode ser encontrado no Centro de Controle e Rastreio em São José dos Campos. Para enviar um telecomando o operador deve se conectar através do sistema de controle na estação terrena de Cuiabá ou Alcântara. Entretanto pode acontecer uma falha de comunicação e o link ficar temporariamente indisponível correndo o risco de perder os dados durante a passagem do satélite sobre a estação. Quando ocorre uma situação dessas, o operador verifica o log do sistema na tentativa de identificar o problema ou entra em contato com as estações terrenas. Outra medida é acionar um backup de segurança instalado na estação terrena e o controle e rastreio passa a ser realizado por lá até o restabelecimento do link. Portanto, questionamentos pertinentes como a duração em que o link esteve fora de comunicação; o mês em que houve maior queda de disponibilidade durante a passagem do satélite; o maior tempo de resposta e a menor acurácia, tornam-se desconhecidos e numa perspectiva sistêmica tendem a inibir a eficiência do processo.

Para (O'BRIEN et al., 2007) o vital sucesso de uma arquitetura de software, torna-se necessário criar uma ponte entre as camadas de forma que os atributos de qualidade (confiabilidade, segurança, desempenho, etc) sejam satisfatórios. Por conseguinte a importância de indicadores de métricas de qualidade de serviço de um sistema de segmento solo e de seus componentes, tornam-se fundamentais para a garantia das características não funcionais.

O padrão da arquitetura de referência proposta neste trabalho apresenta o modelo em camadas, cujo propósito é viabilizar a construção dos componentes de forma desacoplada e independentemente do restante. A divisão das camadas é composta pelos seguintes componentes: Dispositivo/Serviços; Agente Embarcado; Repositório embarcado; Comunicação; Agente de Coleta Não Embarcado; Repositório; Metainformação; Processador de Eventos; Sistema de Entregas API; Segurança; e Aplicação.

Com a adoção da arquitetura proposta, a possibilidade de avaliar a QoS por meio dos indicadores gerados, permitirá o conhecimento situacional de qualquer sistema ou componente prestador de serviço. Portanto, os sistemas prestadores de serviço atualmente implantados, poderão ser conhecidos no âmbito de suas características não funcionais e ainda, promover serviços redundantes como soluções alternativas para a garantia da disponibilidade devido à predição de análise desses sistemas.

Outro aspecto relevante da arquitetura de referência proposta, é que parte de seu modelo foi construído com base no conceito de Internet das Coisas (IoT - *Internet Of Things*) (BASSI et al., 2013), assim, é possível coletar QoS de qualquer "coisa". Entretanto como objeto de estudo deste trabalho, utilizamos os sistemas de segmento solo como referência.

1.1 Objetivos Específicos

Diante de um breve panorama acerca das especificidades dos sistemas de segmentos de solo existentes, verifica-se a importância em aprofundar a temática de estudo por meio de uma arquitetura de referência capaz de definir, coletar, processar, armazenar e disponibilizar métricas de qualidade de serviços de sistemas de segmento solo e de seus componentes. O objetivo geral pode ser dividido nos seguintes objetivos específicos:

a) Definir uma arquitetura capaz de avaliar a QoS de qualquer sistema de segmento solo e de seus componentes;

b) Projetar e implementar uma prova de conceito de um sistema denominado *ArcticFoxQoS (AFQoS)* dirigido pela arquitetura proposta;

c) Realizar um estudo de caso para validação da arquitetura base proposta por intermédio do AFQoS; e

d) Apresentar os resultados obtidos através do sistema AFQoS.

1.2 Metodologia de Pesquisa

Para atingir o objetivo deste trabalho, a arquitetura proposta está baseada em um modelo de referência apresentado em (BASSI et al., 2013). Ela fornece conceitos e definições na qual a arquitetura da internet das coisas pode ser construída. Com base nesse modelo qualquer "coisa" (o foco da aplicação neste trabalho será sistemas de segmento solo e seus componentes) plugada na arquitetura poderá ter a qualidade do serviço monitorada.

Os atributos e métricas de qualidade de serviço que foram utilizadas como parâmetro de medição do estudo de caso proposto neste trabalho, estão baseadas no modelo apresentado em (CHOI et al., 2007). O propósito deste modelo é quantificar a qualidade do serviço prestado pelos provedores de forma a satisfazer as expectativas dos seus consumidores.

Para a garantia desse serviço também será utilizado um acordo de nível de serviço baseado em um documento de especificação apresentado em (BAKALOS et al., 2015). O documento integra métricas como objetivos ou atributos de qualidade específicos, parâmetros e regras. O diagrama de entidade e relacionamento foi baseado no modelo citado, sendo que a partir disso uma base de dados foi criada para armazenar todos os metadados referentes aos sistemas monitorados e suas respectivas métricas coletadas. A disponibilização dos dados coletados será uma saída estruturada nos formatos JSON *(JavaScript Object Notation)* (BRAY, 2014), portanto será possível prover um formato legível por máquina aos clientes possibilitando interoperabilidade entre as aplicações consumidoras do serviço.

A implementação do *back-end* (agentes de coleta não embarcados (*Coletor e Publicador*), o agente processador de eventos (*Loader*), o serviço web contendo as API´s *(Application Programming Interface*) de entrega dos dados (*Deliver*), o sistema de metainformação onde são armazenadas todas as informações do acordo de nível de serviço (*Discovery*) do ArcticFoxQoS) foram implementados utilizando a linguagem de programação orientada a objetos JAVA. A camada responsável por obter a entrada de dados do usuário *front-end* utilizou-se a linguagem de programação AngularJS. Os agentes de coleta embarcados utilizou as ferramentas gratuitas de monitoramento Zabbix e Google Analytics.

A camada de segurança responsável pela autorização de controle de acesso (permissões) e autenticação dos serviços e clientes consumidores utilizou a API *Java Authentication and Authorization Service* (JAAS). Dessa forma foi possível criar regras de segurança e perfis dos usuários. Outras ferramentas foram incorporadas com a finalidade de auxiliar o desenvolvimento, como o framework de mapeamento-objeto-relacional (ORM) eclipselink inspirada na especificação *Java Persistence API* (JPA), o componente da plataforma JEE (*Java Platform Enterprise Edition*) chamada *Enterprise JavaBeans* (EJB) onde deve ser implantada em um servidor de aplicação, cujo objetivo visa facilitar a integração de componentes distribuídos e transacionais.

A fim de que o sistema de coleta das métricas realize o monitoramento de inúmeros sistemas alvo em paralelo, economizando recursos de rede, descongestionando o número de conexões ao banco de dados e viabilizando o armazenamento das métricas coletadas, priorizou-se o uso da API *Java Message Service* que permite aplicativos criar, enviar, receber e ler mensagens. Outra forma de implementação de mensagens baseada no protocolo MQTT (*Message Queue Telemetry Transport*) para fins

deste experimento, foi a utilização do *broker* ActiveMQ, através do paradigma *publish/subscriber* permitindo também a comunicação *machine-to-machine (M2M)*.

Por último, para o armazenamento de todas as informações dos sistemas alvo e as métricas coletadas foi utilizado o banco de dados PostgreSQL. Vale ressaltar, que no estudo de caso foi realizada uma simulação de um serviço web GTex WS disponibilizado pelo Estudo e Monitoramento Brasileiro de Clima Espacial (EMBRACE). Para que isso ocorresse foi implementado dois sistemas redundantes do GTex WS. Os mesmos foram implementados através da linguagem de programação JAVA, banco de dados postgres e mysql, servidores de aplicação Glassfish e Jboss WildFly respectivamente.

1.3 Trabalhos Correlatos

Nesta seção detém-se a uma revisão dos principais trabalhos e seus respectivos autores que contribuíram direta e/ou indiretamente para a construção de conceitos e teorias relacionados aos temas: sistemas no âmbito espacial, arquitetura e qualidade de serviços. Para tanto, faz-se necessário um aprofundamento acerca do funcionamento de tais temáticas, a fim de que seja exitoso o processo de implementação da arquitetura proposta.

No que concerne aos sistemas espaciais, em (SANT'ANNA et al., 2014) verifica-se a existência de um modelo arquitetural denominado *pipeline* responsável pelo fluxo dos dados desde a recepção à visualização e distribuição dos dados de clima espacial. O estudo possibilitou conhecer as camadas arquiteturais responsáveis por todo o processo e seus principais requisitos. Apesar do trabalho mencionar os requisitos não funcionais da arquitetura e seus subsistemas, nada foi explorado em relação à qualidade de serviço dos seus componentes arquiteturais. A ausência de tal vinculação dificulta uma análise comparativa dos serviços prestados pelo EMBRACE ao longo do tempo, optando-se a partir deste levantamento pelo estudo de caso para fomentar o processo de qualidade dos serviços prestados.

No trabalho de (SANTOS et al., 2013) é apresentado um sistema de coleta de dados ambientais localizado no centro de missão em Natal, RN. A execução do modelo arquitetural do projeto SINDA (Sistema Integrado de Dados Ambientais) é constituído de duas partes: a primeira é responsável pelo processamento dos dados recebidos pelas estações, decodificação e armazenamento na base de dados. A segunda parte caracteriza-se pelas funcionalidades de busca, estatísticas e disponibilização dos dados aos usuários. Em relação à analise da qualidade de serviço do sistema,

o SINDA utiliza uma ferramenta analítica *open source* NAGIOS para o monitoramento da infraestrutura como hosts e serviços, emitindo alertas em caso de alguma ocorrência e/ou restabelecimento dos serviços. Para o monitoramento do tráfego da rede a ferramenta analítica gráfica utilizada foi o Cacti, e para a análise global do sistema foram realizadas análises de cobertura baseadas em dados reais através de simulações com o software STK *System Tool Kit*. Outros dados estatísticos foram levantados através de e-mails de *feedbacks* em respostas aos atendimentos dos usuários em relação à qualidade do serviço bem como o crescente número de acessos ao sistema.

Quanto aos sistemas de controle e rastreio de satélites, destaca-se o trabalho de (CARDOSO et al., 2006), o qual contribuiu para um relativo entendimento sobre seu funcionamento, camadas de arquitetura, reutilização de componentes, interoperabilidade e padrões de projetos utilizados visando melhorias na qualidade do serviço. Como não se tem uma efetiva medição, a eficiência torna-se prejudicada, além a inexistência de autonomia do sistema para suprir eventuais falhas e/ou perdas de comunicação com as estações terrenas, impedindo que volte a operar normalmente.

Ainda sob a perspectiva de arquiteturas o termo "Internet das Coisas" conhecido em inglês por *Internet of Things* (IoT) trata uma nova maneira de conexão entre "coisas" através da internet. Em (DINIZ, 2006) o autor expõe que além do novo paradigma permitir a conexão em qualquer tempo e lugar, agora também é possível a comunicação entre coisas, e uma nova sigla é incorporada entre as mais utilizadas na internet sobre transações online como: B2B (*business to bussines*), B2C (*Business to Consumer*), G2C (*Governement To Costumer*), C2C (*Consumer to Consumer*), além de outras como, H2T (*Human-to-Thing*) e T2T (*Thing To Thing*).

Pensando na comunicação T2T (BASSI et al., 2013) apresenta o modelo de referência arquitetural IoT-A *Internet of Things - Architecture* desenvolvido com o objetivo de definir um conjunto de blocos de construção para ambientes de IoT, e fornecer uma base sólida para alavancar sua ampla adoção. O modelo possui um alto nível de abstração, melhores práticas e definição de relacionamentos entre conceitos de instancias como "*Serviços expõem Recursos*". Ressalta-se ainda a importância de definir os principais conceitos abstratos, suas responsabilidades e seus relacionamentos.

No domínio da IoT, o conceito de dispositivo provavelmente permanecerá relevante no futuro, mesmo que os tipos de dispositivos usados apresentem mudanças ao longo do tempo e/ou variações, a depender do contexto da aplicação. Um aspecto relevante a ser abordado sobre a IoT-A é que além de permitir realizar a comunicação

entre dispositivos através da internet, o modelo prevê garantir a qualidade em características não funcionais do tipo: interoperabilidade, disponibilidade e resiliência, confiança, segurança e privacidade, além de desempenho e escalabilidade. Todavia, a temática da qualidade de serviço em uma perspectiva sistêmica para o desenvolvimento da arquitetura proposta nesta dissertação necessita de adaptações, na medida em que não se fala sobre como armazenar, processar e distribuir os dados de QoS (*Quality of Service*) para uma análise dos recursos e serviços disponíveis.

No trabalho de (CHOI et al., 2007), o autor identifica características únicas da arquitetura SOA *Service-Oriented Architecture* para derivar as métricas de QoS. Baseado nisso foram identificados seis atributos de qualidade de serviço e para cada um deles, um conjunto de métricas. Em virtude da relevância do modelo e de sua compatibilidade com os interesses do presente estudo (ao ter como base uma arquitetura SOA e a aplicação alvo monitorada ser um *web service*), optou-se por utilizá-lo como referência, observadas as devidas adaptações. É importante ressaltar que as informações acerca das métricas e dos requisitos necessários para realizar a coleta não foram fornecidas em sua totalidade, evidenciando apenas como foram calculadas através de fórmulas e de um desenho da arquitetura. O estudo de caso foi um serviço de reservas de hotel onde possui 3 tipos de camadas quais sejam, *Service Consumer* (usuário cliente que realiza a reserva), *Bussines Process* (camada responsável pelo recebimento da informação, análise, processamento de reserva, entre outros) e por último a interface de acesso aos serviços. Neste exemplo os serviços apresentados são apenas disponibilizados em SOAP (*Simple Object Access Protocol*). Assim, embasado nessas características os atributos e métricas de qualidade de serviço, apresentados no trabalho acima, são requisitos essenciais a serem observados por prestadores de serviços e avaliados e executados por consumidores.

Métricas da qualidade de serviço também podem ser utilizadas em arquitetura de sistemas adaptáveis. No trabalho de (MIRANDOLA; POTENA, 2011) é apresentado um framework fundamentado em um modelo de otimização *Service-Based System - SBS* cujo o propósito é possibilitar a redução de custos de adaptação e preservação dos níveis exigidos de qualidade do sistema. Uma visão geral dos componentes da estrutura é apresentada, em especial o módulo do Monitor o qual é responsável pelas medições de qualidade de serviço.

Para a garantia da qualidade de serviço em *cloud service* (BAKALOS et al., 2015) apresenta um modelo de referencia denominado SLALOM baseado na norma de especificação ISO para a criação de um acordo de nível de serviço, conhecida em inglês

pela sigla SLA *Service Level Agreement*. É um documento que serve para garantir que requisitos funcionais e não funcionais de um serviço estejam dentro das perspectivas dos consumidores e prestadores de serviço. Nesse documento, provedores de serviços como a Amazon Web Service, Microsoft Azure e Google Cloud fornecem uma análise de seus SLA´s públicos comerciais com o propósito de expor as inovações em termos de SLA disponíveis no mercado. Além disso também sugeriram adições e/ou melhorias. Nesse caso, as sugestões refletem as preocupações dos consumidores em relação à subcontratação da nuvem e controles de segurança mínimos exigidos.

No trabalho de (SHADE et al., 2012) verifica-se que os serviços web vem sendo utilizados em diversos setores como: e-business, e-government, sistemas automotivos, serviços de multimídia, controle de processos, finanças e inúmeros outros domínios. Dessa forma a qualidade de serviço QoS vem sendo frequentemente aplicada para descrever os requisitos não funcionais desses serviços web e empregada como um significativo ponto de separação de serviços distintos, tornando a QoS cada vez mais importante. Diante do exposto, tem-se uma visão geral dos requisitos de qualidade de serviço para *web services* e, baseado nestes, respaldo para selecionar as métricas de maior importância no processo de coleta.

Em (CARVALHO et al., 2009) os autores propuseram um modelo de qualidade baseado na ISO 25000 porém não abordava tudo que precisava para o desenvolvimento de aplicações de software embarcados. Em termos gerais, o modelo apresenta características dos atributos e métricas de qualidade relacionadas à avaliação dos componentes, sendo possível adentrar em características não funcionais como eficiência, manutenibilidade, portabilidade, dentre outras. O artigo também contempla a aplicação da abordagem *Goal Question Metric* para planejar o processo de medição que tem como objetivo usar um determinado conjunto de questões de regras para a interpretação dos dados de medição. Entretanto o trabalho ainda está em andamento e o grau de maturidade desse modelo ainda não foi alcançado, mas algumas boas práticas podem ser aproveitadas.

1.4 Organização da Dissertação

A dissertação está estruturada da seguinte forma:

- O capítulo 2 aborda conceitos de sistemas espaciais no âmbito de segmento solo. Exemplos de aplicações como: controle e rastreio, clima espacial, sistema de coleta e disseminação de dados ambientais e a arquitetura de um

repositório de imagens de satélite;

- O capítulo 3 versa sobre a qualidade de serviço. Além disso, apresentar conceitos de acordo com o nível de serviço para a garantia da qualidade, ferramentas de coleta e análise de métricas disponíveis no mercado;

- No capítulo 4 são explicitadas conceitos relacionados à arquiteturas de software, padrões e visões arquiteturais e protocolos de comunicação;

- O capítulo 5 expõe de forma detalhada a arquitetura de referência proposta nessa dissertação, bem como suas camadas, relacionamentos, funcionalidades e formas de implantação. Ainda, é exemplificada a definição e categorização das métricas;

- O capítulo 6 descreve um estudo de caso que irá simular um serviço web de GNSS GTex atualmente disponibilizado pelo EMBRACE em São José dos Campos. A proposta é realizar uma simulação criando uma redundância do serviço GTex e coletar a QoS dos mesmos utilizando a arquitetura de referência do ArcticFoxQoS;

- O Capítulo 7 apresenta as considerações finais, as contribuições e recomendações de trabalhos futuros que poderão ser incorporados a partir da arquitetura proposta e dos resultados obtidos; e

- O apêndice A apresenta os requisitos arquiteturais e funcionais, casos de uso, diagrama de classe, telas do sistema de gerenciamento da qualidade, e alguns códigos fonte implementados para o estudo de caso. O apêndice B apresenta o diagrama de entidade e relacionamento resultante da implementação do sistema.

2 SISTEMAS ESPACIAIS

Com os crescentes avanços da tecnologia da informação na área espacial, aplicações científicas se tornam cada vez mais sofisticadas. A robustez, eficácia e eficiência no processamento, armazenamento e distribuição das informações tornam-se requisitos essenciais na operação dos produtos gerados a partir de dados coletados por satélites artificiais, aeronaves, balões estratosféricos, robôs, entre outros.

Em termos conceituais, entende-se por programa espacial o desenvolvimento de uma série de funções (de forma gradativa) atrelado à diferentes projetos que visem alcançar o objetivo proposto. Uma missão espacial corresponde a um ou mais objetivos estabelecidos pelos *stakeholders* (termo utilizado às partes envolvidas no processo) promovendo o desenvolvimento de um sistema onde parte dos inclui a utilização de satélites ou outros artefatos no espaço. No Brasil a primeira missão espacial desenvolvida foi a Missão Espacial Completa Brasileira (MECB), cujo o objetivo era alavancar a pesquisa científica, a qualificação de pessoal e a formação de tecnologia espacial no Brasil, com envolvimento da indústria nacional (ORLANDO; KUGA, 2007). Entre os objetivos estabelecidos pela MECB estavam o desenvolvimento de:

- Dois satélites de coleta de dados;
- Dois satélites de sensoriamento remoto;
- A infraestrutura para qualificação de satélites;
- O segmento solo para controle dos satélites;
- As bases de lançamento de Alcantara;
- Quatro foguetes VLS para qualificá-lo.

De forma específica, qualquer elemento que esteja em órbita da Terra com um determinado porte é considerado um satélite. Para fins deste estudo, os satélites desenvolvidos são caracterizados como artificiais. Para que se inicie o ciclo de vida de um satélite que será transitado em órbita, alguns procedimentos são requisitos primordiais e devem seguir um fluxo de sequência de eventos relacionados a fim de alcançar o máximo nível de segurança, confiabilidade e desempenho. Larson e Wertz (1999) ressalta que o ciclo de vida de uma missão espacial dá-se através de quatro fases: exploração de conceitos, estudo inicial no qual resulta em uma definição ampla da missão espacial e seus componentes; desenvolvimento detalhado dos componentes

do sistema, testes de *hardware* e *software*; produção e implantação, construção de equipamentos e softwares terrestres; operações, suporte e manutenção.

A integração de um satélite é composta de dois módulos: carga útil (*"payload"*) e serviço (*"bus"* ou "plataforma"). A carga útil é formado pelos equipamentos (antenas, sensores e transmissores) necessários para o cumprimento da missão. A plataforma contém todos os equipamentos para o funcionamento do satélite, apoiando a operação da carga útil no espaço e garantindo as funções de serviço como apontamento, controle de temperatura, execução das funções de bordo e comunicação com o pessoal de Terra para manter o satélite funcionando.

A missão espacial de um satélite é composta por 4 (quatro) segmentos arquiteturais: Espacial; Lançador; Solo e Usuário. Segundo (MATTIELLO-FRANCISCO, 2003) os segmentos usuário e de lançamento apesar de possuírem requisitos cruciais na especificação da missão e atributos decisivos para o projeto de satélite, não fazem parte do desenvolvimento de uma missão de satélite. Para fins deste estudo, o segmento solo será melhor explorado nos itens seguintes.

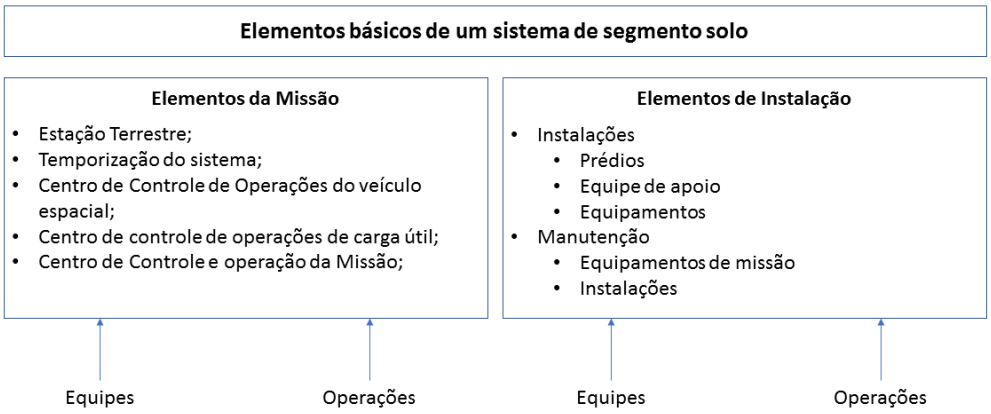
2.1 Segmento Solo

2.1.1 Definição

"Compreende a todos os sistemas terrestres que são usados para apoiar as atividades de preparação que conduzem às operações da missão, a condução das próprias operações e todas as atividades pós-operacionais" (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS), 2008).

"Para apoiar uma missão espacial real, um sistema terrestre deve cobrir simultaneamente diversos veículos espaciais em diferentes órbitas, com níveis de disponibilidade e segurança desejáveis52. Tais sistemas geralmente incluem muitos elementos em várias configurações. (LARSON; WERTZ, 1999). A Figura 2.1 mostra que o sistema solo é dividido em elementos de missão e elementos de instalação.

Figura 2.1 - Elementos básicos do segmento solo.



Fonte: Adaptada de Larson e Wertz (1999).

2.1.2 Comunicação

De acordo com (COMPRESSION, 2013) a comunicação entre os veículos espaciais e um sistema de solo é dada pelo protocolo de enlace de comunicação *Space Link*. Tais protocolos permitem realizar a transferência de diversos tipos de dados espaciais.

A comunicação solo-bordo de serviço é realizada através de subsistemas de telecomunicação conhecidos como: Telemetria, Rastreio e Comando (TT&C), os quais serão abordados nos tópicos a seguir:

- **Telemetria**: refere-se às medições de status geradas pelo veículo espacial que caracterizam o estado e a configuração do satélite;
- **Rastreamento**: determina a posição do satélite através das informações de direção (é obtida por meio da medida dos ângulos de elevação e azimute das antenas de rastreio), distância (obtida pela medida do tempo (*"delay"*) que leva para um sinal emitido ir até o satélite e voltar) e taxa de variação da distância (aproximação ou o afastamento do satélite em relação à antena no solo);
- **Comando**: refere-se aos sinais que são transmitidos ao veículo espacial

para operar e reconfigurar subsistemas e cargas úteis.

2.1.3 Sistemas de Segmento Solo

Existem diversos tipos de aplicações no contexto segmento solo. Dentre elas, podem-se destacar os sistemas de clima espacial, meteorológicos e ambientais, controle e rastreio de satélites, sensoriamento remoto, oceanografia, entre outros. A seguir, tem-se o detalhamento de algumas dessas aplicações tendo em vista os objetivos do presente estudo.

2.1.3.1 Sistemas de Controle e Rastreio de Satélites

É composto de todos os equipamentos, serviços e programas que permitem o controle do satélite durante o ciclo de vida no espaço e o planejamento e execução da missão, inclusive os meios de verificação necessários.

O desenvolvimento dos sistemas de controle de satélite no INPE encontra-se sob a responsabilidade da Divisão de Desenvolvimento de Sistemas de Solo - DSS. O objetivo é desenvolver sistemas cuja arquitetura permita a reutilização máxima para o funcionamento de outros satélites e não se destina apenas uma missão específica. Os sistemas de controles de satélites são capazes de realizar planejamento da missão e programação de atividades, gerenciamento e manutenção do software de bordo, dinâmica orbital, armazenamento, processamento e disseminação de dados, geração de produtos e gerenciamento da base de dados de operação.

Primeiro sistema desenvolvido pela DSS nos anos 80 foi o SICS (Sistema de Controle de Satélite) um sistema monolítico de alto custo de manutenção e ineficiência. Sua finalidade era controlar satélites da MECB. (FERREIRA, 2001).

Atualmente o sistema que se encontra em operação para o controle e rastreio de satélites é o *SATellite Control System* (SATCS). Sua arquitetura foi desenvolvida visando maximizar a reutilização criando aplicações que podem ser facilmente configuradas nas funcionalidades do segmento terrestre e automatização de operações do segmento solo. (CARDOSO et al., 2006). Dessa forma para fomentar o aumento da usabilidade e a redução dos custos relacionados ao desenvolvimento e manutenção do sistema, a arquitetura do SATCS possui um modelo generalizado e suas principais características são listadas a seguir:

- Maximizar o reúso;

- Arquitetura dividida em camadas, permitindo um baixo acoplamento entre os principais componentes;
- Criar aplicações que podem ser facilmente customizadas para serem utilizados nas operações do segmento solo;
- Maior generalização do banco de dados;
- Telemetria e Telecomando mais facilmente personalizado para diferentes missões;
- Utilização de *design patterns* no desenvolvimento permitindo maior flexibilidade;
- Apoio a fase de Montagem, Integração e Testes (AIT);
- Permitir a configuração para diversos tipos de satélites (Femtossatélite, Picossatélite, Nanossatélite, Microssatélite, Minissatélite e Satélites de Grande Porte e outros); e
- Automatizar as operações do segmento solo.

A arquitetura do SATCS é dividida em 3 (três) camadas, as quais serão definidas a seguir:

- **Kernel** - Essa camada é composta por um subsistema de suporte e duas bases de dados.
  - **Operations Database Management** - descreve as telemetrias, telecomandos, eventos, e parte da configuração do sistema que são definidos para uma missão específica.
  - **Archiving and Retrieving Data** - armazena os dados recebidos do satélite (telemetria), e os telecomandos, como medidas de distância e velocidade.
  - **Supporting** - é um subsistema de apoio, responsável pela implementação de vários serviços. Todos os serviços são acessados através do kernel pelas interfaces dos subsistemas a seguir:
    * **Communication**: fornece suporte de comunicação síncrona e assíncrona através do protocolo TCP/IP. O subsistema possui

classes e métodos responsáveis pelo interfaceamento da comunicação com a estação. Os parâmetros necessários para a realização da comunicação como, o IP *Internet Protocol* de algum componente da estação, podem ser localizados na base de dados da operação (*Operations Database Management*).

  - **Configuration Param**: recupera parâmetros de configuração dos usuários armazenados no banco de dados de operação.
  - **Event Message**: armazena todos os logs de mensagens geradas pelo SATCS durante sua execução;
  - **Event Visual**: disponibiliza um serviço de logs de mensagens que são geradas pelo SATCS;
  - **File Manager**: transfere e apaga arquivos pela rede através dos protocolos FTP, SSH ou NETBIOS;
  - **Special Types**: são classes responsáveis pela operação de tipos de dados especiais como: *array of bits, datetime*;
  - **Timer**: *Scheduler Service*, é um sistema agendador para executar serviços de forma síncrona;
  - **User Interface**: implementa serviço de interface de usuário padrão (input masks);
  - **Meta-Instance**: são metadados armazenados no banco de dados de operações (codificação e decodificação), possui dados do tipo: decodificação de dados pra engenharia, emissão alarmes, informação sobre os parâmetros de operação, como: bit = zero é falso;

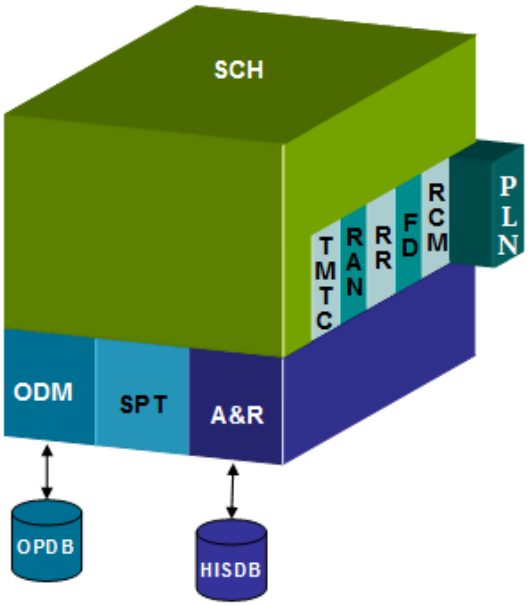
Juntos, esses subsistemas possibilitam isolar detalhes da implementação da parte de operação do sistema.

- **Application** - essa camada é responsável pelas aplicações fundamentais para o monitoramento e controle dos satélites permitindo a execução de comandos como telemetria e telecomando, medidas de distância e velocidade, dinâmica orbital, monitoramento e controle das estações de solo e geração relatórios do comportamento do satélite e seus subsistemas;

- **Automation** - camada responsável por automatizar tarefas que são necessárias a atuação de um operador e viabilizar o controle de satélites adicionais sem a necessidade do aumento da equipe operacional.

Diante das especificidades do SATCS relacionadas às características arquiteturais e divisão em camadas, tem-se na Figura 2.2 uma ilustração da arquitetura completa.

Figura 2.2 - Arquitetura completa do SATCS.



Fonte: Cardoso et al. (2006).

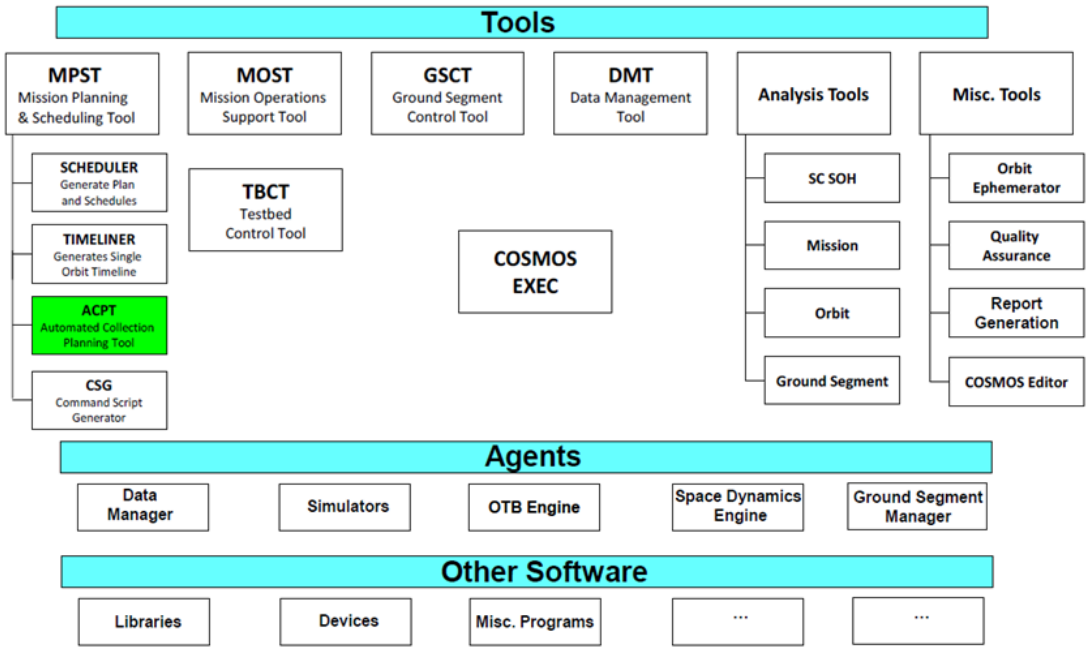
Outro sistema de controle e rastreio de satélites pode ser observado em (SORENSEN, 2012). Um framework *open source* desenvolvido no *Hawaii Space Flight Laboratory* cujo o objetivo é fornecer suporte a missões espaciais de um ou mais veículos de pequeno porte durante todo os seu ciclo de vida. Igualmente ao SATCS o COSMOS (*Compreensive Open-architecture Space Mission Operations System*) pode ser facilmente adaptado para a adição de um ou mais veículos espaciais. O COSMOS foi desenvolvido dentro de um a universidade do Havaí com o propósito de imergir alunos na área espacial e prepará-los para o mercado de trabalho, viabilizando a indústria espacial no Estado do Havaí.

Por ser um framework *open source* uma vantagem do COSMOS é contar com a ajuda de uma comunidade de desenvolvedores ajudando rapidamente a melhorar o que já existe e adicionando novos recursos o que seria um pouco difícil em sistemas proprietários. Sua arquitetura consiste em um conjunto de bibliotecas que disponibilizam as seguintes funcionalidades:

- Funções matemáticas;
- Cálculos, variações de visualização do veículo existente e vistas orbitais;
- Suporte a protocolos;
- Suporte de hardware e simulação para o Operações *Test Bed* (auxílio nos testes e verificação de cargas de comando antes de serem enviados para os satélites);
- Armazenamento de dados distribuídos;
- Comandos em tempo real;
- Painel com luzes de advertência para indicar os níveis de alerta;

A seguir, são evidenciados na Figura 2.3 os componentes de software do COSMOS, preponderantes nesse processo de análise.

Figura 2.3 - Componentes de software do COSMOS.

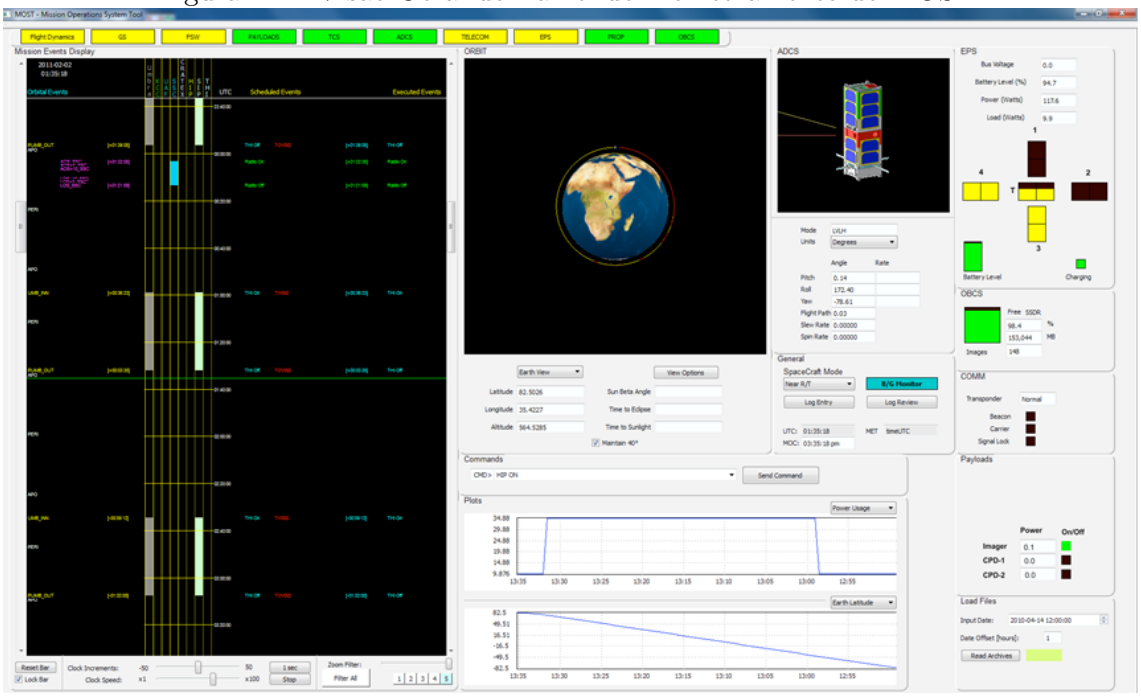


Fonte: Sorensen (2012).

Dentre os componentes da arquitetura citados anteriormente o MOST *Mission Operations Support Tool* é a mais importante, ela é uma ferramenta projetada para

apoiar em tempo “quase“ real as operação de apoio as missões. O MOST é a principal interface entre o operador e o satélite. No painel de monitoramento ilustrado na Figura 2.4 a seguir, é possível exibir a posição e atitude relacionada a um objeto que está em órbita, exibir dados de telemetria, passado e presente orbital e eventos do veículo espacial. Cada evento contém informações sobre um aspecto específico de um veículo espacial como tempo, botões de advertência, painel de eventos, órbita, *payloads*, gráfico de linha do tempo e interface para enviar comandos para o veículo.

Figura 2.4 - Visão Geral do Painel de Monitoramento do MOST.



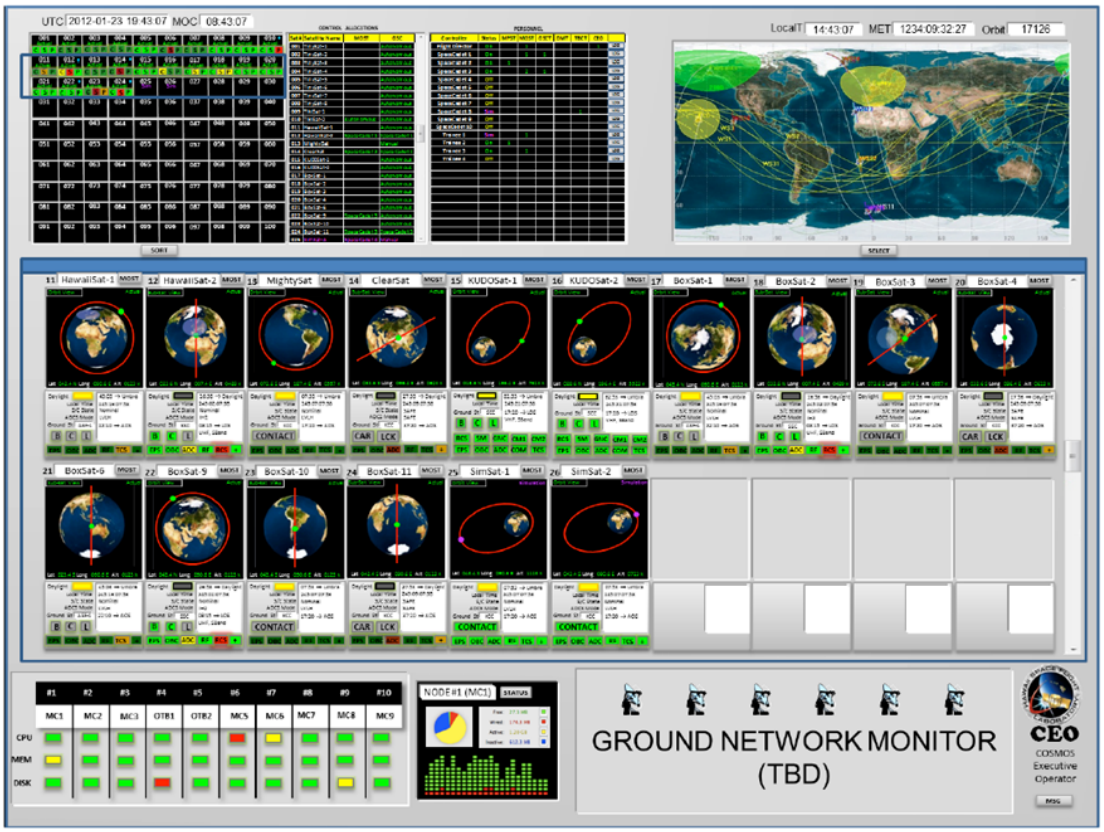
Fonte: Sorensen (2012).

A arquitetura do COSMOS também provê um componente de testes *Operations Test Bed* onde é possivel desenvolver e simular testes para cada componente de hardware e de software. Ele se integra com o centro de operação de missão e realiza testes de scripts de comando, treinamento de pessoal, ensaios de testes de scripts de comando, ensaios de missão e resolução de anomalias.

Devido à arquitetura de componentes reutilizáveis, o módulo de execução de operação *COSMOS Executive Operator* (CEO) permite operar mais de um veículo espacial ao mesmo tempo. Cada satélite possui sua própria sessão e visualizadas no mesmo painel de monitoramento. Assim é possível verificar a situação ou simulação de di-

versos veículos espaciais concomitantemente, como mostra a Figura 2.5 (Painel de Monitoramento do CEO).

Figura 2.5 - Painel de Monitoramento do CEO.



Fonte: Sorensen (2012).

2.1.3.2 Aplicações de Clima Espacial

O EMBRACE é um programa institucional no INPE cujo o 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., 2009, INPE).

Localizada no município de São José dos Campos, a infraestrutura computacional do EMBRACE no campus do INPE é responsável pelo armazenamento, processamento e disseminação de dados de diversos tipos de equipamentos geograficamente distribuídos como: magnetômetros, ionossondas, rede de GPS, dados de rádio e raios

X de explosões solares, entre outros. Para que os dados cheguem até à comunidade, diversos meios de disponibilização dos dados para cada equipamento foram desenvolvidos 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 (SANT'ANNA et al., 2014) é possível observar um modelo denominado *pipeline* que 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;

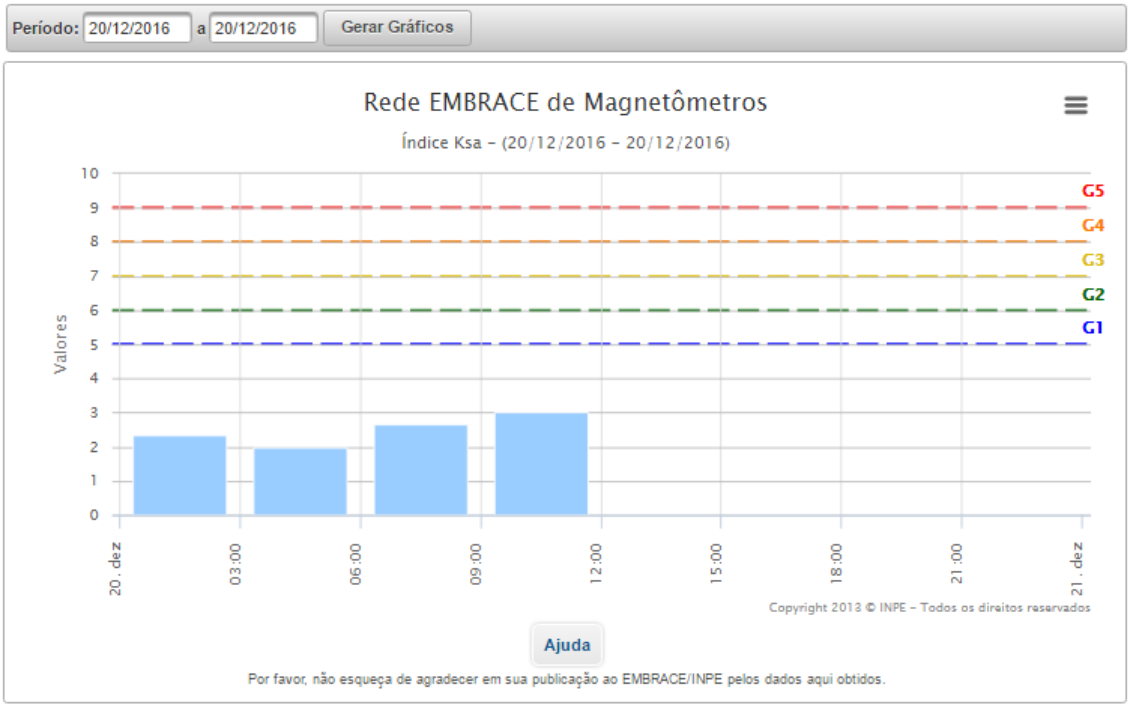
- **Processing Queue**: API (*Application Programming Interface*) encarregada de permitir duas ou mais aplicações se comunicarem por mensagens, garantindo a interoperabilidade e possibilitando a leitura, criação, envio e o recebimento de dados através de mensagens;

- **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. Além disso a aplicação possibilita ao usuário final definir um período de exibição do gráfico, como pode ser visto na figura 2.6.

Figura 2.6 - Aplicação web exibindo os índices de Ksa do magnetômetro.



Fonte: http://www2.inpe.br/climaespacial/portal/resumo-de-indices/

A Figura 2.7 retrata de forma sistemática tais informações, identificando todas as etapas do fluxo de dados.

Figura 2.7 - Modelo do fluxo de dados do Pipeline.

Fonte: Sant'Anna et al. (2014).

Em termos gerais, as principais vantagens nesse tipo de arquitetura de software distribuída é proporcionar o baixo acoplamento e reúso entre os componentes promovendo uma contínua interação entre si.

2.1.3.3 Sistema de Dados de Satélites Ambientais

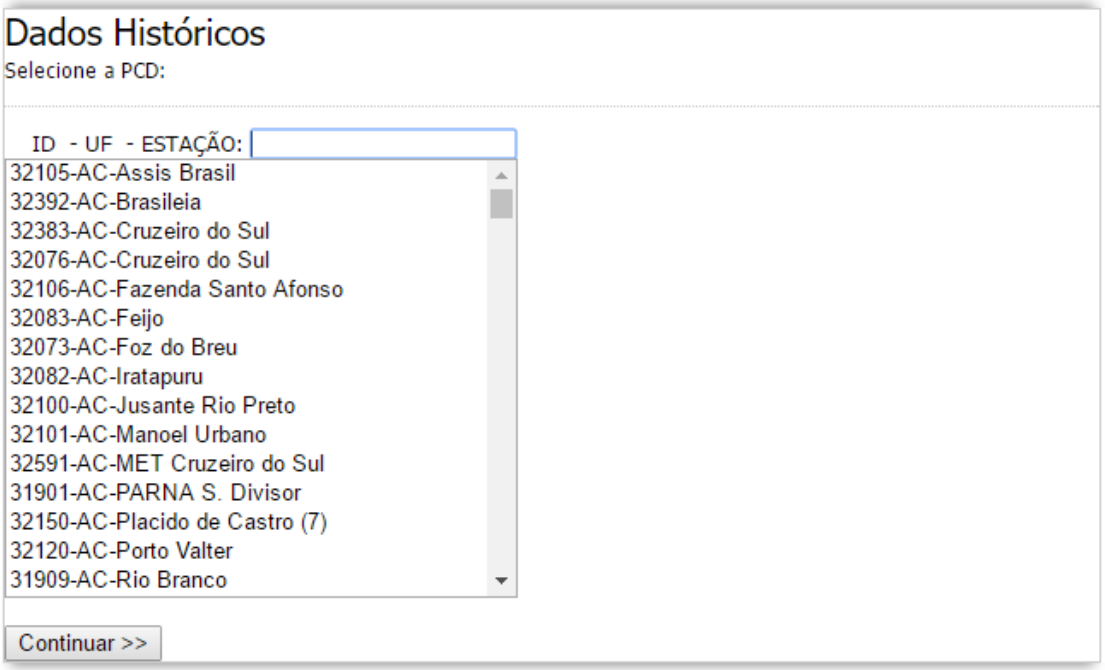
O sistema de coleta de dados ambientais é composto pelos satélites SCD1, SCD2 e CBERS 4, a rede de plataforma de coleta de dados (PCD´s) instalada em todo o território nacional, as estações de recepção de Cuiabá e Alcântara e um centro de Missão em Natal, RN. O Sistema Integrado de Dados Ambientais (SINDA) tem por objetivo realizar o processamento dos dados recebidos e distribuí-los aos usuários finais. Além disso os produtos e serviços gerados são utilizados por diversas instituições dos mais variados segmentos como: hidrologia, meteorologia, oceanografia (boias de deriva, boias ancoradas), química da atmosfera/gestão e fiscalização de recursos hídricos e meio ambiente, monitoração de nível de reservatórios de usinas hidrelétricas, alertas a desastres ambientais, transmissão e geração de energia elétrica e monitoramento ambiental (SANTOS et al., 2013).

Em termos funcionais, as PCD´s realizam o envio dos dados ambientais coletados aos satélites (através de uma rede local ou internet), para posterior transmissão às estações de recepção. Em seguida são enviados ao SINDA para a realização do processamento, armazenamento e disponibilização dos dados via internet com tempo máximo de 30 minutos após a recepção dos dados recebidos pelas PCD´s.

Em (SANTOS et al., 2013) é apresentada a infraestrutura computacional do Projeto SINDA localizada no Centro Regional do Nordeste em Natal, RN. Entre os principais requisitos do sistema estão as seguintes funções:

- Sistema de registro e gerenciamento das PCD´s que estão instaladas por toda a rede de coleta;
- Aquisição e processamento dos dados coletados pelas PCD´S retransmitidos pelos satélites SCD-1 e SCD-2 para as recepções de recepção;
- Inspeção da integridade dos arquivos obtidos pelos sensores de cada PCD;
- Processamento e conversão dos dados coletados para unidades de engenharia;
- Persistência dos dados coletados pelas PCD´s na base de dados;
- Distribuição dos dados através da internet. A Figura 2.8 evidencia um exemplo de disseminação dos dados coletados pelas PCD´s.

Figura 2.8 - Disponibilização dos dados históricos da rede PCD.

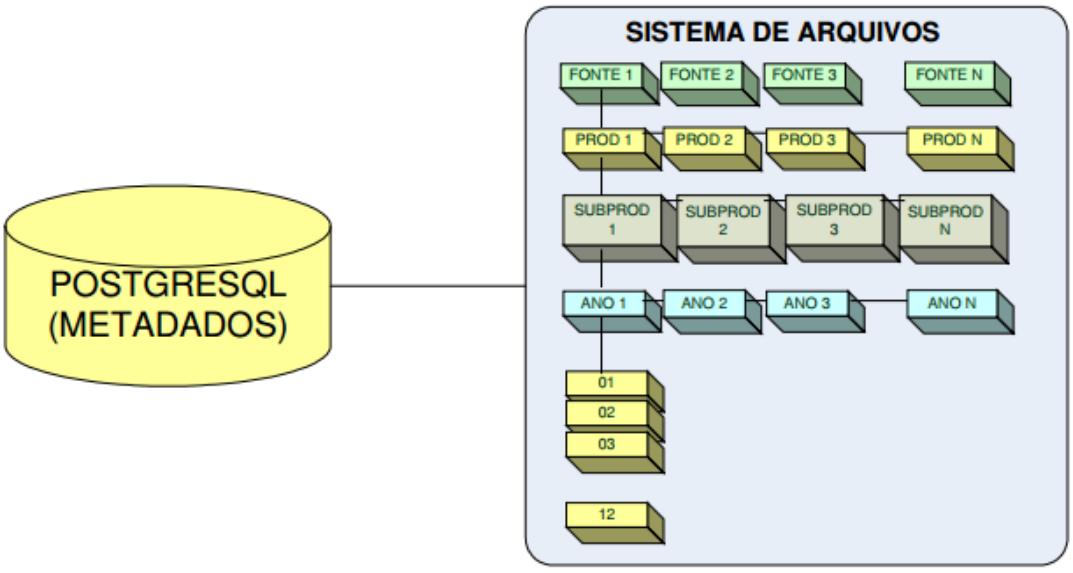


Fonte:
http://sinda.crn2.inpe.br/PCD/SITE/novo/site/historico/index.php.

2.1.3.4 Repositório de Metadados Espaciais

Regularmente os centros de distribuição de imagens de satélite recebem diariamente um enorme volume de dados. Portanto preocupações como escalabilidade, desempenho e disponibilidade da arquitetura tornam-se requisitos essenciais para o bom funcionamento dos sistemas em geral. Em (GARCIA; MOREIRA, 2006) é proposta a criação de uma base de dados de imagens de satélite para padronizar a forma de armazenamento dos arquivos bem como seu formato de gravação. Os sistemas de recepção recebem dados de diversos tipos de satélites como: GOES, NOAA METEOSAT, LANDSAT, entre outros. Os arquivos brutos recepcionados precisam ser processados a fim de serem transformados em diferentes tipos de produtos. A Figura 2.9 mostra o desenho da arquitetura onde diversos repositórios de sistemas de arquivos distribuídos são estruturados em árvores de diretórios e servirão como base para armazenar os dados brutos. Do outro lado um sistema gerenciador de banco de dados Postgres é responsável por armazenar os metadados do tipo: data da aquisição, origem do dado coletado (tipo de satélite), tamanho, dentre outros. Portanto com essa estrutura montada ficaria muito mais acessível a disponibilização de diversos tipos de produtos às comunidades acadêmicas.

Figura 2.9 - Sistema de Arquivos relacionado com o banco de metadados.



Fonte: Garcia e Moreira (2006).

De todos os sistemas citados anteriormente, a principal preocupação dos desenvolvedores e arquitetos dos sistemas é em relação às boas práticas do desenvolvimento de software implementando conceitos de reutilização, padrões padrões de projeto, escalabilidade e desacoplamento de componentes de software. Todavia não se verifica uma preocupação sobre a qualidade dos requisitos não funcionais como tempo de requisição de um processo, disponibilidade de serviço, o desempenho ao longo do tempo de uma aplicação, entre outros.

3 QUALIDADE DE SERVIÇO

A qualidade de um sistema e seus componentes pode ser mensurado quando estes, respondem aos requisitos de seus clientes de forma satisfatória. Segundo (ABNT NBR ISO 9001, 2008) qualidade pode ser definida como grau no qual um conjunto de características inerentes satisfaz a requisitos.

O conceito de qualidade de serviço é encontrado na literatura de diferentes formas. De acordo com (JONES, 2005), para que a definição do serviço seja bem-sucedida, deve haver suporte para o fluxo automático de informações entre arquitetos, designers, desenvolvedores, mantenedores e, o mais importante, o cliente.

"A qualidade subjetiva de um sistema de software baseia-se em grande parte em suas características não funcionaiss" (SOMMERVILLE, 2012).

"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)".

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 do serviços prestados.

"Como a medida de QoS dos serviços da web se torna cada vez mais importante para os clientes, essa informação precisa ser mantida por uma parte independente, ou seja, não está diretamente relacionada aos provedores ou aos clientes. Esta informação tem de ser mantida o mais objetiva possível e tem de ser mantida atualizada, o que implica a importância de manter a medida de QoS através de medições automatizadas. Tal medida permitiria que os clientes de serviços da web escolhessem um serviço web adequado em tempo de execução (baseado em atributos de QoS) e vincularem-se dinamicamente a um determinado serviço da Web (THIO; KARUNASEKERA, 2005)".

Em relação a qualidade de produtos e/ou serviços existem algumas definições, as quais objetivam saber se o produto esta em conformidade com as exigências dos clientes, se possui valor agregado, algo que produtos correlatos não possuem, relação custo/benefício, entre outros. Assim, pode-se entender que a qualidade é um conceito abstrato associado com a percepção de cada individuo.

A qualidade de serviço pode e deve ser aplicada em qualquer segmento onde exista a figura de um prestador de serviço, seja ele uma organização, um sistema, uma arquitetura orientada a serviço, uma infraestrutura de redes, dispositivo móvel, dentre outros.

Em uma rede comutada por pacotes podem ocorrer sucessões de diversos problemas relacionados às características do uso de serviços compartilhadas na rede, trafego dos dados, variação de atraso (jitter), identificação dos melhores caminhos por parte da rede e uma série de outros elementos que podem interferir no funcionamento e na qualidade dos componentes interconectados chegando até o último estágio que é o usuário final. Uma das formas para a prevenção consiste na identificação de requisitos relacionados que possuem características semelhantes, sendo possível criar políticas particulares para o tratamento de forma individualizada, garantindo de maneira apropriada a qualidade para cada aplicação, serviço ou dispositivo de forma isolada. A previsibilidade da QoS não irá eliminar todos os problemas sobre sistemas ou da rede mas sim mantê-los em níveis aceitáveis.

Na Figura 3.1 são listadas diversos tipos de aplicações e seus níveis de qualidade de serviço, segundo (TANENBAUM et al., 2003).

Figura 3.1 - Severidade dos requisitos de QoS para cada tipo de aplicação.

Aplicação	Confiabilidade	Retardo	Flutuação	Largura de banda
Correio eletrônico	Alta	Baixa	Baixa	Baixa
Transferência de arquivos	Alta	Baixa	Baixa	Média
Acesso à Web	Alta	Média	Baixa	Média
Login remoto	Alta	Média	Média	Baixa
Áudio por demanda	Baixa	Baixa	Alta	Média
Vídeo por demanda	Baixa	Baixa	Alta	Alta
Telefonia	Baixa	Alta	Alta	Baixa
Videoconferência	Baixa	Alta	Alta	Alta

Fonte: Tanenbaum et al. (2003)

Outros aspectos podem ser incorporados à qualidade de um serviço. Em (PARASURAMAN et al., 2002) é proposta uma ferramenta para a medição da qualidade dos serviços **SERVQUAL**. O modelo permite comparar a performance e/ou desempenho de uma organização frente à um padrão de excelência de forma a garantir aquilo que o cliente entende como sendo importante. Na lista abaixo são descritos os cinco principais grupos denominados **dimensões da qualidade**:

- **Tangibilidade**: aspectos físicos do serviços, como instalações, equipamentos e equipes;
- **Confiabilidade**: é o compromisso em entregar de fato o que foi acordado com o cliente;
- **Compreensão**: disposição e boa vontade em prestar um serviço com disponibilidade e presteza;
- **Segurança**: conhecimento e a qualificação das pessoas que executam o serviço;
- **Empatia**: atendimento e tratamento dispensado aos clientes.

3.1 Atributos de Qualidade de Serviço

De acordo com a (ISO/IEC JTC 1/SC 38, 2016) para cada tipo de arquitetura existe um conjunto de atributos e métricas de qualidade que podem ser aplicados com o propósito de mensurar a qualidade de serviço. A definição de métrica corresponde a um padrão de medição que decide as condições e regras para a realização da medição e compreensão dos resultados.

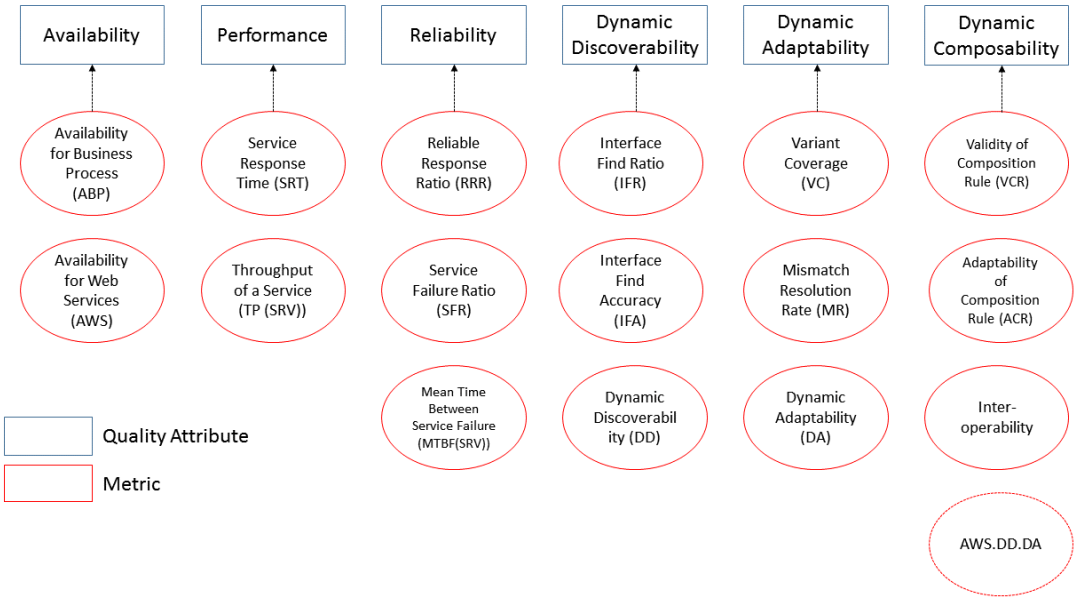
Na literatura podem ser encontrados diversos trabalhos relacionados aos atributos de qualidade a serem considerados em um serviço. Em (SHADE et al., 2012) relata que devido ao aumento da prestação de serviços através da internet, preocupações relacionadas à segurança da informação tornaram-se um requisito fundamental e que pode ser aplicada políticas em diferentes níveis dependendo do provedor de serviço. Com isso, são fornecidos quesitos importantes como autenticação, autorização, confidencialidade, criptografia, prestação de conta e segurança.

Segundo (O'BRIEN et al., 2007) a tecnologia de serviços web deve mencionar como diferentes atributos de qualidade de serviço serão impactados positivamente ou negativamente e seus respectivos compromissos envolvidos. Dessa forma neste tipo de ambiente faz-se necessário um estudo mais aprofundado a ser realizado em ambientes orientado a serviços.

Para objeto de estudo e aplicação neste trabalho foi utilizado como parâmetros os atributos e métricas de qualidade de serviço do trabalho de (CHOI et al., 2007). O autor identifica um conjunto de características únicas em arquiteturas orientadas a serviço como, baixo acoplamento, serviços de contratos bem definidos, descoberta

dinâmica, composição e adaptação em tempo de execução e reusabilidade com o propósito em derivar as métricas de QoS. Dessa forma, baseados nessas características, seis atributos e métricas de qualidade foram definidos conforme mostra a Figura 3.2.

Figura 3.2 - Atributos e métricas de QoS para os provedores de serviço.



Fonte: Adaptada de Choi et al. (2007)

Outro trabalho que foi utilizado como semântica para a interpretação dos atributos e métricas de QoS pode ser observado em (GARCIA; TOLEDO, 2006). Ainda, segundo os autores, QoS é uma combinação de inúmeras características não funcionais. Para (TOSIC et al., 2005) o monitoramento da QoS ajuda a atingir níveis de qualidade dos serviços prestados aos consumidores. Além disso, auxiliar provedores de serviço na detecção de problemas, adaptação a mudanças como, taxas referentes a custos e desempenho dos serviços prestados. A Figura **??** apresenta a descrição os atributos de QoS.

Tabela 3.1 - Atributos de QoS para *web services*.

Tempo de Resposta	O tempo que um serviço leva para completar uma tarefa.
Latência	O tempo gasto para iniciar a manutenção de um serviço pedido.
Taxa de Transferência	A taxa de processamento de uma solicitação suportada por um serviço.
Escalabilidade	Aumento da taxa de transferência em um dado intervalo de tempo.
Capacidade	Número de solicitações simultâneas permitidas.
Disponibilidade	Porcentagem de tempo que um serviço está em operação.
Confiabilidade	O tempo de continuidade de um serviço em funcionamento e transição de um serviço para o estado de funcionamento.
Precisão	A taxa de erro de serviço ao longo de um intervalo de tempo.
Robustez	O nível de resiliência de serviço para entradas e sequências de invocações incorretas.
Estabilidade	A taxa de erro de serviço ao longo de um intervalo de tempo.
Precisão	A taxa de erro de serviço ao longo de um intervalo de tempo.
Custo	Medição do custo na utilização do serviço.
Segurança	Define se o serviço oferece mecanismos de confidencialidade, integridade e autenticação.
Mensagem Confiável	Determina se um serviço oferece mecanismos para garantir a entrega confiável de mensagens.
Integridade	Determina se o serviço suporta propriedades transacionais.
Interoperabilidade	Determina se o serviço é compatível com perfis de interoperabilidade.

Fonte: Adaptada de Garcia e Toledo (2006)

3.2 Acordo de Nível de Serviço

Um serviço não é uma atividade isolada, é um conjunto de recursos relacionados a fim de satisfazer uma ou mais necessidades de usuários, clientes, fornecedores, dentre outros. Exemplos destes na área computacional podem ser os serviços de impressão, onde se tem a impressora, a rede e suas estações, o sistema operacional e o gerenciador de impressão. Todos esses elementos compõem o serviço de impressão, pois todos os itens envolvidos no processo são partes fundamentais para prestação do serviço. Entender o conceito de serviço é essencial para o estabelecimento de um acordo de nível de serviço, também conhecido como *Service Level Agreement (SLA)*. Na versão 3 do ITIL *Information Technology Infrastructure Library* (ITIL, 2007), um serviço é um meio de entregar valor aos clientes, colaborando com as metas que os clientes querem alcançar, sem ter que assumir os custos e riscos.

Um SLA é qualquer relação comercial entre um fornecedor e cliente, a fim de garantir o resultado das contratações por parte do cliente. No SLA são definidos os papéis e responsabilidades de cada parte e as condições de entrega e manutenção em uma relação de demanda de serviço. No trabalho de (TRIENEKENS et al., 2004) o objetivo de um SLA é fazer a ponte entre o prestador de serviço, usuários e/ou clientes. No entanto, existem muitos problemas e questões não resolvidas com relação à especificação e à quantificação dos SLA´s. Em (PHILIP et al., 2008) as organizações procuram desenvolver SLA´s por diversas razões, uma delas está entre as definições a respeito das responsabilidades sobre quem e/ou qual das partes irá ficar responsável pelo serviço, e ainda na maioria das vezes o mais importante é o que cada parte não irá fazer.

Outra definição sobre SLA pode ser encontrada no glossário do ITIL (ITIL, 2007) [1] "Um acordo entre um provedor de serviços de TI e um cliente. O SLA descreve o serviço de TI, documenta metas de nível de serviço, e especifica as responsabilidades entre o provedor de serviços de TI e o cliente. Um único SLA pode cobrir múltiplos serviços de TI ou múltiplos clientes."

Um SLA pode ser criado e aplicado para qualquer tipo de serviço, seja ele sistemas web, standalone, SOA, serviços web, entre outros. Em (PATEL et al., 2009) pode-se observar a importância de um SLA de computação na nuvem, pois devido ao dinamismo dos sistemas em nuvem o monitoramento contínuo dos atributos de qualidade

[1] ITIL é framework para o gerenciamento de serviços de TI. Através de melhores práticas o ITIL fornece processos a serem implantados apropriadamente em organizações da área de T.I buscando qualidade nos serviços prestados.

de serviço garantem um acordo de nível de serviço satisfatório para ambas as partes. Para ajudar nos processos de operação dos serviços, o ITIL disponibiliza uma abordagem para a implementação de melhores práticas na tentativa dos serviços de entrega agregarem valor ao cliente de forma efetiva. Os processos envolvem três itens:

- **Incidente**: é qualquer acontecimento, interrupção, anomalia de um serviço. Exemplo, um email que não é enviado devido ao servidor de email que está fora do ar, a indisponibilidade de um site, um serviço de impressão que não funciona, ou seja, o serviço pelo qual foi designado não atende as expectativas do contratante. Deve ser prioritário o tratamento do erro, pois gera prejuízo aos envolvidos. Pode-se criar processos em como atender essas incidências com o objetivo de diminuir as ocorrências.
- **Evento**: é uma ocorrência que possui algum tipo de informação auxiliando no processo de manutenção preventiva de um sistema. Exemplo, criação de alertas que devem ser enviados ao gerente responsável da área sobre um estouro de memória de um processo que está rodando em paralelo, aumento do uso de utilização da CPU, controle do limite de armazenamento do banco de dados a fim de evitar um *full storage*, entre outros. O monitoramento desses eventos são muito importantes para a garantia da qualidade, pois o diagnóstico precoce pode evitar um problema maior tornando-se um incidente.
- **Requisição**: é uma solicitação, pode ser por um usuário pedindo um acesso a um sistema ou funcionalidade, criação de email, ou algum tipo de mudança padrão conhecida pelo processo.

Com essas características em um SLA, é possível tratar de forma isolada todas as ocorrências que possam acontecer durante a vigência do contrato, como por exemplo: onde ocorreu, quando, por quem, qual o impacto, grau de prioridade, status do sistema, exceções (o que não será feito), penalidades, dentre outros. Dessa forma auxilia no funcionamento do sistema.

- **O quê**: - Qual o serviço apresenta problemas, impressão, e-mail, ERP, indisponibilidade do sistema, site, entre outtros;
- **Quando**: - quando ocorrer um chamado, qual o período que tenho que dar o atendimento, data, hora, vigência do contrato, o suporte é 24h?

- **Quanto**: - qual será o tempo gasto para realizar a manutenção e resolver o problema reportado?

- **Como**: - o atendimento à ocorrência será por telefone, chat, existe algum processo automatizado?

- **Quem**: - quem vai fazer; qual o perfil da pessoa; na falta dela quem vai atender?

Um SLA deve descrever de forma clara e adequada as necessidades particulares de cada parte do sistema. Exemplo, se o tempo de download de um catálogo de imagens de um sistema disponível na internet for um requisito crítico, então o grau de importância para esse item deve ser observado atentamente, garantindo de forma satisfatória aos usuários utilizadores do catálogo de imagens.

Além disso, outras qualidades de um acordo de nível de serviço podem ser observadas, como o aumento da qualidade na entrega, o controle mais apurado das condições em que são ofertados os serviços para um aumento da QoS, o acompanhamento, o tempo de utilização, a necessidade de melhorias, entre outras. Contudo inevitavelmente há uma redução de custos, pois o acompanhamento permite que possa saber quais serviços estão sendo mais utilizados, quais horários, quantidade de acesso de usuários. Por exemplo, quando se contrata um serviço como a Amazon Web Services, Azure Microsoft ou qualquer serviço onde tem arquitetura do tipo IaaS (Infraestrutura como um serviço *Infrastructure as a Service*), PaaS (plataforma como um serviço *Platform as a Service*) ou SaaS (Software como um serviço *Software as a service*) e o cliente paga por cada serviço utilizado, com as métricas de serviços utilizados pode-se configurar uma redução de carga dos serviços menos utilizados visto que os serviços não exigem tanto poder computacional. Um exemplo, um sistema de backup de log onde os dados históricos precisam ser armazenados por um longo período mas o acesso a eles é relativamente baixo. Então opta-se por armazená-los em um servidor de baixo custo e com menos recursos de hardware.

A produção de métricas para gerenciamento da qualidade permite a medição, quanto ele está sendo eficiente, até quando pode-se usar com os recursos de hardware atuais, qual a capacidade, taxas de erros, enfim inúmeras métricas que podem ser gerenciadas sobre QoS, até mesmo cobrar do provedor sobre algum requisito não atendido.

Porém, a desvantagem em relação à contratação de um acordo de nível de serviço envolve questões relacionadas ao custo da aquisição ou fornecimento. Entretanto

pode-se criar um equilíbrio definindo exigências menores a partir da disponibilidade financeira do cliente (relação menor exigência X menor valor de contrato).

Na inexistência de um acordo, fica sub entendido que será fornecido suporte a qualquer momento e circunstância. Ao negociar e definir um SLA, são perceptíveis uma série de melhorias, tais quais: a comunicação com o cliente, a visibilidade no mercado, os procedimentos a serem seguidos, a diminuição de chances de processos jurídicos. Um SLA é muito mais que um papel com prazo de atendimento e resolução de problemas, no acordo deve deixar claro como ele será reportado, mensurado e evoluído continuamente. Portanto uma conduta viável é criar o(s) acordo(s) junto com a definição de requisitos, dessa forma é possível já identificar qual vai ser a estrutura necessária para o atendimento.

Algumas características importantes na hora de descrever um acordo de nível de serviço são:

- **Realista**: o projeto deve ser factível de execução, objetivos e condições devem ser possíveis de ser alcançadas;
- **Metas tangíveis**: preciso definir quanto de disponibilidade o serviço tem, quanto tempo um SLA deve ser atendido, tempo máximo de espera de um chamado, tempo mínimo, volume de chamadas aceitáveis, ou seja todos os indicadores devem ser alcançáveis;
- **Considerar acionamento de terceiros**: caso o provedor não possua um especialista para atender alguma solicitação de um serviço que esteja no acordo, então o mesmo pode integrar um serviço terceirizado, como email, ERP (*Enterprise Resource Planning*) [2], segurança, entre outros.

Como citado nos parágrafos anteriores um acordo de nível de serviço pode ser criado para alinhar as expectativas entre prestadores de serviço e consumidores a respeito da QoS, prioridades, responsabilidades e outros. É muito comum que os acordos sejam descritos de forma textual pelos stakeholders do projeto para que possam ser lidos por humanos. Nele pode conter tabelas, gráficos e todo o tipo de informação que possa ser acrescentada ao documento. Porém, como garantir a qualidade de serviço em sistemas autônomos onde não há interação direta de um usuário e os serviços provedores e consumidores são máquinas (M2M). Como garantir a disponibilidade de um servidor, ou um serviço cujo o tempo de resposta é um requisito crítico e não

[2] são sistemas que integram todos os dados e processos de uma organização em um único sistema

permite que o tempo da requisição seja menor que N? Para isso existem algumas soluções principalmente em computação em nuvem. Em (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*) para ser lida por máquina *Machine Readable SLAs* durante o cliclo de vida de um SLA podendo de forma dinâmica criar negociações automatizadas.

Outro exemplo de um SLA cujo um modelo de especificação foi criado a fim de permitir negociações automatizadas pode ser visto em (BAKALOS et al., 2015). O documento mostra um formato em XML *schema* onde todo o contrato é apresentado em linguagem marcação recomendada pela W3C para a criação de documentos com dados organizados hierarquicamente. Permitindo dessa forma que qualquer aplicação consumidora independente de tecnologia possa conseguir ler o acordo estabelecido. A linguagem de saída estruturada pode ser vista na Figura 3.3.

Figura 3.3 - SLA XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="abstractMetric">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="metricDefinitions">
               SLALOM SLA specification and reference model - a - D3.2
               © Institute of Communication and Computer Systems and other members of the SLALOM consortium, 201561
               <xs:complexType>
                  <xs:annotation>
                     <xs:documentation>These are the parameter definitions for the Abstract Metric</xs:documentation>
                  </xs:annotation>
                  <xs:sequence>
                     <xs:element name="parameterId" type="xs:string" />
                     <xs:element name="name" type="xs:string" />
                     <xs:element name="type">
                        <xs:simpleType>
                           <xs:restriction base="xs:string">
                              <xs:enumeration value="Integer" />
                              <xs:enumeration value="Decimal" />
                              <xs:enumeration value="String" />
                              <xs:enumeration value="Boolean" />
                              <xs:enumeration value="Byte" />
                           </xs:restriction>
                        </xs:simpleType>
                     </xs:element>
                     <xs:element name="note" type="xs:string" minOccurs="0" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>

            <xs:element name="ruleDefinitions" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:annotation>
                     <xs:documentation>These are the rule definitions for the Abstract Metric</xs:documentation>
                  </xs:annotation>
                  <xs:sequence>
                     <xs:element name="ruleId" type="xs:string" />
                     <xs:element name="name" type="xs:string" />
                     <xs:element name="ruleExpression" type="xs:string" />
                     <xs:element name="consequenceOfViolation" type="xs:string" minOccurs="0" />
                     <xs:element name="note" type="xs:string" minOccurs="0" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="dependencyDefinitions" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:annotation>
                     <xs:documentation>These are the dependency definitions for the Abstract Metric</xs:documentation>
                  </xs:annotation>
                  <xs:sequence>
                     <xs:element name="dependencyId" type="xs:string" />
                     <xs:element name="name" type="xs:string" />
                     <xs:element name="dependentMetricId">
                        <xs:simpleType>
                           <xs:restriction base="xs:string">
                              <xs:pattern value="^\w+(;\w+)*$" />
                           </xs:restriction>
                        </xs:simpleType>
```

Fonte: Adaptada de Bakalos et al. (2015)

Em se tratando de serviços web, no trabalho de (LUDWIG et al., 2003) é apresentado um modelo abstrato para o gerenciamento em tempo de execução de um WSLA (*Web Service Level Agreement*). O modelo é dividido em três grupos de funcionalidades de medição e gerenciamento:

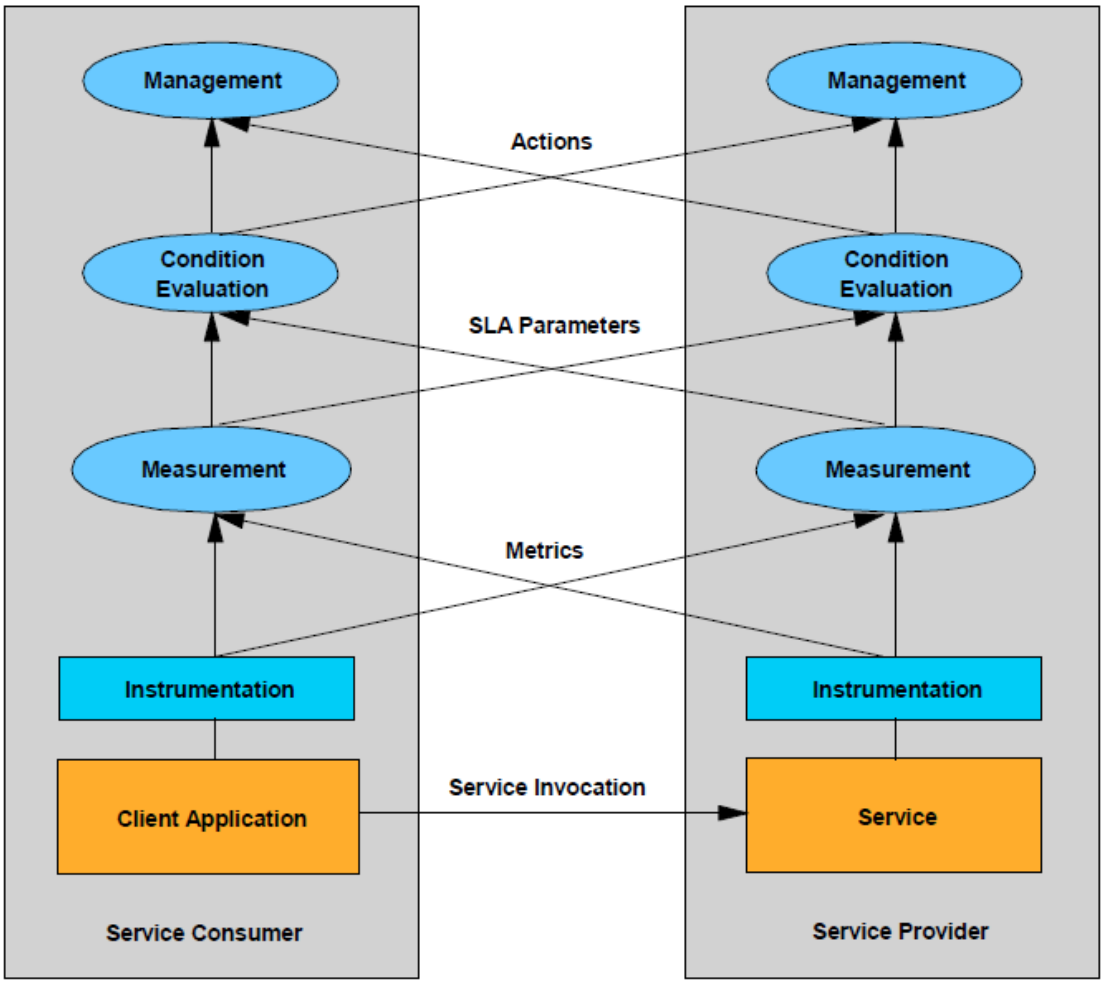
- **Medição**: essa funcionalidade realiza tarefas como calculo das métricas de alto nível como o tempo de resposta de um *cluster* de servidores.
- **Conjunto de métricas**: São as métricas definidas para a avaliação fun-

cionalidade de medição;

- **Avaliação da condição**: são as garantias e violações do acordo de nível de serviço baseadas nas medições realizadas.

A leitura do WSLA onde sua estrutura é descrita em XML tem as seguintes composições: as partes envolvidas, como nome do fornecedor dos serviços ou do cliente; as definições do serviço comum às duas partes, quais serão as métricas utilizadas, como será realizada a medição e; as obrigações e condições para realizar o serviço. A Figura 3.4 apresenta os níveis de interações entre os provedores e consumidores dos serviços.

Figura 3.4 - Gerenciamento em tempo de execução de um WSLA.



Fonte: Ludwig et al. (2003)

3.3 Ferramentas de avaliação da QoS

Pode-se concordar que é impossível controlar um satélite sem o auxílio dos instrumentos, estações, e os centros de comunicação e controle. Como seria se houvesse apenas o operador enviando comandos para o satélite sem saber como está o funcionamento da carga útil, a conexão com as estações de recepção, o status do sensor de temperatura do satélite, entre muitas outras definições, a demora na detecção de falhas seria muito maior. O mesmo acontece quando se administra uma rede de computadores sem realizar o monitoramento de toda sua estrutura. Portanto existem diversas ferramentas no mercado que permitem monitorar periodicamente todo o parque computacional de uma organização, e utilizar os dados coletados para a geração de gráficos de suma importância, garantindo o total funcionamento de todos os dispositivos da arquitetura.

Devido à diversidade das ferramentas a escolha deve ser feita de acordo com a necessidade da organização, ou ainda a utilização de mais de uma ao mesmo tempo realizando o monitoramento. A maioria disponibilizam serviços de monitoramento de: tempo de resposta de um servidor, site, *web service*, um sensor, e muitos outros; quantidade de 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, entre muitos outros. O acompanhamento do comportamento dos dispositivos e serviços permitem criar ações de predição de falhas e ainda pode-se configurar o ambiente para agir juntamente com um SLA para atuar na automatização de serviços. Dessa forma o monitoramento realizado de maneira eficaz atua diretamente na garantia da qualidade de serviço de todo o arcabouço computacional de uma organização, tornando os sistemas altamente robustos oferecendo um *downtime* [3] quase zero.

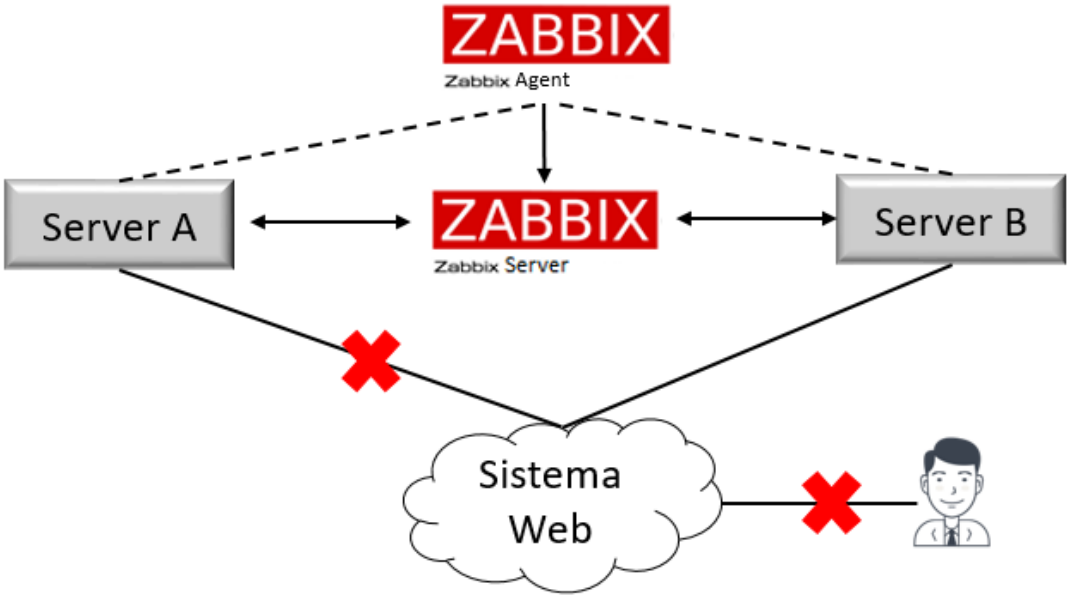
O CACTI é uma ferramenta de monitoramento que pode ser configurada facilmente através de um interface web. Nele é possível criar *scripts* personalizados para a realização da coleta dos dados. Todos os dados coletados ficam armazenados em uma base de dados podendo ser obtidos à qualquer momento pelo administrador do sistema.

O Zabbix é um software de monitoramento de código aberto e possui mais de cem parceiros no mundo todo que colaboram com o desenvolvimento e melhoria. Nele é possível realizar o monitoramento de diversos tipos de dispositivos de forma gra-

[3] é o tempo de parada de um serviço. Através dessa métrica é possível identificar o quanto custa financeiramente o tempo de inatividade na empresa.

tuita. Ele pode enviar SMS, email, permite a comunicação com protocolos do tipo HTTP, MQTT, TCP e outros. Ainda, é possível trabalhar com gatilhos (*trigger*) e automatizar tarefas. A Figura 3.5 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. Outra característica muito interessante e importante do Zabbix, é que ele permite realizar o monitoramento sem sobrecarregar os serviços monitorados.

Figura 3.5 - Monitoramento dos servidores usando o Zabbix.



O Zabbix possui dois modos de verificação:

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

- **Passivo**: o servidor do Zabbix vai diretamente no sistema alvo monitorado e coleta a informação.

Caso o Zabbix Server esteja realizando uma atualização, parado ou em manutenção, os agentes continuam coletando os dados e quando o servidor volta a ficar disponível os dados são enviados. Outra funcionalidade em relação ao Zabbix é a verificação do quanto de memória o servidor vai usar em um determinado período, dessa forma é possível antecipar o quanto e quando investir na infraestrutura do sistema.

Para realizar o monitoramento de web sites a ferramenta gratuita do Google Analitycs (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, monitoramento da analise de desempenho, controle de tráfego, realização da otimização do conteúdo a fim de aumentar as visitas do web site e diminuição do número de rejeição, localizações geográficas dos usuários, tipo de sistema operacional do cliente, dentre muitos outros.

Segundo o GA é possível entender como os usuários chegam até o seu site. Eles podem vir de duas formas, através de buscas orgânicas e pagas, cada uma delas pode ter um tipo de comportamento, logo é possível otimizar o web site para cada um desses públicos e ter um melhor aproveitamento. Outra importante característica da ferramenta é a análise de metas de destino, ela mapeia os usuários que chegaram até a pagina de destino do seu site. Um exemplo pode ser aplicado em um site que permite a realização de download de imagens de satélite e a página de destino é onde o usuário final recebe uma mensagem informando que o download foi realizado com sucesso. Assim o objetivo desse site é fazer com que todos os usuários consigam realizar todos os procedimentos necessários até a finalização do download.

Na internet é possível encontrar diversas outras ferramentas analíticas que são específicas para cada tipo de necessidade a fim de realizar o monitoramento de toda a arquitetura computacional de uma organização.

4 ARQUITETURAS DE SISTEMAS

Assim como qualquer metodologia que se propõe a ser construída, informações relevantes devem ser levantadas e discutidas previamente. Na engenharia automotiva para a concepção da arquitetura de um veículo, requisitos são levantados, um projeto é estabelecido e aprimorado com auxílio de diversas ferramentes técnicas que permitem a visualização preliminar, testes e diversos cenários de simulações, dentre outros. Em um projeto de arquitetura e edificação são compreensíveis as fases do estudo preliminar, anteprojeto e/ou projeto de aprovação, execução e assistência à execução da obra dentre os quais caracterizam como coleta de informações e desenvolvimento de estudos/serviços técnicos para a emissão de produtos finais.

Com o crescente avanço tecnológico de hardware e software o uso da tecnologia computacional atualmente é abordado em praticamente todos os campos de trabalho. Na década de 70 computadores ocupavam salas inteiras e seu custo era muito alto. Como é o caso do ENIAC (MCCARTNEY, 1999) desenvolvido nos anos de 1942 a 1945 na Universidade da Pensilvânia (EUA). Nessa época os estudos eram baseados em sua maioria no hardware. Segundo (DEITEL; DEITEL, 1999) o computador é um dispositivo capaz de realizar cálculos e tomar decisões lógicas com uma velocidade milhões ou mesmo bilhões de vezes mais rápido do que os humanos. Entretanto inicialmente os primeiros computadores realizavam apenas um trabalho ou tarefa por vez. Na década seguinte computadores já podiam se conectar através de linhas telefônicas e redes locais. Devido às inovações nos componentes de hardware logo o desenvolvimento do software começou a ser necessário para suprir e acompanhar os avanços.

Nesse contexto o evolutivo papel do software foi ganhando destaque e no que tange ao desenvolvimento de software onde antigamente era apenas um pequeno pedaço de código, com o passar dos anos novas tecnologias foram surgindo e logo novas demandas mais complexas foram aparecendo. Assim começou a se pensar em métodos e metodologias para o desenvolvimento. A partir deste cenário a arquitetura de software surge com o objetivo de fornecer um conjunto de elementos arquiteturais que possuem alguma organização, identificando os principais aspectos estruturais e seus relacionamentos.

Em um projeto computacional onde visa desenvolver um sistema qualquer, uma arquitetura de software bem estruturada permite definir pontos cruciais, tais como: organização da composição dos componentes, seus relacionamentos e funcionalidades, protocolos de comunicação e ainda uma concepção abstrata de todo o sistema

e do controle necessário para seu desenvolvimento.

Para o estilo e a estrutura da arquitetura particular a ser definida para um determinado sistema deve-se estabelecer uma relação de interdependência com os requisitos não funcionais, devido ao contínuo vínculo existente (SOMMERVILLE, 2012). De modo a facilitar a compreensão, tem-se a seguir um detalhamento destes requisitos não funcionais:

- **Desempenho**: a arquitetura deve ser projetada conforme o ambiente em que o sistema irá se encontrar. Caso a comunicação da rede seja um requisito instável é indicado realizar as operações em um pequeno número de componentes e no mesmo computador, evitando a comunicação de transmissão pela rede, criando um número maior de componentes, e, por conseguinte, aumentando o nível de granularidade do sistema;

- **Proteção**: a arquitetura deve ser estruturada nas camadas que necessitam de mais proteção e estejam protegidas internamente com níveis de autenticação, autorização, gerenciamento de identidade, dentre outros.

- **Segurança**: caso a segurança do sistema seja um requisito crítico, a arquitetura deve ser projetada onde a camada de segurança possa interfacear com outras camadas garantindo a proteção das mesmas se necessário e/ou disponibilizando funcionalidades que permitem assegurar todo o sistema. Exemplo: um botão de desligar de maneira segura em caso de falha;

- **Disponibilidade**: a arquitetura deve ser projetada com componentes redundantes garantindo que sempre haverá um ou mais componentes de backup em caso de falha, assumindo o lugar sem a necessidade de parar o sistema;

- **Manutenção**: caso a manutenção seja um requisito essencial no sistema, a arquitetura deve ser projetada com um baixo nível de granularidade onde possam ser reutilizados os componentes e rapidamente alterados se necessários.

A importância da definição da arquitetura de software resulta em diversos fatores que auxiliam a desenvoltura do processo de concepção do objetivo proposto. Elementos como generalização e representação em alto nível permitem trazer clareza e facilita a comunicação entre os stakeholders, prematuram decisões relevantes do projeto como

regras de negócio, escolha da infraestrutura, definições de padrões na implementação e implantação do sistema. Contudo todas as decisões arquiteturais de um projeto são fatores relevantes para alcançar atributos de qualidade, e conduzem a evolução do processo de desenvolvimento do sistema.

Na concepção arquitetural de um sistema de software diversos fatores podem influenciar diretamente seu desenvolvimento. Questionamentos em diferentes aspectos tornam-se pertinentes, com destaque para um planejamento quanto à presença, disponibilidade, quantidade e custos de manutenção de desenvolvedores (Júnior, Pleno, Sênior); definição de estratégias de venda e pós-venda, tempo hábil de propaganda, recursos financeiros palpáveis para investir; além de aspectos voltados para manutenabilidade do sistema e devido atendimento aos requisitos de desempenho, segurança e confiabilidade. Diante do exposto, todas essas inquietações devem apresentar um equiparado grau de importância durante todo o processo de desenvolvimento do sistema com o objetivo de que o mesmo seja eficiente e alcance os resultados esperados.

A realização de um sistema de software supera a escolha de uma linguagem, estruturas de dados, testes e algoritmos bem escritos, sendo de tamanha complexidade esse processo de criação, modificação e alcance de desempenho do sistema para uma posterior utilização pelos usuários finais. Atributos de qualidade como segurança, interoperabilidade, portabilidade e confiabilidade devem ser requisitos primordiais da arquitetura, todavia, torna-se crucial mensurar se as qualidades estão sendo satisfatórias.

A importância de avaliar uma arquitetura de software se dá porque muitas organizações se frustram ao tentar realizar seus negócios, pois seus sistemas apresentam diversos problemas, como: baixa qualidade, falhas e comportamentos imprevisíveis, além de julgamentos dos usuários quanto ao sistema ser inutilizável. Este foi tema de um artigo publicado ((SEI)., 2016) e que trata exatamente sobre a questão da avaliação da arquitetura de software.

4.1 Definições

Diversas definições de arquitetura de sistemas podem ser encontradas na literatura e algumas destas definições são apresentadas a seguir.

A arquitetura de software é de extrema importância, pois afeta o desempenho e a robustez, bem como a capacidade de distribuição e de manutenabilidade de um sistema (BOSCH, 2000). Como discute BOSCH, os componentes individuais implementam os

requisitos funcionais do sistema. Por outro lado, os não funcionais dependem da arquitetura do sistema, além da forma, como os elementos encontram-se dispostos e do nível de comunicação existente.

"O projeto de arquitetura é um processo criativo no qual você projeta uma organização de sistema para satisfazer aos requisitos funcionais e não funcionais de um sistema. Por ser um processo criativo, as atividades no âmbito do processo dependem do tipo de sistema a ser desenvolvido, a formação e experiência do arquiteto de sistema e os requisitos específicos para o sistema. Por isso, é útil pensar em projeto de arquitetura como uma série de decisões, em vez de uma sequência de atividades" (SOMMERVILLE, 2012).

"Uma arquitetura de referência de software é uma arquitetura genérica para uma classe de sistemas de informação que é usada como base para um projeto de arquiteturas concretas desta classe." (ANGELOV et al., 2009).

"A arquitetura de software de um programa ou sistema de computação é a estrutura ou estruturas do sistema, que compreendem elementos de software, as qualidades externamente visíveis desses elementos e as relações entre eles" (CLEMENTS, 2002).

"A arquitetura serve como modelo para o sistema e para o projeto que o desenvolve. Ela define as atribuições de trabalho que devem ser realizadas pelas equipes de projeto, bem como a sua implementação. A arquitetura é a portadora principal de qualidade do sistema, como o desempenho, a modificabilidade e a segurança, sendo que nenhuma dessas pode ser alcançada sem uma visão unificada da arquitetura. Arquitetura é um artefato para análise precoce para se certificar de que a abordagem de design irá produzir um sistema aceitável. Além disso, a arquitetura é a chave para o entendimento, a manutenção e os esforços de mineração da pós-implantação do sistema. Em suma, a arquitetura é a cola conceitual que mantém todas as fases do projeto juntos para todos os seus muitos stakeholders" (CLEMENTS et al., 2002)

"Arquitetura é a organização fundamental de um sistema incorporado em seus componentes, relacionamentos com o ambiente, e os princípios que conduzem seu design e evolução." (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC. IEEE, 2000)

Em virtude dos termos abstratos e de englobar diversos conceitos complementares entre si, verifica-se na literatura inúmeras definições sobre a temática de arquitetura, as quais contribuem diretamente para o desenvolvimento de estudos inovadores e de

maior impacto quanto aos resultados esperados.

4.2 Visões Arquiteturais

Inicialmente entende-se por visão uma projeção de um modelo sobre uma determinada perspectiva. É a forma como os diversos stakeholders que desempenham papéis diferenciados no processo de desenvolvimento de software enxergam o problema, além do modo como cada entidade e/ou componente da arquitetura pode ser observada.

Podemos extrair um exemplo de visão na arquitetura de um sistema de coleta, processamento e disponibilização de dados de clima espacial. De uma forma simplificada têm-se dois tipos de stakeholders: o time de desenvolvimento cujas preocupações são em como os componentes irão se comunicar com outros na aplicação, melhores práticas de desenvolvimento, utilização de padrões de projeto, tecnologias, classes e seus relacionamentos, restrições de acesso dentre muitos outros; enquanto que a equipe de infraestrutura que está preocupada em como o sistema será distribuído, se há criticidade no requisito disponibilidade e performance, backup periódico dos dados, tempo de resposta das requisições, dentre outros.

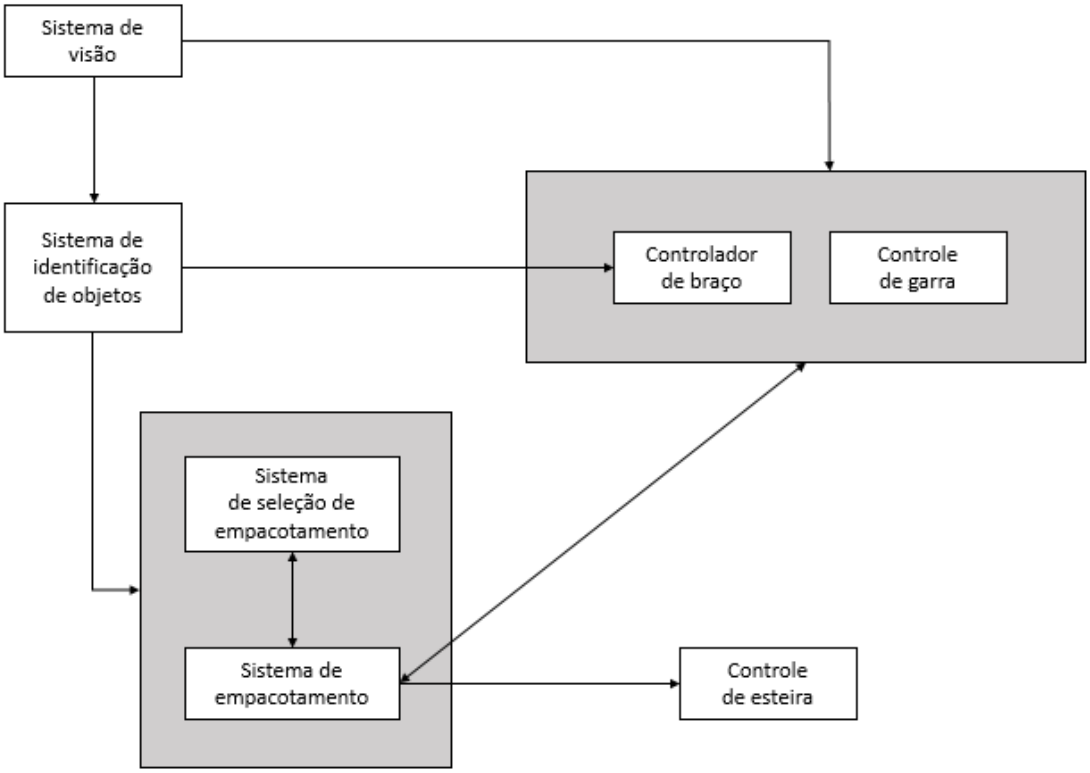
Assim, ressalta-se que cada visão de uma arquitetura é usada para propósitos diferentes por cada stakeholder. Na literatura podemos encontrar diferentes visões para representar uma arquitetura. Em (KRUCHTEN, 1995) no seu modelo de visão 4+1 é proposto a utilização de casos de uso ou cenários para representar o uso das quatro visões fundamentais da arquitetura. São elas:

- **Visão Lógica**: essa visão permite criar abstrações do sistema ou componente através de classes de objetos facilitando o design da aplicação. Se necessário fazer uso de um diagrama de entidade e relacionamento auxiliando na forma de entender como irá funcionar o sistema;
- **Visão de Processo**: essa visão permite descrever os processos do sistema e pode-se utilizar diagramas de atividades com o propósito de avaliar os requisitos não funcionais do sistema, como desempenho e disponibilidade. Particularmente quando são ocorridos processos concorrentes;
- **Visão de Desenvolvimento**: auxilia a equipe de desenvolvimento na organização do sistema. É usada também na distribuição da implementação e na manutenção de código, reutilização de componentes, visualização das dependências, bibliotecas e outros; e

- **Visão Física**: mostra a exibição do hardware do sistema, seus computadores, periféricos e suas conexões. Podem-se usar diagramas de implantação com o objetivo de auxiliar a equipe responsável pela implantação do sistema nos respectivos ambientes de desenvolvimento, homologação e produção.

Outra literatura sobre visões arquiteturais pode ser encontrada em (HOFMEISTER et al., 1999) cujo conteúdo se assemelha ao trabalho citado anteriormente, porém acrescentando uma visão conceitual. Essa visão ajuda a descrever complementos estruturais, comportamentais e de execução da arquitetura. A Figura 4.1 é um exemplo de uma visão conceitual do sistema.

Figura 4.1 - Visão conceitual de um sistema de robô de controle de empacotamento.



Fonte: Adaptada de Sommerville (2012).

Em cada projeto, sistemas orientados a objetos, estruturados, banco de dados, serviços web, dentre outros, existe um modelo particular de estruturar sua arquitetura. Para (HERZUM; SIMS, 2000) as visões permitem separar diferentes aspectos objetivando o gerenciamento da complexidade. As quatro visões definidas são a técnica,

a aplicação, a gestão de projeto, e a funcional.

4.3 Padrões Arquiteturais

Desenvolver um sistema de software orientado a objetos traz inúmeras vantagens ao projeto, como reusabilidade, escalabilidade, possibilidade do desenvolvedor trabalhar em um nível mais elevado de abstração, dentre outros. Entretanto isso não significa que um projeto será sempre bem sucedido mesmo tendo seguido todo o processo de concepção através de usos de metodologias. A grande maioria dos projetos existentes são baseados em experiências anteriores de outros projetos similares com o uso da reutilização, assim como podemos ver em diversas áreas da indústria, seja automobilística, eletrônica, espacial, bem como na computação.

Assim, a reutilização tem interferência positiva no custo do projeto, pois o usuário não precisa desenvolver algo novo, as soluções são mais confiáveis e geralmente são mais fáceis de serem refatoradas ou substituídas.

Verifica-se na atualidade um intenso uso de padrões como uma forma de apresentar, compartilhar e reutilizar o conhecimento sobre sistemas de software. Neste sentido, deve-se pensar em um padrão de arquitetura como uma descrição abstrata, estilizada, de boas práticas experimentadas e testadas em diferentes sistemas e ambientes. Portanto, um padrão de arquitetura deve descrever uma organização de sistema bem sucedido em sistemas anteriores, contendo informações sobre adequação, vantagens e também suas desvantagens (SOMMERVILLE, 2012).

Algumas definições sobre padrões podem ser encontradas na literatura:

“Padrões não são novas soluções, mas soluções que foram implementadas com sucesso de forma recorrente em diferentes contextos. A partir deles, é possível aprender com a experiência de outros desenvolvedores e absorver esse conhecimento que eles levaram diversos anos para consolidar“ (GUERRA, 2014).

“A melhor maneira de usar os padrões é carregar o seu cérebro com eles e depois reconhecer locais em seus designs e aplicativos existentes onde eles possam ser aplicados. Em vez da reutilização de código, com os padrões obtém-se a reutilização de experiência“ (FREEMAN et al., 2005).

“Cada padrão descreve um problema que ocorre repetidamente em um ambiente e, por conseguinte, descreve uma solução para esse problema de tal forma que se pode utilizar a solução milhares de vezes sem nunca fazê-lo da mesma maneira duas

vezes“(ALEXANDER, 1977).

Para (GAMMA, 1995) o termo “*pattern*“ (padrão) possui quatro elementos fundamentais:

- **Nome do padrão**: um bom nome de padrão ajuda na identificação e apresentação de um problema. Ainda facilita a comunicação entre equipes de projeto, documentação e na utilização modelos abstratos;
- **Problema**: a identificação de quando utilizar um padrão se dá quando um problema é encontrado;
- **Solução**: a solução não é baseada em um problema específico onde irá encaixar perfeitamente como uma solução. Ela é baseada em diversas experiências e exposta através de soluções abstratas para um problema de projeto; e
- **Consequência**: são os resultados da aplicação do padrão à necessidade encontrada. A avaliação em relação ao padrão utilizado é muito importante, pois um padrão pode influenciar em requisitos como flexibilidade, extensibilidade ou portabilidade de um sistema.

Para exemplificar de forma sucinta o uso de padrões arquiteturais, as seções seguintes serão destinadas a exemplos de padrões e mais informações podem ser vistas em (BUSCHMANN et al., 2007).

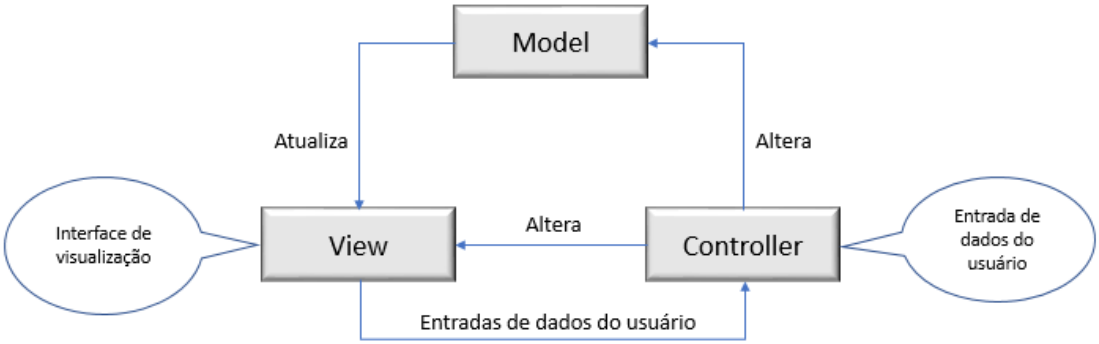
4.3.1 Model-View-Controller (MCV)

O padrão MVC diminui a complexidade no desenvolvimento dos sistemas de software através da separação de interesses, desacoplando a lógica de negócios da interface gráfica com o usuário. Esse padrão é o eixo do gerenciamento de interação em muitos sistemas baseados na internet e, além disso, atualmente muitos dos frameworks mais utilizados no mercado foram criados a partir desse modelo como ASP.NET, Spring MVC, Cake PHP e muito outros. Desta forma o aprendizado sobre este padrão torna-se essencial para quem quer desenvolver algum sistema utilizando estes frameworks.

O MVC consiste em três tipos de objetos a são brevemente descritos a seguir. **Model** é o coração da aplicação, nele pode conter regras de negócio, entidades, acesso aos dados e tudo que pertence à aplicação em termos de validação. O componente

View renderiza e define como a resposta será apresentada ao usuário, como gráficos, tabelas, diagramas, entre outros. O terceiro componente é o **Controller** ele é o que chamamos de coordenador do fluxo de dados do model para o view, obtendo as informações obtidas pelo model, processando e entrega pro View. A Figura 4.2 resume de forma eficiente como se dá esse processo de comunicação do padrão MVC.

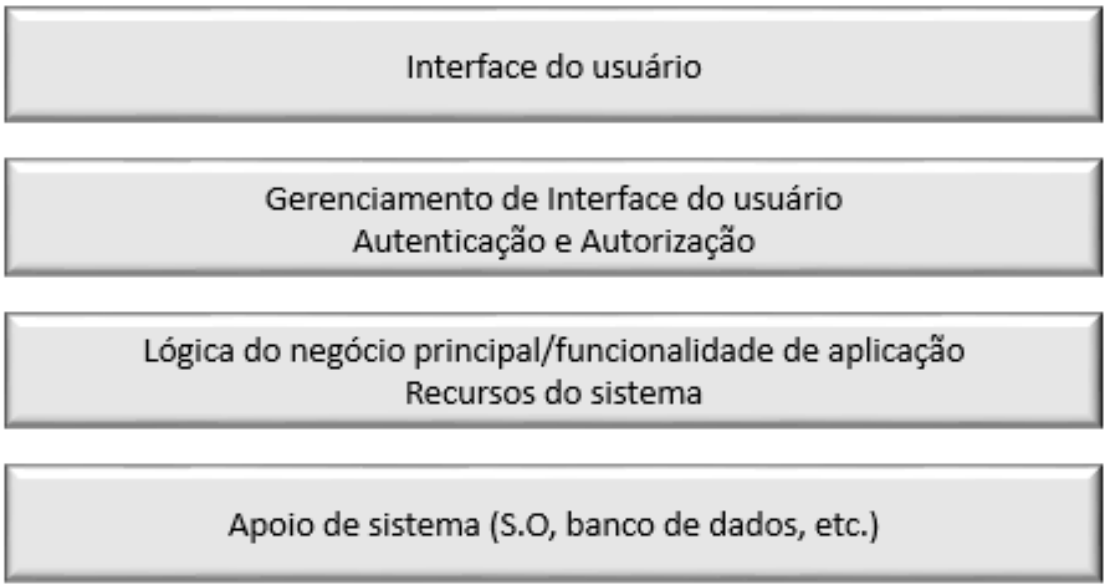
Figura 4.2 - Fluxo de comunicação do padrão MVC.



4.3.2 Arquitetura em camadas

Assim como o padrão MVC, a estrutura em camadas permite a separação e independência dos componentes da arquitetura e auxilia no processo de decisão, tornado os sistemas mais flexíveis e reutilizáveis. As camadas devem possuir diversas características, como responsabilidade bem definida; camadas inferiores com serviços gerais e as superiores, serviços mais específicos do sistema. A arquitetura em camadas pode ser utilizada quando novos requisitos do sistema precisam ser implementados e cada equipe de desenvolvimento pode ficar responsável por uma determinada camada. Uma desvantagem é que as camadas superiores dependem dos recursos e serviços disponibilizados pela camada abaixo.

Figura 4.3 - Exemplo de uma arquitetura genérica em camadas.



Fonte: Adaptada de Sommerville (2012)

4.3.3 Duto (*pipe*) e Filtro (*filter*)

É um padrão extensamente usado em sistemas operacionais UNIX onde o uso do caractere "|" faz a junção de dois comandos para produzir uma determinada saída. Segundo (GARLAN; SHAW, 1993) os processos são incrementais, por exemplo, quando um processo **A** gera alguns dados de saída, o processo **B** não precisa esperar a finalização do processo **A** para começar a receber esses dados. A função do componente Duto *pipe* é ler os fluxos de dados de entrada produzindo os resultados como fluxo de saída; já o conector *pipe* é responsável por direcionar o fluxo através de uma conexão com o fluxo de saída de um filtro.

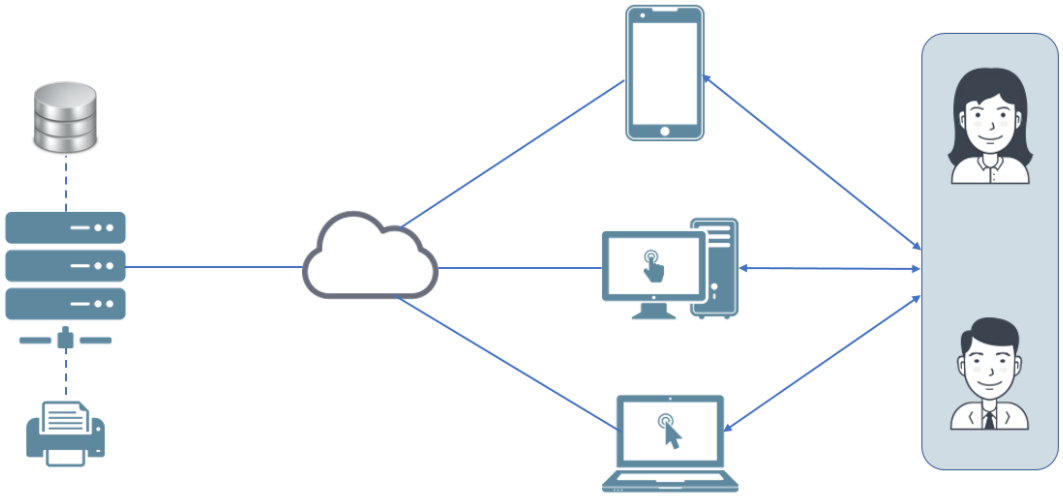
4.3.4 Repositório

O padrão repositório é utilizado quando há uma grande quantidade de dados envolvidos que precisam ser armazenados por um longo período e compartilhados em um repositório central de forma que os dados possam ser acessados por todos os componentes do sistema. Toda a comunicação ocorrida entre os componentes é realizada por intermédio do repositório.

4.3.5 Cliente-Servidor

A arquitetura cliente-servidor é um modelo muito utilizado para sistemas distribuídos onde os clientes acessam serviços disponibilizados pelos servidores através de protocolos de internet. Entre os recursos mais comuns disponíveis estão, os servidores de impressão e arquivos. Com a utilização deste tipo de arquitetura é possível obter facilidade de manutenção, pois a centralização dos serviços fornecidos está no servidor. É importante mencionar que os clientes podem ou não fazer uso dos serviços disponibilizados. Quando um computador cliente está com algum problema, ele pode ser substituído sem que os outros clientes sofram alterações. Os clientes podem ter tecnologias diferentes, como sistemas operacionais por exemplo. Os servidores e serviços também podem ser alterados sem afetar outras partes dos sistemas, desde que haja prudência na troca, como uma redundância do servidor a fim de garantir a disponibilidade dos serviços. A 4.4 demonstra um exemplo de uma arquitetura cliente-servidor.

Figura 4.4 - Fluxo de uma arquitetura cliente-servidor.



4.3.6 REST (A Representational State Transfer)

REST é um padrão arquitetural que foi desenvolvido durante uma tese de doutorado por um dos principais autores do protocolo HTTP (FIELDING, 2000). Entre os princípios fundamentais estão: recursos com múltiplas representações, identificadores (onde qualquer “coisa“ deve possuir um identificador global), padronização

dos métodos e comunicação sem estado (*stateless*), ou seja não guarda o cache das informações, deixando as requisições e respostas mais simples. O REST é a forma de representação de um recurso na internet, possibilitando que aplicações interajam com o servidor por meio dos métodos HTTP (ver 4.4). O REST expõe os recursos através de seus identificadores (URIs) e a manipulação desses recursos são feitas através de seus métodos. Seu padrão de arquitetura permite o desacoplamento dos componentes assim como o padrão MVC, onde um componente controlador pode ser um servidor de aplicação, a camada de modelo pode ser implementada em qualquer linguagem, e várias camadas de apresentação trocam informações com os controladores através do protocolo HTTP por uma linguagem estruturada, sendo que a mais utilizada é a JSON (*JavaScript Object Notation*).

4.4 Protocolos de Comunicação

Em um cenário tecnológico caracterizado pela presença de computadores, serviços, dispositivos e também pelo conceito de internet das coisas (onde há uma constante conexão e troca de informações), verifica-se a importância de uma espécie de linguagem que viabilize a comunicação entre dois ou mais dispositivos. Sabe-se que as partes conectadas nem sempre utilizam da mesma tecnologia e linguagem. Para realizar essa comunicação faz-se necessário um protocolo de rede que é um conjunto de regras e padrões utilizado para possibilitar a comunicação, definição de formatos de mensagens, ordens de mensagens enviadas e recebidas e ações tomadas entre "coisas" distintas através de uma rede.

A linguagem usada pelos membros de uma rede é chamada protocolo de comunicação da rede. Protocolos facilitam o entendimento na comunicação ao criar uma linguagem comum para os membros. De uma perspectiva geral um protocolo de comunicação de rede é um conjunto estabelecido ou aceito de procedimentos, regras ou especificações formais que governam um comportamento ou linguagem específicos (GALLO et al., 2003).

Os principais protocolos TCP/IP de comunicação, utilizados como objeto de estudo para este trabalho são:

- **HTTP (Hyper Text Transfer Protocol)**: é um protocolo que controla a comunicação baseado em requisições e respostas entre o cliente e o servidor. O endereço eletrônico para localizar algum recurso através da internet é chamado de *Uniform Resource Locator* (URL). É possível ver um exemplo de uma solicitação HTML de um cliente tentando acessar

um recurso. O cliente abre o navegador e digita uma URL, por exemplo: http://www.inpe.br; o navegador formata a solicitação e faz o envio ao servidor; em seguida o servidor encontra a página solicitada e pode-se ver textos, imagens, links ou outros serviços; o servidor formata a resposta e envia ao navegador; o navegador resgata o HTML e compila em formato visual para o cliente, conforme ilustrado na Figura 4.5.

Figura 4.5 - Comunicação entre cliente e servidor utilizando o protocolo HTTP.

Quando é acessada a internet o navegador está enviando requisições a todo instante para o servidor. As requisições são especificadas pelo que pode-se chamar de verbos ou métodos HTTP. Na versão 1.1 do protocolo HTTP tem-se a opção de nove verbos que podem ser utilizados e vistos com mais detalhes em https://tools.ietf.org/html/rfc2616#section-9.8:

– **GET**: por meio dessa requisição é retornada uma representação de um recurso que pode ser um tipo de arquivo html *(HyperText Markup Language)*, xml *(eXtensible Markup Language)*, json *(JavaScript Object Notation)*, dentre outros. O método GET é o mais comum e utilizado entre os métodos. Exemplo: um simples acesso a uma página web é um exemplo de um método GET;

- **POST**: utilizado muito por programadores para inserir algum recurso em uma base de dados ou sistema de arquivo. Os dados da requisição são enviados ao servidor no corpo da requisição e não na URI *(Uniform Resource Identifier)*. Assim os dados enviados ficam ocultos na hora do envio;
- **PUT**: envia um recurso para que seja armazenado na URI fornecida. Caso o recurso exista, o mesmo deve ser atualizado, caso contrário, um novo recurso é inserido;
- **DELETE**: exclui o recurso indicado na requisição. A especificação do recurso a ser excluído pode vir pela URI ou pelo corpo da requisição;
- **TRACE**: utilizado para realizar chamadas remotas. O TRACE soa de volta a requisição recebida para que o cliente verifique se houve alguma mudança e/ou adições feitas por servidores intermediários;
- **OPTIONS**: retorna os métodos HTTP suportados pelo servidor;
- **PATCH**: atualiza partes de um recurso;
- **CONNECT**: transforma a requisição de conexão para um túnel TCP/IP para o uso de um proxy; e
- **HEAD**: retorna metainformação a respeito dos cabeçalhos da resposta. Esse método é parecido com o GET, exceto que o servidor não retorna a mensagem no corpo da resposta.

Para cada requisição de método que o cliente faz, uma resposta é retornada juntamente com um respectivo status. O status serve como um diagnóstico da resposta em forma numérica. Cada valor corresponde a um tipo de informação que é devidamente identificada e compreendida pelo usuário. Maiores informações podem ser encontradas no site da *World Wide Web Consortium* (W3C), a partir de `https://www.w3.org/Protocols/HTTP/HTRESP.html`. As escalas numéricas correspondem a cinco valores que serão delimitados a seguir:

- **100**: esse status significa uma resposta temporária e apresenta apenas informações de cabeçalho;
- **200**: esse status corresponde às requisições que retornaram sucesso;
- **300**: esse status corresponde aos redirecionamentos e o cliente deve realizar medidas adicionais para finalizar a solicitação;

– **400**: esse status significa que houve erros por parte do cliente ao realizar uma solicitação ao servidor; e

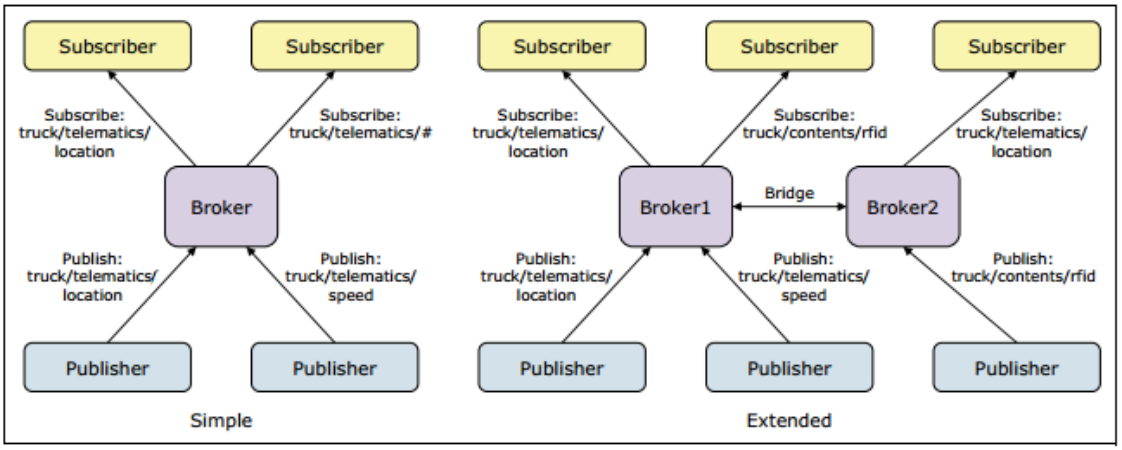
– **500**: esse status significa que houve algum erro por parte do servidor.

Verifica-se, ainda, a existência de outros protocolos de comunicação, como é o caso do protocolo *Message Queue Telemetry Transport* (MQTT). O MQTT é um protocolo que atua na mesma camada do modelo OSI que o HTTP. Entretanto o MQTT foi projetado para trabalhar com baixo consumo de banda de rede, e isso acaba sendo um diferencial quando a comunicação é realizada em dispositivos com baixos recursos de hardware e conexões de redes do tipo GSM (*Global System for Mobile*), por exemplo. Assim, muitas vezes o MQTT acaba sendo o protocolo mais utilizado para a comunicação da internet das coisas, pois proporciona a comunicação *machine-to-machine* (M2M).

Outra característica do MQTT é o paradigma *publish/subscribe (pub/sub)* para a realização de troca de mensagens. Para isso faz-se a necessário implementação de um componente intermediário (*middleware*) chamado de broker, cuja responsabilidade é receber, enfileirar e disparar as mensagens que foram recebidas dos publicadores para os assinantes. O envio de uma mensagem é realizado da seguinte maneira: o agente publicador realiza uma conexão no broker; em seguida o broker publica a mensagem; logo após o agente assinante recebe uma mensagem avisando que existe uma mensagem enviada a ele.

Uma característica interessante é que o assinante(s) pode filtrar as mensagens que ele quer receber. Exemplo: receber apenas informações de temperatura do satélite ou ainda informações da disponibilidade do link em uma rede. A utilização do broker permite o desacoplamento dos componentes, ou seja, o publicador e assinante podem estar em computadores e tecnologias distintas, algo não permitido em modelos de comunicação entre cliente e servidor. A Figura 4.6 mostra como é a configuração do paradigma pub/sub.

Figura 4.6 - Dois exemplos da configuração do *publish/subscriber*.



Fonte: Lampkin et al. (2012).

Os serviços web ou *web services* facilitam a comunicação distribuída entre aplicações que residem em múltiplas plataformas, através de diferentes modelos de objetos e linguagens. O *web service* é a tecnologia de integração preferida pelas organizações que implementam arquiteturas orientadas a serviços (ERL, 2004). Muitas empresas disponibilizam algumas de suas funcionalidades através dos *web services*, exemplos destes são: os correios ao disponibilizar uma API de serviço de busca de CEP, serviços governamentais como o Ministério da Fazenda que disponibiliza o serviço de nota fiscal eletrônica, dentre outros.

Simple Object Access Protocol (SOAP) é um protocolo de comunicação baseado em XML e sua comunicação é realizada através de mensagens entre aplicações via HTTP, normalmente utilizado em *web services*. Toda informação enviada pelo SOAP é através de um documento XML que contém elementos para facilitar a identificação da mensagem. Os elementos são:

- **Envelope**: identifica se o documento XML é uma mensagem SOAP;
- **Header**: são passadas nesse elemento todas as informações sobre o cabeçalho da requisição;
- **Body**: contém as informações passadas durante a requisição e resposta; e
- **Fault**: contém as informações de erros e status da mensagem;

A Figura 4.7 apresenta um modelo de um arquivo XML contendo uma mensagem

SOAP.

Figura 4.7 - Estrutura de um documento XML de mensagem SOAP.

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Fonte: https://www.w3schools.com/xml/xml_soap.asp.

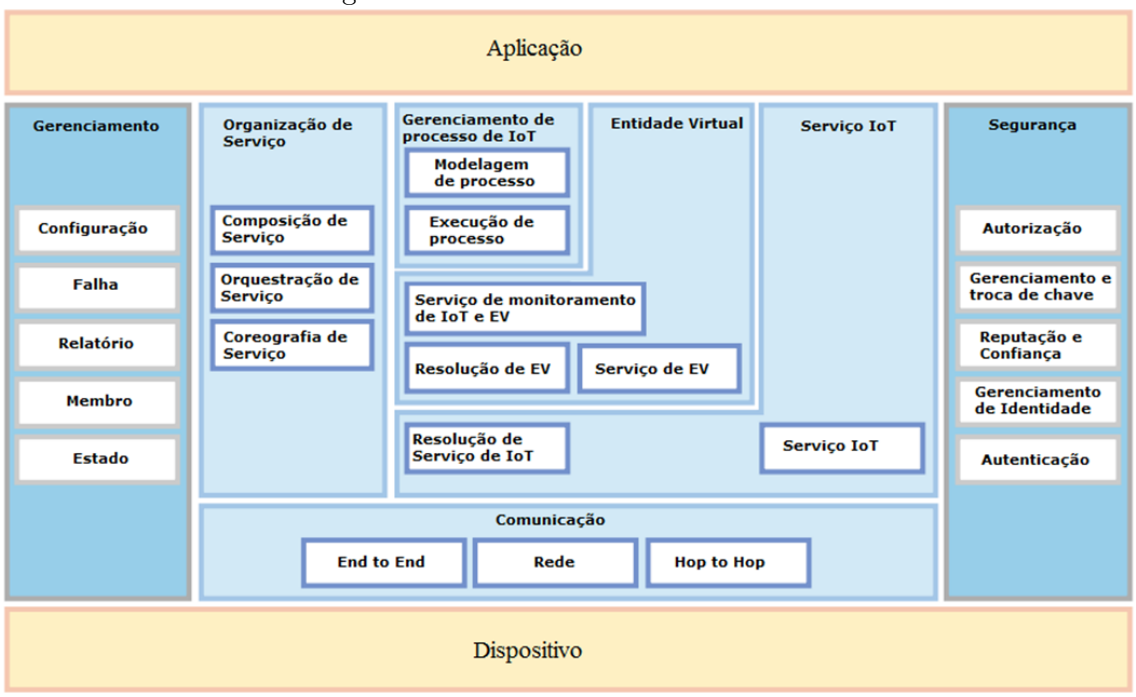
4.5 Sistemas IOT (Internet Of Things)

Um enfoque importante a ser dado diz respeito ao conceito de internet das coisas, onde objetos inteligentes possam colaborar de forma ativa com outros objetos físicos ou virtuais. Todavia, os ambientes relacionados à *Internet of Things* (IoT) possuem um alto grau de diferenças entre dispositivos e tipos de comunicação de rede. Para isso já existe no mercado diversas plataformas mediadoras (*Middleware*) no campo da computação distribuída que propõem promover a interoperabilidade entre os dispositivos e protocolos de rede. Mesmo assim, existe uma falta de padronização e as plataformas acabam sendo desenvolvidas de modo isolado, adotando tecnologias distintas umas das outras com diferentes modelos de programação.

A Figura 4.8 apresenta um diagrama de modelo de referência IoT representado na

linguagem UML.

Figura 4.8 - Modelo de Referência IoT.



Fonte: Bassi et al. (2013).

4.6 Arquitetura Adaptável

"Adaptação: Modificação de um sistema de software satisfazendo as novas necessidades e circunstâncias mutáveis" (TAYLOR et al., 2009).

"Software auto-adaptável modifica o seu comportamento em resposta a mudanças em seu ambiente operacional. Entendemos qualquer elemento observável pelo sistema de software, entradas do usuário, dispositivos de hardware externos e sensores ou programas de instrumentação" (OREIZY et al., 1999).

"Adaptabilidade é a habilidade de um sistema de software satisfazer novos requisitos e ajustá-los às novas condições operacionais durante seu tempo de operação" (TAYLOR et al., 2009).

Para (BARESI et al., 2006), os tipos de sistemas que preveem a aplicação de um mecanismo de adaptação são descritos como:

- **Sistemas fechados**
  - Conjunto de elementos definidos; e
  - A execução da adaptação é realizada apenas nos elementos definidos, garantindo a supervisão do sistema.
- **Sistemas Abertos**
  - Elementos podem aparecer e desaparecer; e
  - Os elementos existentes são "descobertos", atuados e supervisionados pelo sistema de adaptação.

Em (OREIZY et al., 1999) (BARESI; GUINEA, 2012) observa-se que para atingir os objetivos de adaptabilidade são necessários alguns processos para monitorar o ambiente, tais como analisar, planejar as mudanças e implantar as modificações em um sistema de software de modo a responder as seguintes questões: Sob quais condições? Adaptação de sistemas abertos ou fechados? Qual o custo efetivo? Qual o tipo de autonomia? Qual a frequência? Qual o tipo e precisão da informação? Esta percepção também é descrita em (GROUP et al., 2006) onde é possível categorizar atividades de autogerenciamento 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 a necessidade, criar um plano para atender a necessidade e executar esse plano.

Software autoadaptativo visa ajustar vários artefatos (componentes) ou atributos em resposta a mudanças no contexto (ambiente) ou no próprio sistema de software (SALEHIE; TAHVILDARI, 2009). Os itens a seguir apresentam as autopropriedades responsáveis pela autoadaptação de um sistema.

- **Autoconfigurável *(self-configuring)***: capacidade de reconfiguração automática e dinâmica em resposta às mudanças do ambiente operacional. Realizando tarefas como: instalação, atualização, integração, composição e decomposição de entidades de software;
- **Autorreparação *(self-healing)***: é a capacidade de descobrir, diagnosticar e reagir às perturbações no sistema. Ainda, antecipar a ocorrência de problemas e, consequentemente, tomar medidas adequadas para evitar uma falha;

- **Auto-otimização *(self-optimizing)***: também é chamado de auto-ajuste ou autorregulação, é a capacidade de gerenciar as características relacionadas ao desempenho como: tempo de resposta, vazão e utilização e carga, bem como alocação de recursos para satisfazer as necessidades dos diferentes usuários. Group et al. (2006) define auto-otimização como: "Habilidade do ambiente de TI maximizar eficientemente a alocação e utilização de recursos para satisfazer as necessidades do usuário com mínima intervenção. Auto-otimização endereça a complexidade da gestão do desempenho do sistema";

- **Autoproteção *(self-protecting)***: é a capacidade de detectar falhas na segurança e de recuperar seus efeitos. Tem dois aspectos, ou seja, defender o sistema contra ataques maliciosos, e antecipar problemas e tomar medidas para evitá-los ou para mitigar seus efeitos.

Ainda, para que o sistema seja monitorado e mantenha informação relevante para adaptação, é necessário o monitoramento de métricas de QoS. Os tipos de conhecimentos adotados são:

- **Autoconhecimento *(self-awareness)***: significa que o sistema tem conhecimento dos seus estados e comportamentos; e

- **Conhecimento do contexto *(context-awareness)***: significa que o sistema identifica o seu contexto, isto é, conhece dados do seu ambiente operacional.

Berns e Ghosh (2009) salienta que além das autopropriedades descritas anteriormente, apresentam-se mais cinco outras 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 de permanecer com estas configurações;

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

- **Autoescala *(Self-scaling)***: refere-se à capacidade de um sistema de manter um bom desempenho sob cargas (tamanhos) diferentes do sistema. Um sistema autoescalável 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, independente da ocorrência de falhas. Como resultado, o sistema incorre menos sobrecarga, quando falhas ou perturbação deixam de ocorrer e passa para o modo de mascaramento (auto-imune), quando falhas de qualquer tipo particular começam a ocorrer;

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

5 ARQUITETURA DO ARCTICFOXQoS

Neste capítulo são apresentados os elementos que compõem a arquitetura de referência do ArcticFoxQoS (AFQoS). Tais elementos são responsáveis pela criação de ambientes desenvolvidos para avaliar a qualidade de serviço em sistemas de segmento solo e de seus componentes. O AFQoS é integrante de uma arquitetura adaptável denominada ArcticFox proposta por (CEREJA JUNIOR, 2016a). Neste sentido, é dada ênfase a diversos aspectos, como: requisitos arquiteturais; contextualização do framework de adaptação ArcticFox; arquitetura de referência proposta do AFQoS; categorização das métricas coletadas; e formas de implantação para a realização da coleta de QoS.

5.1 Requisitos

Nesta seção são apresentados os requisitos arquiteturais e funcionais do AFQoS. Segundo (EELES, 2005) um requisito descreve uma condição ou capacidade de um sistema em conformidade diretamente com as necessidades do usuário e devem ser identificados em um contrato, norma, especificação ou outro documento.

Segundo (IBM, 2009) um bom conjunto de requisitos envolve interação humana e uma abordagem organizada. Os requisitos são a linha de comunicação vital entre usuários e desenvolvedores. A engenharia de sistemas é uma questão humana, e funciona apenas quando todos os envolvidos cooperam para resolver os problemas.

Para (SOMMERVILLE, 2012) os requisitos não são independentes e muitas vezes geram ou restringem outros requisitos. Contudo, as características e serviços muitas vezes não condizem apenas aos requisitos de sistema, mas também a respeito das funcionalidades necessárias para garantir a entrega dos serviços de forma satisfatória.

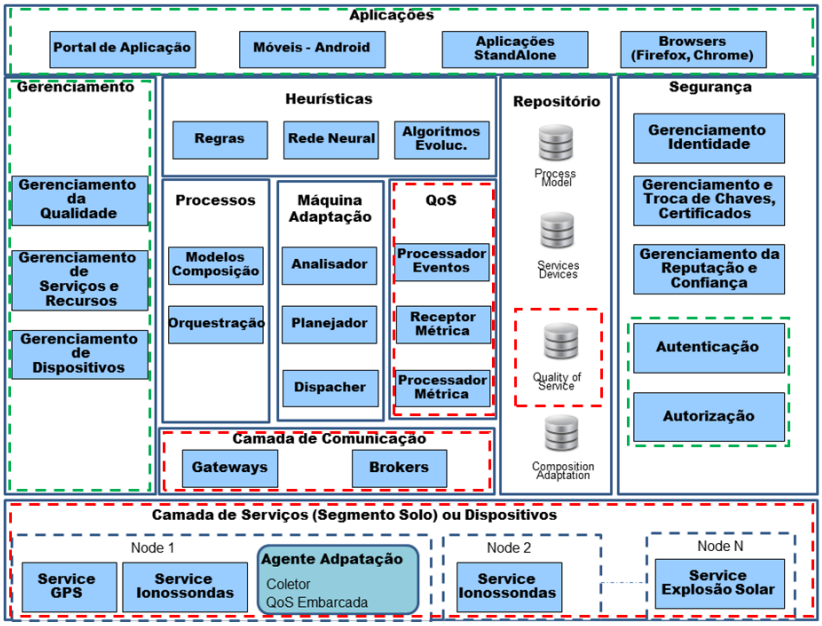
A partir da representação da arquitetura de referência em concordância com os requisitos especificados, será possível a modelagem de interfaces, componentes e sistemas, como também a construção completa de ambientes cujo o objetivo é a avaliação de QoS de qualquer dispositivo/serviço e/ou componente. Os requisitos arquiteturais e os requisitos funcionais do ArcticFoxQoS podem ser vistos no apêndice A.1.

5.2 ArcticFox

Sistemas de software têm se tornado cada vez mais complexos e frequentemente surge a necessidade de adaptação do mesmo devido a um *bugfix* ou a um novo requisito. De maneira geral estas adaptações são realizadas de forma manual. Para

manter os sistemas em operação e entregar os resultados esperados, surgem questões desafiadoras: Como medir a qualidade do serviço prestado pelo sistema? Como ajustar o sistema de modo a continuar operando em caso de indisponibilidade de algum componente? Segundo (CEREJA JUNIOR, 2016a), sistemas autoadaptáveis se apresentam como uma solução para adaptar o sistema face às alterações em seus ambientes operacionais. Neste contexto, é apresentado o framework ArcticFox para adaptação dinâmica de sistemas. O ArcticFox provê suporte autoadaptativo para sistemas de software que contempla: monitoramento de métricas de qualidade so sistema, análise, planejamento de alterações e execução da adaptação no sistema de software. A Figura 5.1 apresenta um modelo de referência arquitetural do ArcticFox.

Figura 5.1 - Modelo Arquitetural do ArcticFox.



Fonte: CEREJA JUNIOR (2016b)

As camadas pontilhadas em vermelho são integrantes do AFQoS e as camadas pontilhadas em verde são reutilizadas pelo AFQoS. Todas elas serão melhores detalhadas nas seções seguintes.

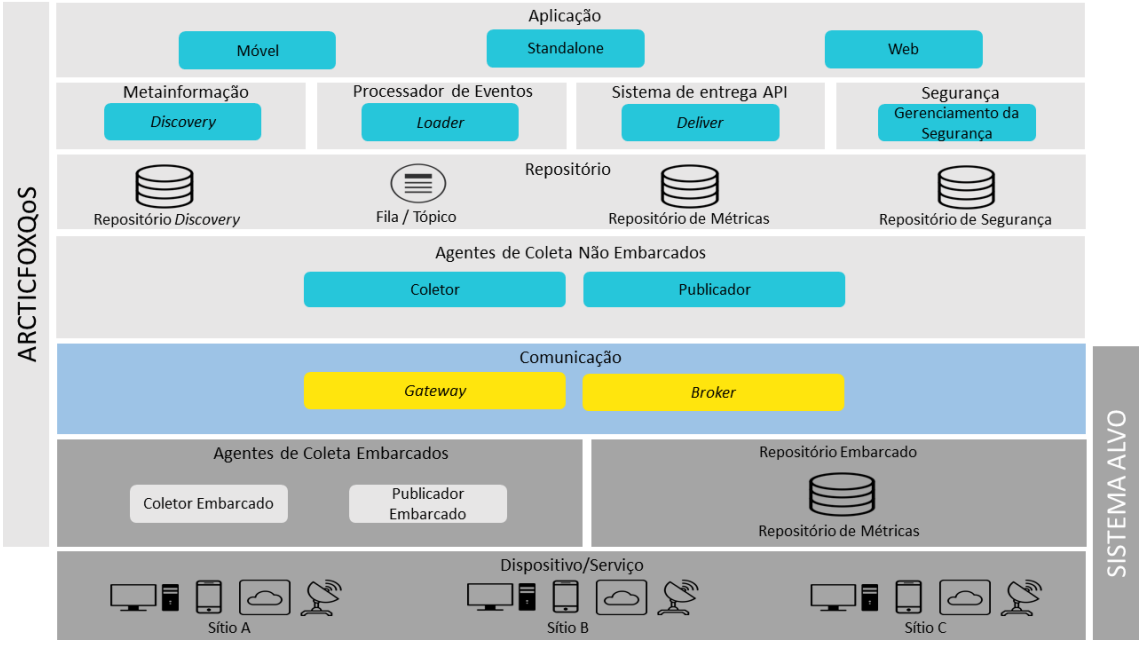
5.3 ArcticFoxQoS

O ArcticFoxQoS (AFQoS) é um modelo arquitetural em camadas que permite a implementação de seus componentes de forma desacoplada e independentemente de

tecnologia. Assim, a partir dela é possível produzir sistemas capazes de coletar, processar, armazenar e disponibilizar dados de qualidade de serviço para o framework citado acima. Entretanto a arquitetura de referência do AFQoS permite ser implementada de forma separada, fornecendo um padrão arquitetural para possibilitar a avaliação da QoS de qualquer dispositivo/serviço e/ou componente sem a necessidade da implantação de todo o sistema de autoadaptação.

As camadas que compõem a arquitetura proposta e suas respectivas interações estão descritas nas próximas seções. A Figura 5.2 apresenta as camadas e os componentes arquiteturais do AFQoS.

Figura 5.2 - Modelo Arquitetural do ArcticFoxQoS.



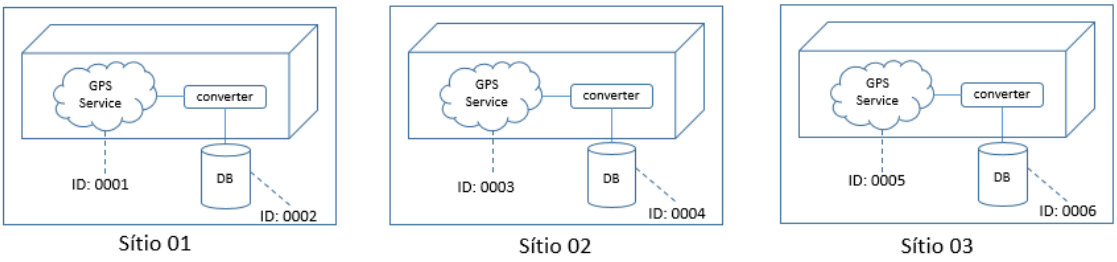
5.3.1 Camadas da Arquitetura

5.3.1.1 Sistema Alvo

- **Dispositivo/Serviço**: É a camada mais baixa da arquitetura proposta. Nela é possível identificar todos os serviços (dispositivos lógicos e físicos, sistemas computacionais, componentes de hardware e software, *web services*, entre outros) que serão monitorados pelo AFQoS. A fim de agrupar todos os sistemas alvo que estão sob o domínio de monitoramento do

AFQoS a camada de Dispositivo/Serviço pode ser subdividida por grupos chamados **Sítio** [1] **<identificador>**. Cada dispositivo deve possuir um identificador único e pertencer a um Sítio. Um exemplo pode ser aplicado onde dois ou mais sistemas alvo replicados ou redundantes encontram-se geograficamente distantes. A Figura 5.3 a seguir ilustra a identificação dos sítios e seus respectivos sistemas alvo.

Figura 5.3 - Número de identificação dos serviços.



- **Agentes de Coleta Embarcados**: Essa camada é responsável por toda a coleta de métricas embarcadas no sistema alvo. A responsabilidade do fornecimento das métricas é do AFQoS. Para que isso ocorra é necessário a criação de um ou mais agentes do tipo:
  - **Coletor**: O agente coletor pode ser um código fonte embarcado na aplicação do cliente, *web service*, aplicação standalone, entre outros. O mesmo deve ser fornecido pelo AFQoS onde será embarcado dentro do sistema alvo a fim de coletar, calcular e processar as métricas de QoS. Outra forma de coleta embarcada seria utilizando um software *analytics* de terceiros implantado dentro do sistema alvo cujo a função é extrair métricas de qualidade de serviço. Após a coleta, os dados são armazenados no repositório embarcado e enviados à fila de mensagens da camada **Repositório** do AFQoS localizada na nuvem que será explicada mais à frente.
  - **Publicador**: O agente publicador é uma outra alternativa de coleta de métricas embarcadas. Ele também pode ser um código fonte criado pelo AFQoS ou um software *analytics* de terceiros implantado dentro do sistema alvo. Após a coleta, armazena os dados no repositório

[1] conjunto de dispositivos/serviços dos sistemas alvo

central. Entretanto o envio para o **Repositório** do AFQoS é através do padrão de troca de mensagens *publish/subscriber*.

- **Repositório Embarcado**: Responsável por armazenar todas as métricas coletadas pelo agentes embarcados nos sistemas alvo. A finalidade deste repositório é obter uma cópia das métricas coletadas e disponibilizá-las caso haja a necessidade da implantação de serviços autônomos para adaptação de sistemas dentro dos sítios. Objetiva-se que os serviços monitorados estejam em lugares remotos e seus recursos de rede limitados.

- **Comunicação**

  - **Gateway e Broker**: Visa facilitar a conexão entre a camada de dispositivo/serviço e os agentes responsáveis pela coleta. Quando na camada de dispositivo/serviço o sistema alvo é um dispositivo físico, para a realização da comunicação com o mesmo é necessário a obtenção de um *driver* de comunicação onde na maioria das vezes é disponibilizado pelo próprio fabricante do sistema alvo. Uma vez que a conexão entre os agentes de coleta e o sistema alvo é habilitada pelos *drivers* (para dispositivos físicos) os dados podem ser enviados utilizando requisições HTTP. Quando o sistema alvo é um sistema de software web, *web service* ou uma aplicação standalone, não é necessária a utilização dos *drivers*, mesmo havendo tecnologias distintas por ambas as partes. Um dos principais exemplos de *gateways* que podem ser desenvolvidos são: estilo arquitetural REST e SOAP baseados em padrões Web. Dessa forma a utilização do protocolo HTTP não se limita apenas em realizar a comunicação, mas fornecer subsídios a todas as interações com os serviços interconectados.

    Outro protocolo importante que pode ser utilizado para realizar a interface de comunicação com a camada de serviço é o MQTT (*Message Queuing Telemetry Transport).* O mesmo utiliza o paradigma *publish/subscribe* (pub/sub) para a troca de mensagens. O modelo implementa um *middleware* chamado de broker responsável pela mediação entre os serviços.

    A realização das coletas pode ser realizada dentro ou fora do sítio monitorado e mais detalhes podem ser vistos na seção 5.5. A camada de comunicação foi projetada na arquitetura de forma a permitir a criação do *gateway* e *broker* em ambos os lados.

5.3.1.2 ArcticFoxQoS

- **Agentes de Coleta Não Embarcados**: Como o próprio nome da camada diz **Não Embarcado**, o agente publicador é instalado dentro ArcticFox-Cloud que está localizado em um servidor fora do sítio onde está localizado o(s) sistema(s) alvo.

  - **Coletor**: Basicamente o agente coletor acessa o sistema alvo, calcula as métricas, processa e envia à fila de mensagens da camada **Repositório** do AFQoS através do protocolo HTTP.
  - **Publicador**: O agente publicador possui características semelhantes ao agente coletor descrito no tópico acima, diferenciando-se apenas pela utilização do protocolo MQTT cujo o padrão de padrão de troca de mensagens é o *publish/subscriber*.

- **Metainformação**: Essa camada é responsável por armazenar no repositório *Discovery* todas as informações referentes às métricas de qualidade de serviço coletadas dos sistemas alvo. Por conseguinte acrescentando informações com o objetivo de identificar a origem do dado coletado.

  - **Discovery**: O conceito de *Discovery* é descobrir de forma autônoma ou não os novos dispositivos/serviços que serão monitorados pelo AFQoS. Outra característica dessa camada, é reorganizar os componentes que já fazem parte da arquitetura e do conhecimento do AFQoS, pois os mesmos podem sofrer ao longo do tempo diversos tipos de alterações como, acréscimo de um ou mais nós, indisponibilidade de um serviço ou alto tempo de resposta. Dessa forma, faz-se necessário o *Discovery* pra manter a base de dados consistente com a sua arquitetura atual. A descoberta pode ocorrer de duas formas: um sistema autônomo de adaptação ou um engenheiro responsável pelo planejamento e acompanhamento da qualidade, registrando cada componente novo que entra na composição dos serviços monitorados pelo AFQoS. A camada de metainformação é dividida em três subcamadas: Gerenciamento da Qualidade, Gerenciamento de Recursos e Serviços e Gerenciamento de Dispositivos, as quais serão detalhadas a seguir.

    * **Gerenciamento da Qualidade**: Esse componente da arquitetura foi baseado no documento de especificação (BAKALOS et al.,

2015) que visa identificar indicadores de qualidade para o desenvolvimento de um acordo de nível de serviço (*Service Level Agreement (SLA)*). A ideia é fornecer subsídios para a montagem de um SLA que seja legível por máquina (ou seja, um esquema JSON ou XML), onde será possível apresentar os dados de qualidade de serviço entre os sistemas alvo monitorados. As seguintes entidades fazem parte do gerenciamento da qualidade:

- **Quality Attribute**: A modelagem deste parâmetro foi baseado no trabalho de (CHOI et al., 2007), onde cada métrica é derivada de um atributo de qualidade. Ex: Atributo de qualidade: *Performance*, métrica: *Service Response Time*.
- **Metric**: A métrica é a entidade principal que corresponde à finalidade da coleta dos dados de qualidade de serviço. Nela deve conter informações necessárias a fim de auxiliar no entendimento do valor coletado, como unidade de medida, escala e atributo de qualidade. O exemplo na Tabela 5.1 mostra a definição de uma métrica cadastrada no AFQoS.
- **Unit**: Essa entidade permite registrar a unidade de medida da respectiva métrica.

Tabela 5.1 - Definição da métrica

Atributo	Performance
Métrica	Conexões Simultâneas
Escala	Proporção
Unidade	Número
Nota	Máximo de conexões simultâneas

- **Frequency**: Valores relacionados à frequência de coleta. Exemplo: a cada 2h, semanalmente, 1 vez por mês, entre outros.
- **Parameter**: Os parâmetros auxiliam na identificação da tipagem (*float, integer, double*) dos valores recebidos em cada métrica explicitando suas particularidades e o que se deve esperar de cada métrica.
- **Dependency**: Algumas métricas para se chegar ao seu valor final devem ser calculadas baseadas em outras métricas.

Ou seja, possuem uma relação de interdependência. Exemplo: A disponibilidade indica que os serviços estão ativos e respondendo às requisições dos usuários.
Availability = totalErrors (total de erros por requisição)/simultaneousConnections (Número máximo de conexões simultâneas) (BAKALOS et al., 2015). Ou seja, a maior disponibilidade é alcançada com mais conexões simultâneas.

· **SLA**: Nessa entidade são gravadas todas as informações pertinentes à respeito do acordo de nível de serviço prestados pelos sistemas alvo. A partir dele é possível identificar se os requisitos estão sendo atendidos, bem como fornecer subsídios para o ArcticFox avaliar a QoS dos sistemas alvo monitorados. A Tabela 5.2 demonstra as informações básicas de um SLA.

Tabela 5.2 - Informações básicas do acordo de nível de serviço

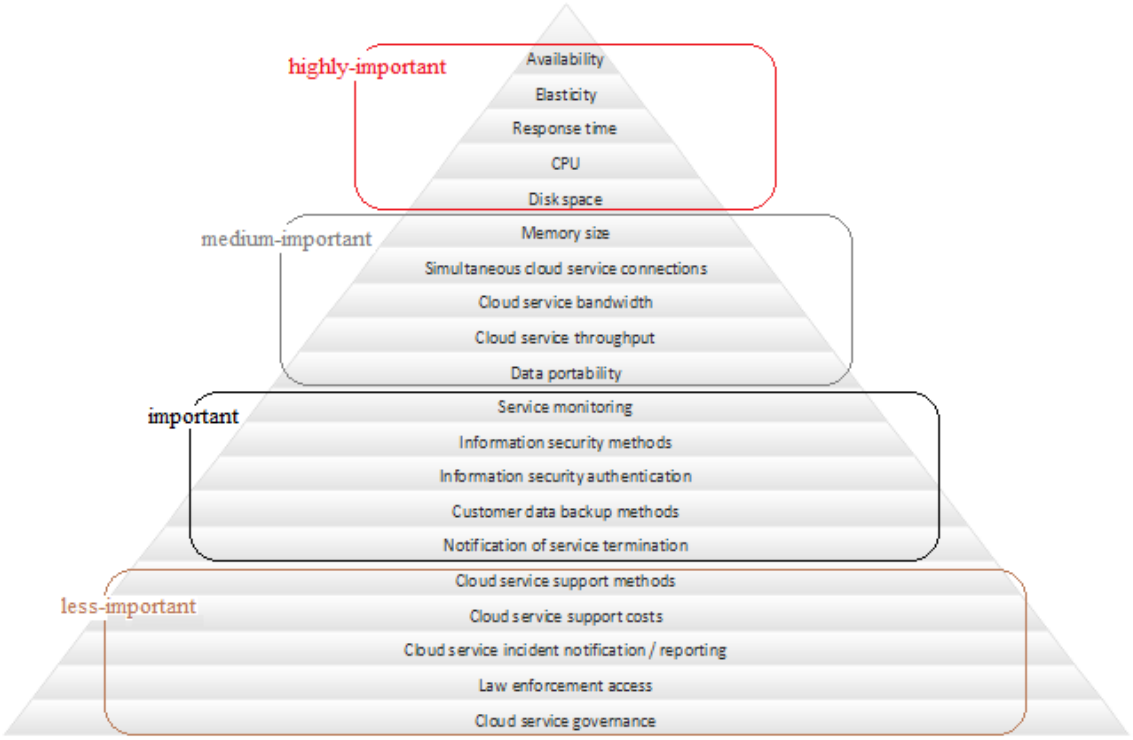
Nome do Serviço	Space Weather Service Level Agreement GNSS Application Service
Descrição	This Space Weather GNSS Application Service Service Level Agreement (SLA) is a policy governing the use of GNSS Web Services
Data de Criação	dd/MM/yyyy
Data de Atualização	dd/MM/yyyy
Data de Expiração	dd/MM/yyyy

· **Service Contract**: A partir dessa entidade é possível estabelecer um relacionamento entre *Architecture Element* e um *Resource*.

· **Importance Level** - São atributos que ajudam a definir graus de prioridade em dois níveis: SLA e Métricas. SLA: quando dois ou mais serviços monitorados pelo AFQoS possuem métricas coletadas nas quais não violaram as regras definidas no contrato de serviço. Pode-se definir um grau de importância para cada serviço. Exemplo, o serviço A e o Serviço B estão com níveis aceitáveis de taxa de erro em requisições *Restful*. Assim, o framework ArcticFox de Adaptação irá decidir qual serviço será utilizado pelo grau de importância definido. Métrica: segundo o documento de especificação

(BAKALOS et al., 2015), as métricas de qualidade de serviço são difíceis de serem classificadas, devendo-se descobrir quais são mais ou menos importantes, e como elas podem mudar de lugar sem afetar a eficiência dos outros. Dessa forma o documento separa em quatro (4) grupos: Altamente Importante, Médio Importante, Importante e Menos Importante, conforme ilustrado na Figura 5.4. Entretanto esse documento foi escrito e especificado para SLA´s Cloud e essa abordagem não deve ser utilizada em todos os casos.

Figura 5.4 - Níveis de importância das métricas.



Fonte: Bakalos et al. (2015)

- **Event Conditional** - São as restrições definidas para cada métrica e o AFQoS às utiliza para indicar se o valor coletado está dentro dos parâmetros de QoS aceitáveis, bem como suas consequências de violação. Como exemplo pode-se utilizar um sistema de *download* de imagens de satélite na qual deve possuir uma disponibilidade de 99,9% de seus serviços. Caso isso não ocorra, a consequência de violação deste serviço

seria enviar um e-mail para o engenheiro responsável para tomar as medidas cabíveis, ou poderia utilizar como parâmetro para o ArcticFox substituir o serviço atual por outro serviço redundante ou replicado com melhores níveis de QoS.

- **Status**: Essa entidade permite alterar a condição de um SLA ou Métrica. Exemplo: O Engenheiro de acompanhamento da qualidade decide por alguma razão que a métrica que calcula a taxa de bits trafegada na rede não é mais necessária. Então, é alterado o status dessa métrica para que não ocorra mais a coleta. O mesmo caso pode ser aplicado ao SLA. A Tabela 5.3 apresenta alguns tipos de status de um SLA e/ou Métrica.

Tabela 5.3 - Informações sobre os possíveis status dos SLA´s, métricas e contratos de serviços.

Ativado	Pleno funcionamento
Desativado	Temporariamente fora de funcionamento podendo voltar ou não
Pausado	Parado esperando retormar o status de ativado
Expirado	Prazo de validade vencido
Finalizado	Contrato encerrado
Cancelado	Contrato cancelado por ambas as partes

* **Gerenciamento de Recursos e Serviços**

  - **Resources** - São alguns códigos fonte que fornecem alguma funcionalidade, coletando ou atuando sobre os dispositivos. Podem ser de dois tipos: *OnDeviceResource*, onde o dispositivo é responsável pelo fornecimento do dado. Eles podem disponibilizar em tempo real e armazenar os dados coletados de um sensor de temperatura, por exemplo. Porém podem ser limitados pelos seus recursos físicos. O segundo é o *Network Resource*, onde as informações são coletadas através da rede em um servidor ou em algum serviço na nuvem, dessa forma não são limitados aos recursos de hardware. Vale lembrar que os dados coletados dos sensores não são importantes, mas sim às suas características não funcionais, como o tempo de resposta, uptime do sensor ou do dispositivo, entre outros. Assim os metadados armazenados são referentes à forma de como foi coletado os dados de QoS, se foi embarcado ou pela

rede, UUID do dispositivo (*Universally unique identifier*) do dispositivo, entre outras informações. Os resources são responsáveis por hospedar elementos de software. A Tabela 5.4 apresenta uma instância de um Resource.

Tabela 5.4 - Exemplo de uma metainformação sobre um Resource.

Resource	Coletor embarcado dentro do servidor N01
ResourceType	Adaptation Agente, Collector, Driver
ResourceProperty	HTTP, http://teste.com, gtexQueueSJC
ResourcePropertyType	Protocolo, URL, Nome da Fila

- **Architecture Element**: Essa entidade do gerenciamento da qualidade possui informações sobre como acessar os *endpoints* de acesso aos recursos, disponibilizando metainformações, como porta de acesso, endereço host, protocolo de conexão e outros. Desse modo é possível abstrair a complexidade de acesso e possibilitar a comunicação dos mesmos com o mundo real. A 5.5 apresenta uma instância de uma Architecture Element.

Tabela 5.5 - Agrupamento das informações de um elemento arquitetural.

ArchitectureElement	GTex WS
ArchitectureElementType	Web Service
ArchitectureElementProperty	http://gtex.com, SOAP, REST
ArchitectureElementPropertyType	Endpoint, Protocolo, Estilo Arquitetural

* **Gerenciamento de Dispositivos**
  - **Device**: Hospeda um elemento arquitetural. Exemplo, um servidor de aplicação web cujo o serviço monitorado está implantado. Outros exemplos também se aplicam a um satélite, celular, sensor, máquina virtual ou seja, qualquer “coisa“ capaz de embarcar um software. A Tabela 5.6 apresenta uma instância de um Device.

Tabela 5.6 - Informações cadastrais de um dispositivo.

Device	Servidor HP 01, N01, N02
DeviceType	Servidor
DeviceProperty	Linux, 192.168.0.1, Debian 9, 3.2
DevicePropertyType	S.O, IP, Versão do S.O, Release

· **Hosting**: A entidade *Hosting* faz parte de um relacionamento auxiliar entre as Entidades *Architecture Element* (AE) e *Device*. Dessa forma é possível identificar em qual dispositivo o elemento está instalado. Exemplo: GTex WS (AE) está instalado no servidor N2 (Device).

· **Entity**: Nível mais alto de abstração contendo informações do(s) sistema(s) alvo. A partir dessa entidade, podem surgir diversas especializações. Exemplo: **Entidade:** *GNSS*

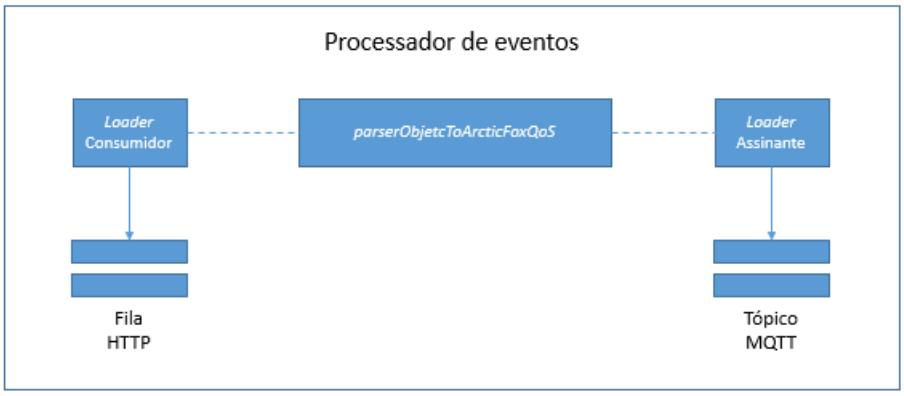
Tabela 5.7 - Entidade e suas especializações.

Entidade	GNSS
Architecture Element	GNSS GTex
Architecture Element	GNSS TecMap
Architecture Element	GNSS TecMap Estações
Architecture Element	GNSS TecMap Mobile

Obs.: Architecture Element, são outros exemplos de aplicações do GNSS.

- **Processador de Eventos**
  - ***Loader***: São agentes responsáveis em consumir as métricas que foram enfileiradas pelos agentes coletores na fila de mensagens e adicioná-las ao repositório de métricas. Os agentes *Loader* também são responsáveis em realizar um processamento no dado consumido da fila/tópico de mensagens. O propósito deste processo é formatar o dado em um objeto do AFQoS para ser persistido no repositório. Outra forma de implementar um agente processador de eventos pode ser feito utilizando o protocolo MQTT e o *Loader* passa a ser um agente assinante recebendo as mensagens mediadas pelo broker. A seguir, tem-se uma ilustração de como se dá este processo (Figura 5.5):

Figura 5.5 - Agentes *Loader* implementando os protocolos HTTP e MQTT.



- **Sistema de entrega API**: Essa camada é responsável em prover uma API *(Application Programming Interface)* que oferece uma interface de serviço com objetivo de entregar as metainformações de qualidade dos sistemas monitorados. O serviço oferece um conjunto de regras através de documentos dos dois principais idiomas padrões da web como JSON para serviços REST através do protocolo HTTP e WSDL (*Web Services Description Language*) para atender clientes que implementam o protocolo SOAP.

  - **Deliver**: É um *web service* responsável por disponibilizar informações a respeito dos sistemas monitorados. Tais informações vão desde o cadastro até valores das métricas coletadas. O serviço disponibiliza uma API de interface que permite ser acessada por qualquer cliente previamente autorizado e autenticado pelo sistema do AFQoS. Este é o único meio de acesso aos dados, impossibilitando o acesso direto ao repositório. Cada *web service* da API possui um documento de especificação descrevendo ao cliente como o serviço deverá ser acessado. As informações podem ser: tipo do parâmetro e como ele deve ser esperado pelo serviço, informações de cabeçalho, o tipo de retorno do serviço, entre outros. As figura 5.6 e 5.7 respectivamente mostram exemplos de dois tipos de contratos no formato JSON para serviços REST e WSDL para serviços SOAP.

Figura 5.6 - Documento de descrição de serviços para clientes REST.

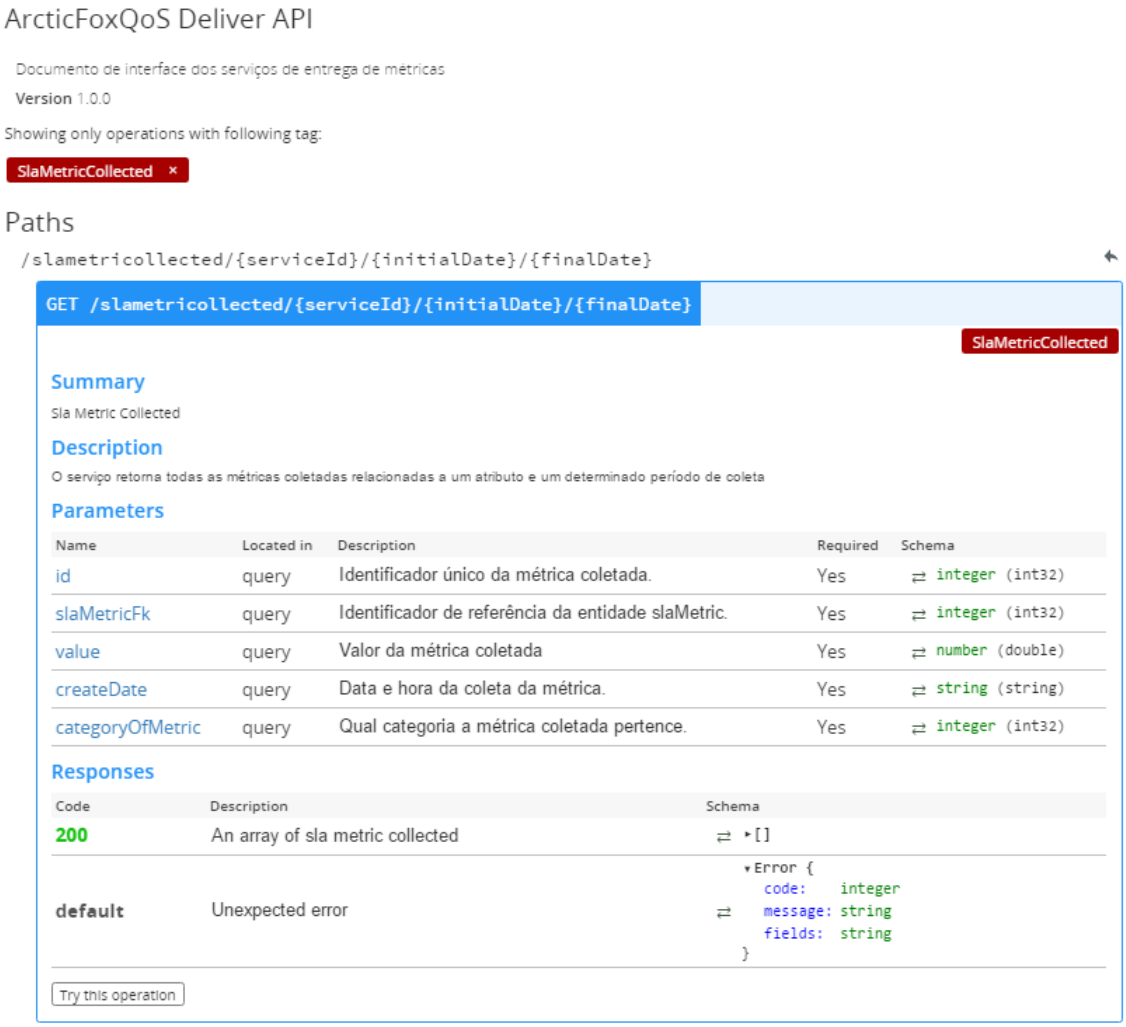


Figura 5.7 - Descrição em formato XML de um *Web Service* para clientes SOAP.

```
<?xml version="1.0" encoding="UTF-8"?>
- <definitions targetNamespace="br.com.arcticfox.qos.wsdl" name="O serviço retorna todas as métricas coletadas
relacionadas a um atributo e um determinado período de coleta" xmlns:xsd1="br.com.arcticfox.qos.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="br.com.arcticfox.qos.wsdl" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">
        <!-- definition of datatypes -->
  - <types>
      - <schema targetNamespace="br.com.arcticfox.qos.xsd" xmlns="http://www.w3.org/2000/10/XMLSchema">
          - <element name="id">
              - <complexType>
                  - <all>
                      <element name="value" type="int"/>
                    </all>
                </complexType>
            </element>
          - <element name="slaMetricFk">
              - <complexType>
                  - <all>
                      <element name="value" type="int"/>
                    </all>
                </complexType>
            </element>
          - <element name="value">
              - <complexType>
                  - <all>
                      <element name="value" type="float"/>
                    </all>
                </complexType>
            </element>
          - <element name="createDate">
              - <complexType>
                  - <all>
                      <element name="value" type="string"/>
                    </all>
                </complexType>
            </element>
          - <element name="categoryOfMetricFk">
              - <complexType>
                  - <all>
                      <element name="value" type="int"/>
                    </all>
                </complexType>
            </element>
        </schema>
    </types>
        <!-- response messages -->
        <!-- request messages -->
        <!-- server's services -->
    <portType name="buscar métricas coletadas por período e serviço"/>
        <!-- server encoding -->
  - <binding name="buscar métricas coletadas por período e serviço_webservices" type="tns:buscar métricas
    coletadas por período e serviço">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    </binding>
        <!-- access to service provider -->
  - <service name="deliver">
      - <port name="deliver_0" binding="buscar métricas coletadas por período e serviço_webservices">
            <soap:address location="/slametricollected/{serviceId}/{initialDate}/{finalDate}"/>
        </port>
    </service>
</definitions>
```

- **Segurança**

  - **Gerenciamento de Segurança**: Nesta camada é possível estender recursos de segurança como criptografia, infraestrutura de chaves públicas e privadas, comunicação segura, autenticação e controle de acesso, administração e gerenciamento de grupos e usuários do sistema, entre outros.

* **Autorização**: Qualquer recurso que o usuário queira acessar de forma segura e confiável pode ser um componente EJB, sistema web, *web service*, dentre outros. Ao acessar o AFQoS a primeira atividade que o sistema deve realizar é a autenticação. Dessa forma deve existir uma "Entidade" responsável por armazenar as identidades dos usuários como pode ser visto na Tabela 5.8.

Tabela 5.8 - Dados de acesso do usuário

User:	marcos
Pwd	1234

Ao acessar um serviço, a camada de segurança fará a autenticação verificando se as credenciais do usuário **marcos** se encontram no banco de dados de identidade.

* **Autenticação**: Garante que os usuários tenham permissões necessárias para realizar determinadas ações. Ao acessar uma página e a mesma estiver protegida, o sistema verifica se o usuário tem permissão de acesso à página solicitada. Exemplo: Usuário **marcos** do grupo **Gestão da Qualidade** possui permissão de leitura e escrita para acessar a página solicitada.

- **Repositório**: A camada de repositório centraliza todas as informações relacionadas ao sistemas alvo monitorados. A centralização de um repositório na nuvem, permite integrar as informações de forma padronizada, possibilitando a realização de avaliações e utilização dos resultados para analise da QoS.

  – **Repositório Discovery**: todas as metainformações sobre os dispositivos/serviços que estão sob o domínio de monitoramento do AFQoS são armazenadas neste repositório. Informações como serviços, recursos, dispositivos e informações de SLA´s (regras, consequências de violação, níveis de prioridade, atributos e métricas que serão objeto de medição e outros);

  – **Fila/Tópico**: informações dos serviços de mensagens podem armazenar qualquer tipo de informação que posteriormente será armazenada no repositório de métricas. Este repositório faz parte da comunicação entre os agentes de coleta e os processadores de eventos (*Loader*);

  - **Repositório de Métricas**: armazena os valores coletados pelos agentes embarcados ou não dos sistemas alvo monitorados;
  - **Repositório de Segurança**: são metainformações referentes ao gerenciamento de segurança, como controle de acesso, regras, gerenciamento de identidade e outros.

- **Aplicação**: Camada responsável pela interface com o usuário. Caso uma aplicação móvel que entrega o mesmo tipo de dado queira entrar no monitoramento do AFQoS, ela deve ser cadastrada pelo engenheiro de qualidade no sistema de metainformação.

Os relacionamentos entre as entidades descritas anteriormente nas quais são responsáveis pelo armazenamento das metainformações da QoS dos sistemas alvo monitorados pelo AFQoS, podem ser visualizadas no diagrama do Apêndice B.1.

5.4 Categorização das métricas

Como a coleta das métricas de QoS nos sistemas alvo podem ser realizadas por de diversas perspectivas, criou-se uma categorização das métricas. Dessa forma é possível realizar a identificação da forma de implantação do AFQoS durante a coleta das métricas, uma vez que em algumas situações o tipo de implantação pode impactar diretamente no valor da métrica coletada.

- **Métricas Embarcadas Naturalmente Existentes (MENE)** - São métricas providas prontamente pelo sistema alvo ao AFQoS. Ex: UPTIME. O cálculo da métrica é responsabilidade do sistema alvo.

- **Métricas Obtidas Não Embarcadas (MONE)** - São métricas coletadas pelo AFQoS sem a necessidade de embarcar código ou ferramenta para extrair do sistema alvo. Ex: Tempo de Resposta, PING. O cálculo da métrica é responsabilidade do AFQoS.

- **Métricas Embarcadas pelo AFQoS (MEA)** - São as métricas extraídas do sistema alvo, pode ser de três tipos:

  - **AFQoS (MEAC)** - Código fonte inserido dentro do sistema alvo ou através dos agentes coletores embarcados do AFQoS para calcular e obter a métrica do sistema alvo.

  - **Terceiros (MEAT)** - Um software *analytics* instalado dentro do nó é responsável por extrair as métricas do sistema alvo. Ex: Zabbix, Pulsar, Nagios, etc.

  - **Embarcada e Fora do Sítio ao mesmo tempo (MEAEFS)** - Um software analytics instalado fora do sítio extrai métricas do sistema alvo. Entretanto é necessário inserir um código fonte dentro do sistema alvo. Ex: Google Analytics, Clicky, Flurry Yahoo, etc.

- **Externo (EXT)** - São métricas obtidas dos sistemas alvo que não fazem parte do domínio de cobertura do AFQoS. Entretanto podem ser solicitadas para fins de comparação caso seja necessário. Dessa forma não existe a possibilidade de extrair uma métrica tipo **MEA**. As métricas do tipo Externo podem ser de dois tipos: **MENE** e **MONE**.

5.5 Formas de implantação

Existem duas formas de implantar a arquitetura do AFQoS para realizar a coleta, processamento, armazenamento e a distribuição das métricas de qualidade de serviços dos sistemas alvo. São elas:

- **Invasivas** - É quando dentro do sistema alvo é embutido um pedaço de código em alguma aplicação existente ou algum software ou ferramenta analítica com o propósito de obter as métricas de QoS do sistema alvo. Ou seja, o cliente que deseja ter seus sistemas avaliados, permitiu a implantação do AFQoS dentro do seu parque computacional; e

- **Não Invasivas** - Quando o sistema alvo não sofre alteração para a obtenção de métricas. A coleta é realizada fora do sítio do cliente.

5.5.1 Responsabilidade de Fornecimento X Formas de implantação

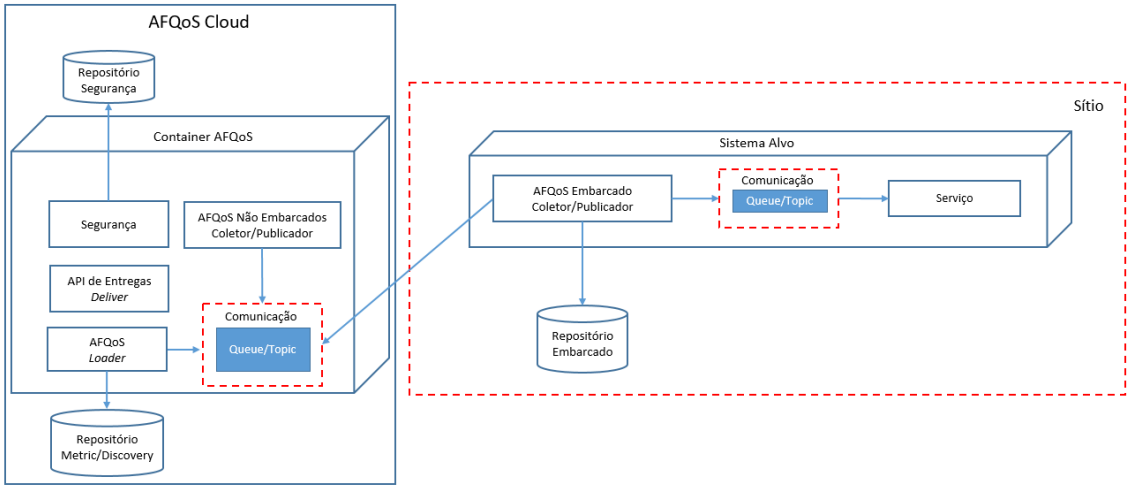
A Figura 5.8 apresenta de quem é a responsabilidade do fornecimento das métricas de qualidade de serviço e seus respectivos cenários de implantação.

Figura 5.8 - Responsável pelo fornecimento da métrica e as formas de implantação.

Responsabilidade de Fornecimento do dado de QoS	Não Invasivo	Invasivo
Sistema Alvo	MENE	----------
AFQoS	MONE	MEAC MEAT MEAEFN
Ambos	EXT	----------

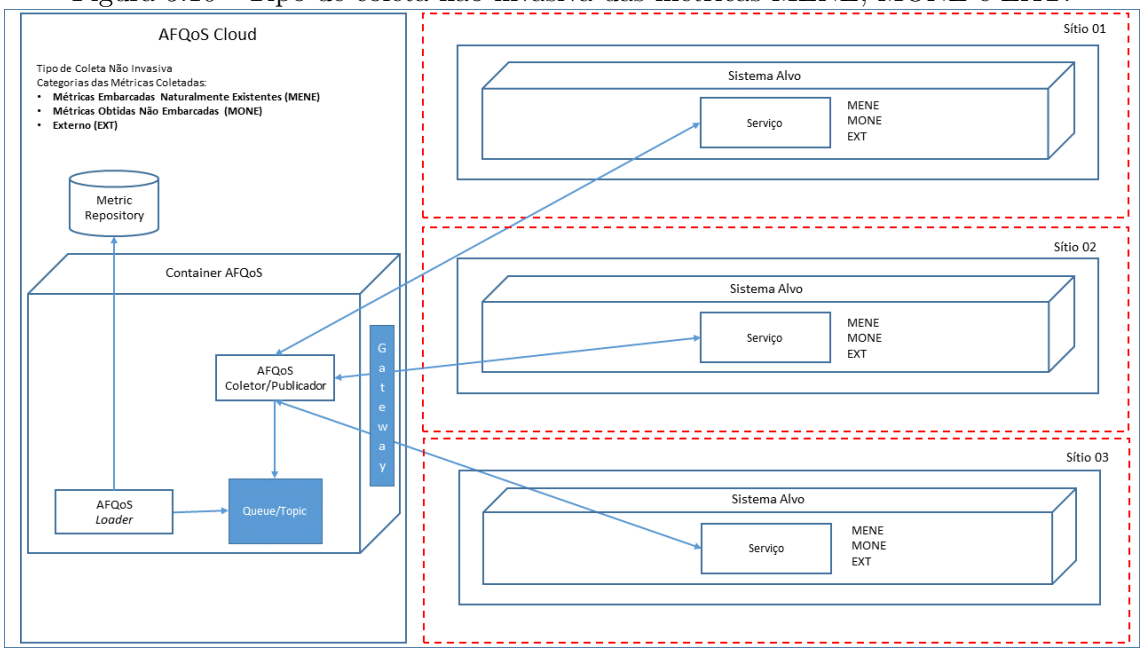
A Figura 5.9 apresenta um modelo conceitual de implantação geral de todos os componentes do AFQoS.

Figura 5.9 - Modelo de implantação geral dos componentes do AFQoS.



Os itens seguintes apresentam as formas de implantação do AFQoS para a realização da coleta, processamento, armazenamento e disponibilização dos dados de QoS. A Figura 5.10 demonstra uma redundância de três serviços, sendo um para cada sítio diferente. Em um servidor na nuvem está implantado o AFQoS completo, coletando os dados de fora dos sítios (não invasivo) e de métricas dos tipos MENE, MONE e EXT. Portanto, a comunicação entre os agentes coletores e os processadores de eventos se dão a partir da troca de mensagens utilizando filas ou tópicos. As setas representam os fluxos de dados trafegados entre os componentes.

Figura 5.10 - Tipo de coleta não invasiva das métricas MENE, MONE e EXT.



As Figuras 5.11 e 5.12 apresentam modelos cujos agentes coletores/publicadores são implantados dentro dos sistemas alvo de forma embarcada e um ou mais agentes coletores/publicadores dentro do sítio para cada serviço redundante. Outra característica dessa forma de implantação e a instalação de um repositório embarcado.

Figura 5.11 - Tipo de coleta invasiva das métricas MEAC usando coletor embarcado.

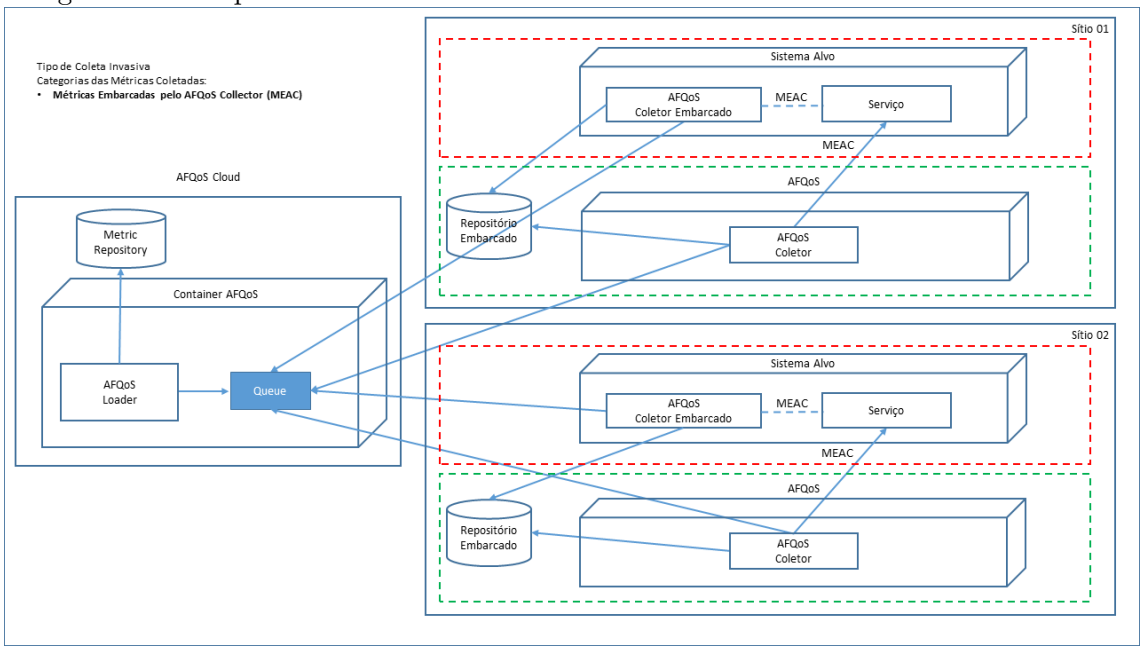
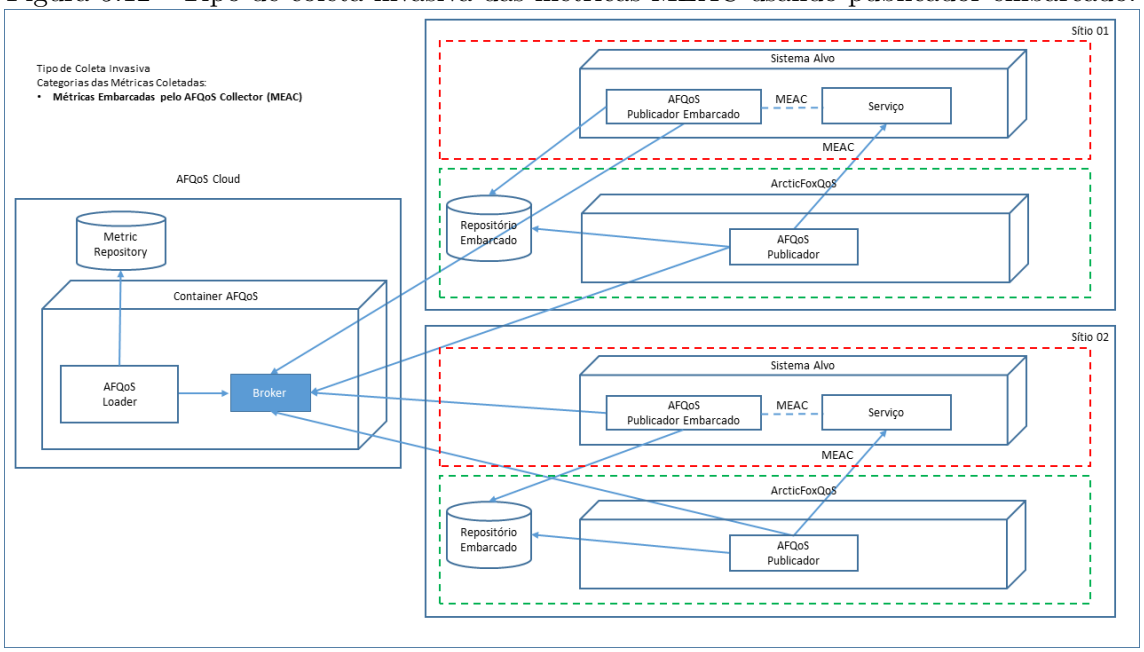


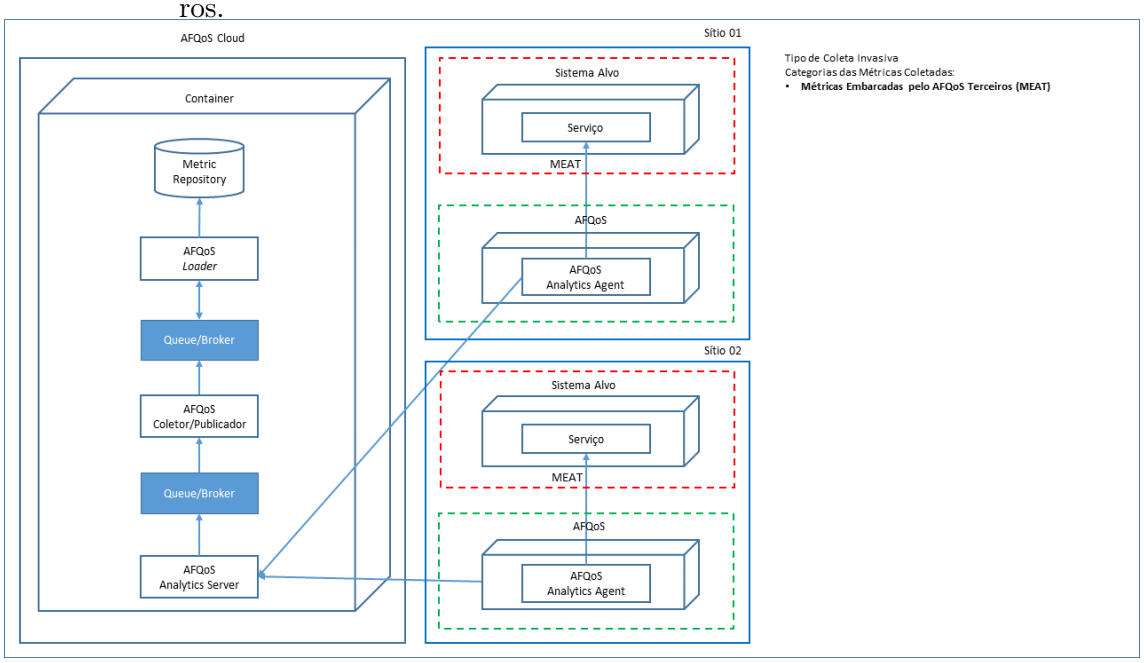
Figura 5.12 - Tipo de coleta invasiva das métricas MEAC usando publicador embarcado.



A Figura 5.13 expõe um modelo de coleta invasiva utilizando-se de ferramentas

analíticas para a realização do monitoramento dos sistemas alvo. A integração entre os agentes coletores de terceiros e os processadores de eventos, da-se pelo padrão de troca de mensagens.

Figura 5.13 - Tipo de coleta invasiva das métricas MEAT embarcadas pelo AFQoS terceiros.



As Figuras 5.14 e 5.15 apresentam um modelo de implantação para realizar coletas das métricas utilizando outra fonte de terceiros. Neste modelo é necessário a implantação de um código fonte embarcado dentro do sistema alvo necessário para a comunicação com a ferramenta de analise. A troca de dados entre a ferramenta e o AFQoS deve ser realizada pelo AFQoS Coletor/Publicador em seguida enviados à fila/tópico de mensagens.

Figura 5.14 - Tipo de coleta invasiva das métricas MEAEFS embarcada e fora do sítio ao mesmo tempo usando um coletor embarcado.

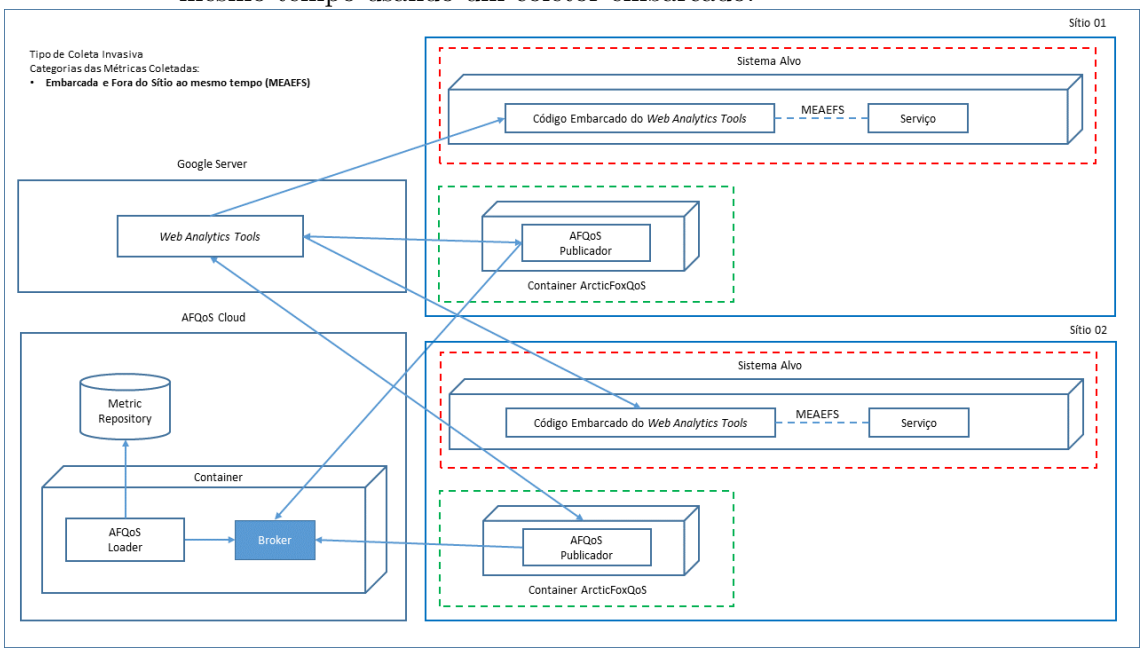
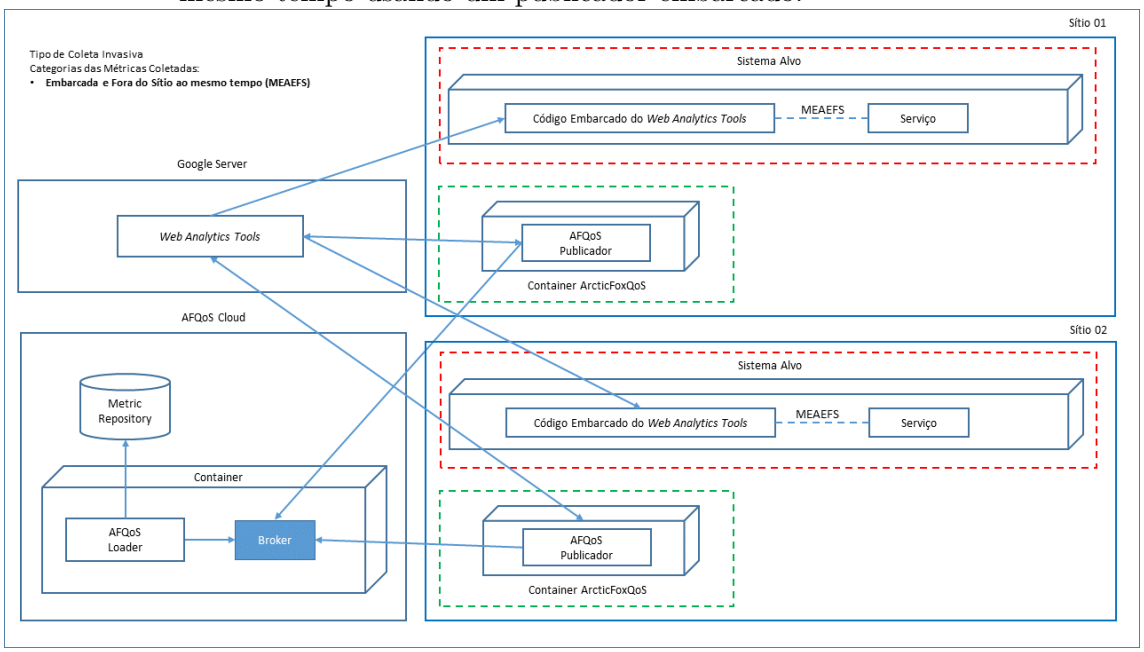


Figura 5.15 - Tipo de coleta invasiva das métricas MEAEFS embarcada e fora do sítio ao mesmo tempo usando um publicador embarcado.



Como foi apresentado nos parágrafos anteriores, existem diversas maneiras de implantação do AFQoS para realizar a avaliação da QoS dos sistemas alvo. Desde a implantação de coletores não invasivos onde pode haver um stress no serviço que está em produção à políticas de privacidade onde o cliente provedor permite a instalação de ferramentas de monitoramento de métricas de QoS. Vale lembrar que a decisão caberá ao engenheiro da qualidade responsável pela implantação de toda arquitetura computacional.

5.5.2 Engenheiro da Qualidade

Todas as metainformações existentes nos repositórios à respeito dos dispositivos, recursos, métricas, sla, estratégias de adaptação e outras são de responsabilidade do engenheiro da qualidade e podem ser subdivididas em:

- **Planejamento**: realizar o cadastro e planejamento dos serviços que serão monitorados, como: a escolhas das métricas que serão inicialmente coletadas, serviços, dispositivos que farão parte da composição dos sistemas alvo, definição dos acordos de níveis de serviços, quais ferramentas utilizar para a realização da coleta e as formas de implantação do AFQoS; e
- **Acompanhamento**: garantir que os requisitos do acordo de nível de serviço sejam atendidos pelo provedor dos serviços, realizando tarefas como monitoramento do AFQoS, verificação e avaliação desde a coleta, processamento, armazenamento e disponibilização dos dados de QoS coletados. Caso surja a necessidade da inclusão de um novo sistema alvo redundante, o engenheiro da qualidade responsável pelo acompanhamento deverá realizar todo o processo de registro desse novo sistema.

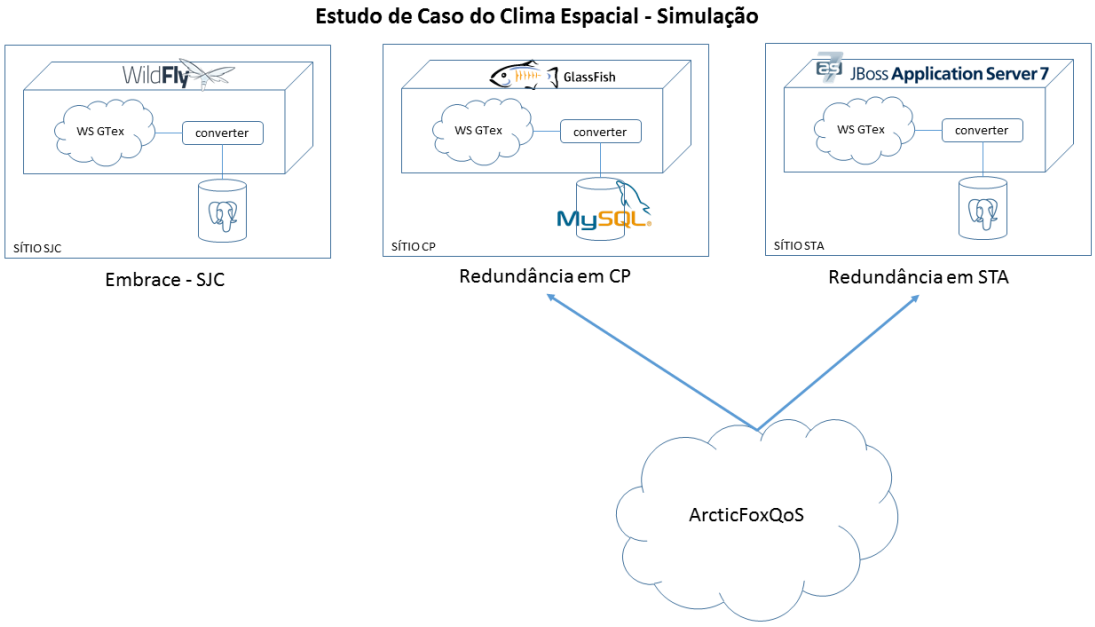
6 ESTUDO DE CASO

Este capítulo apresenta um estudo de caso para validar a arquitetura de referência proposta neste trabalho. O experimento simula um *web service* que disponibiliza dados de GNSS GTex, que é um serviço em operação no EMBRACE. Para realizar a simulação do cenário redundante do clima espacial foi necessário a criação dos seguintes itens:

- **Sítio de Cachoeira Paulista**
  - Criação de uma máquina virtual Ubuntu LTS 16 64-bit, 2048 MB;
  - *Web service* GTex implementado na linguagem JAVA;
  - Implementação Jersey API;
  - EclipseLink para o mapeamento dos atributos entre o banco de dados relacional MySQL e o modelo objeto da aplicação Java;
  - Banco de dados Mysql; e
  - Servidor Glassfish.
- **Sítio de Santa Maria**
  - Criação de uma máquina virtual Ubuntu LTS 16 64-bit, 2048 MB;
  - *Web service* GTex implementado na linguagem JAVA;
  - Implementação RestEasy API;
  - Hibernate para o mapeamento dos atributos entre o banco de dados relacional PostgreSQL e o modelo objeto da aplicação Java;
  - Banco de dados Popstgresql; e
  - Servidor JBoss AS 7.

A Figura 6.1 representa a estrutura de redundância criada dos serviços GTex WS em comparação com o ambiente de produção do EMBRACE em São José dos Campos.

Figura 6.1 - Redundância do serviço GTex WS do EMBRACE.



O cenário de simulação da coleta da qualidade de serviço do GTex WS foi realizado de forma invasiva, ou seja, todos os recursos necessários para a coleta das métricas foram instaladas dentro do sítio do sistema alvo. A categoria das métricas coletadas foram as seguintes:

- **Métricas Embarcadas pelo AFQoS**
  - **AFQoSCollector** - código Java desenvolvido para coletar e processar as métricas do sistema alvo;
  - **Embarcada e Fora do Sítio ao mesmo tempo (MEAEFS)** - utilizando a ferramenta web analítica Google Analytics; e
  - **Terceiros** - utilizando a ferramenta analítica Zabbix;

Em seguida, a escolha das métricas foi baseada no documento de especificação de (BAKALOS et al., 2015). O mesmo apresenta uma pesquisa realizada sobre as diferentes métricas de SLA utilizadas pela ISO/IEC JTC 1/SC 38 (2016) e os principais CSP *Cloud Service Providers*[1] (Amazon Web Services SLA, Google Cloud SLA, Microsoft Azure SLA, Pivotal Web Services SLA e Heroku SLA).

[1]São empresas que oferecem serviços, infraestruturas ou aplicações na nuvem.

Para que o sistema alvo seja monitorado, primeiramente foi realizado o cadastro de todos os parâmetros necessários para o monitoramento do GTex *web service*, dessa forma toda métrica coletada do sistema alvo possui uma identificação detalhada de suas informações. As telas de cadastro podem ser vistas no Apêndice A.4. A listagem abaixo apresenta a sequência dos itens necessários para cadastrar um novo sistema alvo, um acordo de nível de serviço e suas respectivas métricas.

6.1 Cadastro dos parâmetros de QoS

1ª Parte: Essa etapa o engenheiro planejamento da QoS realiza apenas uma vez para cada dispositivo novo que fará parte da composição de monitoramento do AFQoS.

a) **Device**

- **Device**: Servidor HP 01
- **DeviceType**: Server
- **DeviceProperty**: Linux
- **DevicePropertyType**: S.O

b) **Entity**: GNSS

c) **Resource**

- **Resource**: AFQoSCollector, AFQoSPublisher
- **ResourceType**: Coletor, Publicador
- **ResourceProperty**:http://localhost:8080/climaespacial/gnss, HTTP
- **ResourcePropertyType**: URL, Protocolo

d) **DeviceResource**: Relacionar um Device a um Resource

e) **Architecture Element - AE**

***Web service* localizado em Cachoeira Paulista**

- **AE**: GText WS - CP
- **AEType**: Web Service
- **AEProperty**: /gtex/cp, REST
- **AEPropertyType**: Endpoint, Arquitetural

***Web service* localizado em Santa Maria**

- **AE**: GText WS - STA Maria
- **AEType**: Web Service
- **AEProperty**: /gtex/staMaria, REST
- **AEPropertyType**: Endpoint, Arquitetural

f) **Hosting**: Relacionar um AE a um Device

2ª Parte: Essa etapa é realizada com mais frequência, pois a cada métrica que deseja ser coletada, um novo registro deverá ser realizado.

a) **Definição das Métricas**

- **Name**: ResponseTime
- **Description**: SRT is an elapsed time between the end of a Request to a service and the beginning of the services response.
- **Scale**: Ordinal
- **Unit**: Milisecond
- **Frequency**: Each 1 minute
- **Attribute**: Performance
- **Dependency**: No dependency

b) **Cadastrar um SLA**

- **Name**: Space Weather Service Level Agreement GTex Application Service
- **Description**: This Space Weather GTex Application Service Service Level Agreement (SLA) is a policy governing the use of GTex Web Services
- **CreateDate**: 2017-01-01 16:00:00
- **ExpirationDate**: 2017-12-31 23:59:59
- **ImportanceLeve**: Highly Important
- **Resource**: AFQoSCollector
- **Status**: Active

c) **Service Contract**: Criação de um contrato de serviço entre um Resource e um Architecture Element

Contrato de Serviço para o sistema alvo localizado no sítio de Cachoeira Paulista

- **SLA**: Space Weather Service Level Agreement GOES Application Service
- **Resource**: AFQoSCollector
- **Architecture Element**: GText WS - CP
- **Importance Level**: Highly Important
- **Name**: Contrato de serviço GTex no Servidor HP 01 - CP
- **Status**: Ativo

Contrato de Serviço para o sistema alvo localizado no sítio de Santa Maria

- **SLA**: Space Weather Service Level Agreement GOES Application Service
- **Resource**: AFQoSCollector
- **Architecture Element**: GText WS - STA Maria
- **Importance Level**: Highly Important
- **Name**: Contrato de serviço GTex no Servidor Dell 01 - STA Maria
- **Status**: Ativo

d) Relacionar um Contrato de Serviço a uma Métrica (SLA Contract Metric)

Contrato de Serviço por Métrica do *web service* GTex de Cachoeira Paulista

- **Service Contract**: Contrato de serviço GTex no Servidor HP 01 - CP
- **Metric**: ResponseTime
- **ImportanceLevel**: Highly Important
- **Status**: Active
- **CreateDate**: 2017-01-01 17:00:00
- **DefaultValue**: 105ms

Contrato de Serviço por Métrica do *web service* GTex de Santa Maria

- **Service Contract**: Contrato de serviço GTex no Servidor Dell 01 - STA Maria
- **Metric**: ResponseTime
- **ImportanceLevel**: Highly Important
- **Status**: Active
- **CreateDate**: 2017-01-01 17:10:00
- **DefaultValue**: 105ms

A partir das considerações acima e em consonância com o objetivo que norteia este estudo, tem-se inicialmente a apresentação e discussão da coleta das métricas do tipo Embarcadas pelo AFQoSCollector. Em seguida, tem-se a coleta das métricas do tipo MEAEFS e Terceiros.

6.2 Coleta da métricas do tipo embarcadas pelo ArcticFoxQoS Agente Coletor

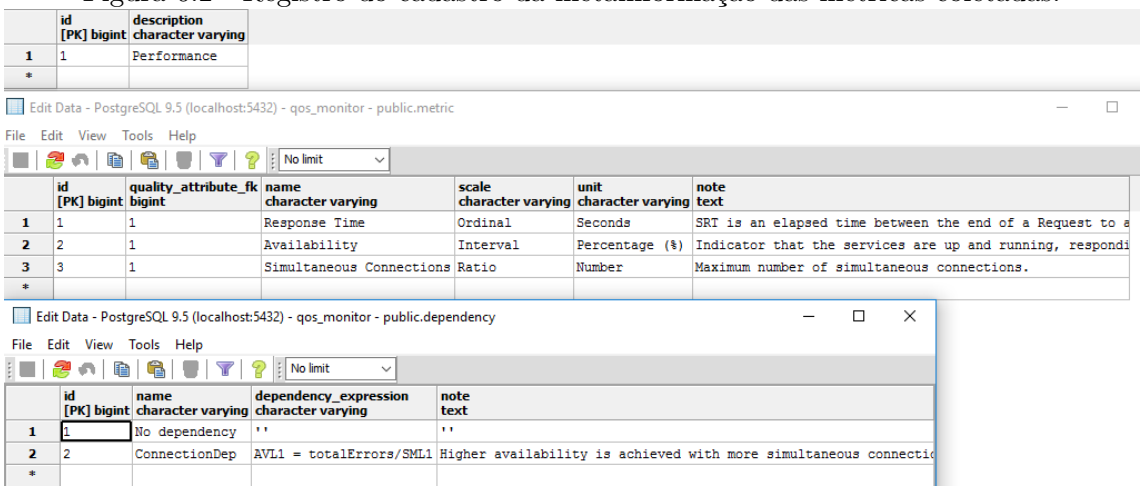
A coleta de métricas de QoS do sistema de *web services* do GTex, foi através do AFQoS Agente Coletor utilizando o método de coleta invasiva. Para que isso acontecesse foi necessário a criação dos seguintes itens dentro da Máquina Vitrual Ubuntu 16 LTS criada como redundância do ambiente operacional do EMBRACE para o ambiente de simulação.

- **Projeto Web Java Maven**;
- **Servidor de Aplicação Glassfish 4.0**;
- **EclipseLink** - Permite o java interagir com vários serviços, entre eles, banco de dados, *web services*, mapeamento XML, JSON, e outros;
- **Jersey** - Implementação Java para simplificar o serviço RESTful;
- **JMS JBoss AS 7** - Implementação Java para troca de mensagens assíncronas; e
- **Banco de dados postgresql** - Responsável pelo repositório local das métricas;

Como a aplicação do GTex é um serviço RESTful e disponível através de diversas URI´s, a maneira de acessar estes serviços são os próprios métodos HTTP. O *AFQoSCollector* é um cliente RESTful que foi desenvolvido com o propósito de trabalhar diretamente com os próprios métodos HTTP.

O atributo de qualidade escolhido foi a *Performance*. A escolha foi baseada no trabalho de (CHOI et al., 2007). Não esquecendo que todas as métricas foram pré cadastradas (ver seção 6.1) através do sistema de gerenciamento do AFQoS. Como pode ser visto na Figura 6.2 foram cadastrados o atributo, as métricas, e suas dependências. Dessa forma, quando a métrica for solicitada ficará factível a identificação do serviço, a data da coleta, dados de host, porta de comunicação do serviço, entre outras metainformações referentes às métricas coletadas.

Figura 6.2 - Registro do cadastro da metainformação das métricas coletadas.



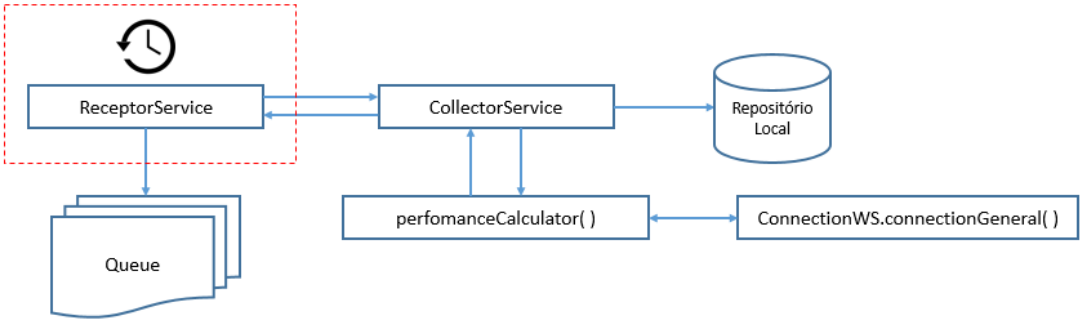
Após a criação do Projeto *CollectorQoSAgentWS*, foi criado o agente *CollectorService* cuja a responsabilidade é conectar ao serviço cliente, coletar a métrica, processar e persistir no repositório local. O método responsável pelo cálculo é o *performanceCalculator(String url)* onde é passada a URL do cliente a ser conectada e uma *Thread* a cada N segundos faz uma requisição.

O serviço responsável por enfileirar as métricas coletadas pelo *CollectorService* é *ReceptorService*.

A Figura 6.3 mostra o fluxo desde a coleta, processamento, armazenamento local e

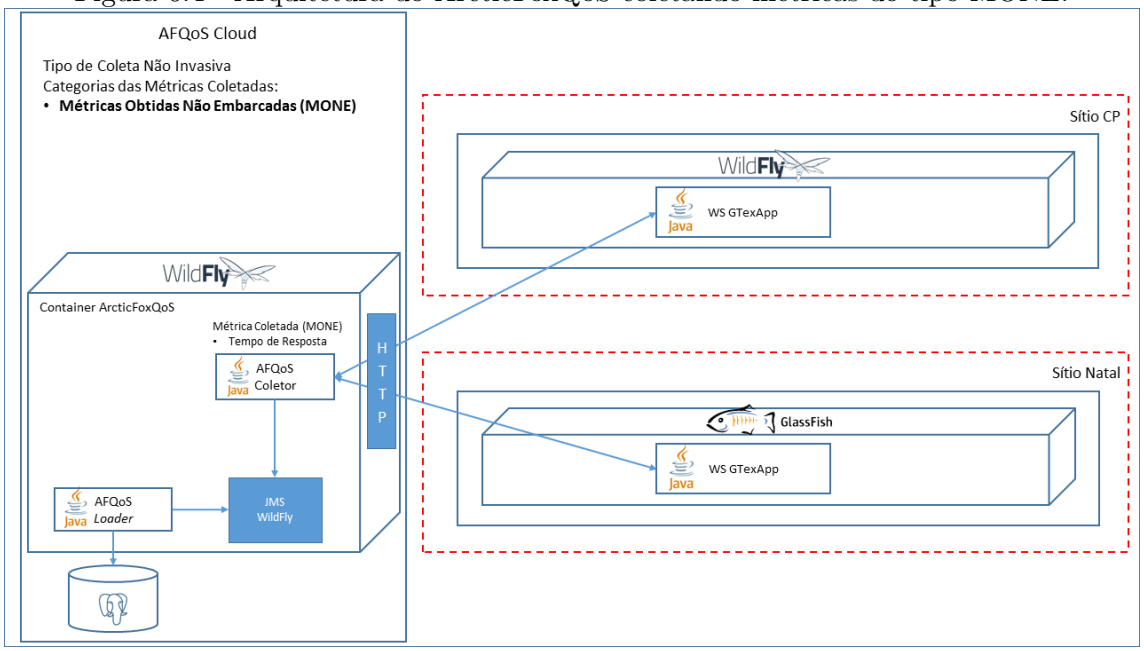
enfileiramento da métrica.

Figura 6.3 - Pipeline da coleta realizada pelo AFQoSCollector.



A Figura 6.4 apresenta o modelo arquitetural de implantação do AFQoS usado para este estudo de caso.

Figura 6.4 - Arquitetura do ArcticFoxQoS coletando métricas do tipo MONE.



A realização deste experimento foi de 1 hora por dia totalizando um período de

1 mês. O objetivo foi simular um horário de pico de requisições de um serviço. O coletor acessava o serviço do *GTex WS* obtendo o tempo de resposta do mesmo, e a cada 1 minuto ele repetia o processo. Na Figura 6.5 é apresentada uma tabela com a média do tempo de resposta dos dois sistemas monitorados CP e Sta. Maria respectivamente.

Figura 6.5 - Comparação do Tempo de Resposta do serviço GTex WS.

Architecture Element: GTex WS
Architecture Element Property: /gnss/gtex/arch_elem/{id}{initialDate}/{finalDate}
Metric: Response Time
Dependency: No Dependency
Scale: Ordinal
Unit: Second
Frequency: Each 1 minute during 1 hour

Período de solicitação de dados do GTex	Média de Cachoeira Paulista	Média de Santa Maria
2016-01-01 a 2016-01-31	0,337	0,317

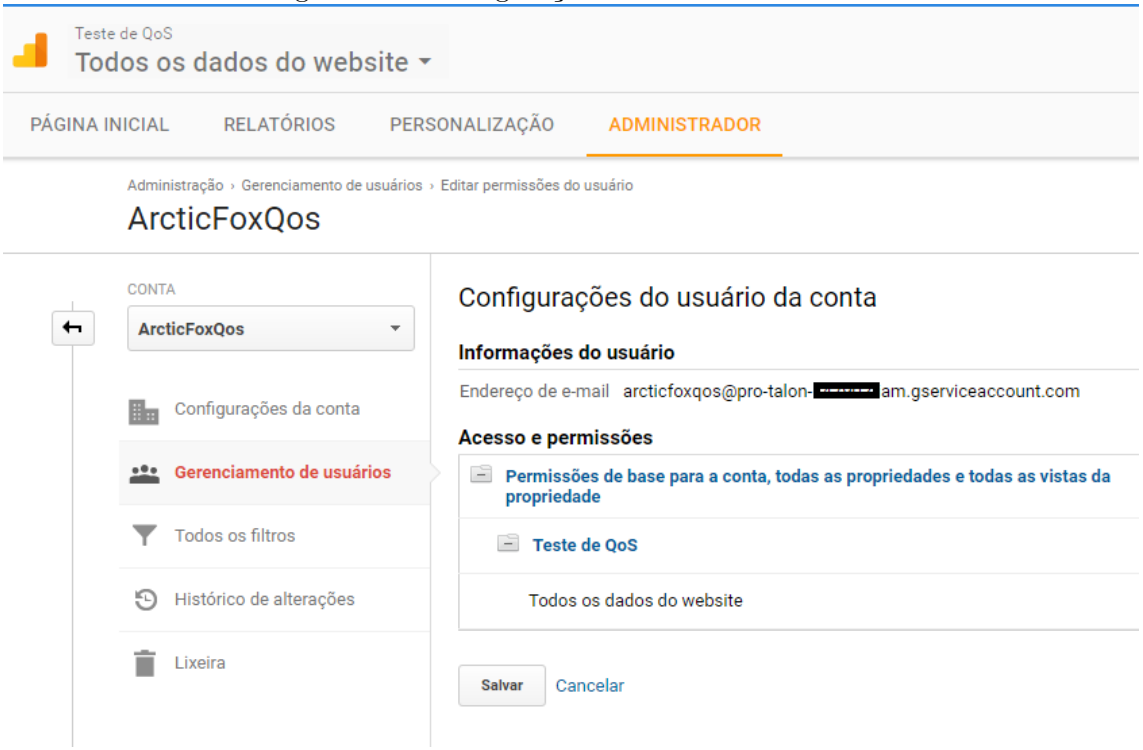
6.3 Coleta das métricas do tipo embarcada e fora do sítio ao mesmo tempo

O agente responsável em coletar as métricas do tipo **Embarcada e Fora do Sítio ao mesmo tempo** é o *AFQoSCollectorAnalyticsReporting*. É um código cliente desenvolvido em Java que utiliza a API do Google Analytics (GA) com a finalidade de coletar dados de qualidade de serviço em tempo real e/ou definidos por um período pré estabelecido do sistema web monitorado. Para que a integração com a ferramenta de análise funcionasse, foram necessários alguns passos.

Inicialmente criou-se uma conta no GA onde através dela foi possível verificar todo o gerenciamento do sistema monitorado. Além disso, definir métricas, criar e visualizar relatórios de interesses, como segmentos de mercado, localização, idioma, sistema operacional, navegador, entre outros. Em seguida ainda dentro do painel de monitoramento, foi criada uma nova conta de serviço denominada GTex onde ao passar a URL do sistema alvo é gerado um código fonte que deve ser embarcado dentro da aplicação que será monitorada. Para que o *AFQoSCollectorAnalyticsRe-*

porting utilize a API de relatórios do GA, inicialmente deve-se usar uma ferramenta de configuração que orienta a criação de um projeto no console de APIs do Google, para a ativação da API e na criação de credenciais. A Figura 6.6 mostra o ambiente necessário para a configuração da conta no GA.

Figura 6.6 - Configurações da conta do usuário.



A API do GA permite a criação de um corpo de solicitação e corpo de resposta baseada em um conjunto de objetos do tipo *Report* da API. Dessa forma foi possível estruturar o formato de entrada (quais métricas seriam coletadas) e seu formato de saída.

A Figura 6.7 mostra o exemplo JSON utilizado neste estudo de caso para a solicitação e a resposta das métricas coletadas.

Figura 6.7 - Arquivo JSON usado para modelar as métricas de entrada.

Request JSON

```
{
   "reportRequests":[
      {
         "viewId":"XXXXXXXX",
         "dateRanges":[
            {
               "endDate":"today",
               "startDate":"7DaysAgo"
            }
         ],
         "dimensions":[
            {"name":"ga:pagePath"}
         ],
         "metrics":[
            {"expression":"ga:pageviews"}
            {"expression":"ga:uniquePageviews"},
            {"expression":"ga:timeOnPage"},
            {"expression":"ga:bounces"},
            {"expression":"ga:entrances"},
            {"expression":"ga:exits"}
         ],
         "orderBys":[
            {"fieldName":"ga:pageviews"}
         ]
      }
   ]
}
```

Response JSON

```
{
   "reportRequests":[
      {
         "viewId":"XXXXXXXX",
         "dimensions":[
            {"name":"/index.html"}
         ],
         "metrics":[
            {"pageviews":"21"},
            {"uniquePageviews":"6"},
            {"timeOnPage":"389.0"},
            {"bounces":"1"},
            {"entrances":"6"},
            {"exits":"6"}
         ]
      }
   ]
}
```

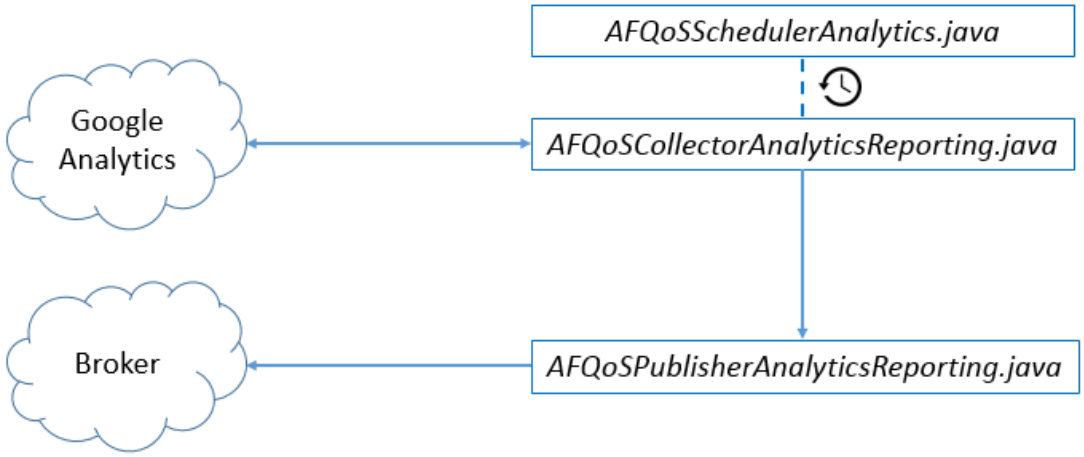
Legenda das métricas:

- **pageviews**: Visualizações de página
- **uniquePageViews:** Calcula o valor de visitas ao site eliminando as múltiplas visitas à mesma página.
- **timeOnPage:** Tempo de visita na página (segundos)
- **bounces:** Taxa de Rejeição
- **entrances:** Valor total de entradas no site no primeiro hit de exibição da página
- **exits:** Calcula a quantidade de visitantes que estavam no site e saíram. Além disso, diferentemente da taxa de rejeição, a taxa de saída não leva em consideração se foi a primeira visita ou não do usuário no site.

Para garantir que os dados coletados fossem os mais atualizados, foi criado um *scheduler* onde a cada N minutos o *AFQoSCollectorAnalyticsReporting* é executado e todo o processo de aquisição de novos dados de qualidade de serviço é obtido novamente. Para enviar os dados coletados ao repositório central do AFQoS localizado

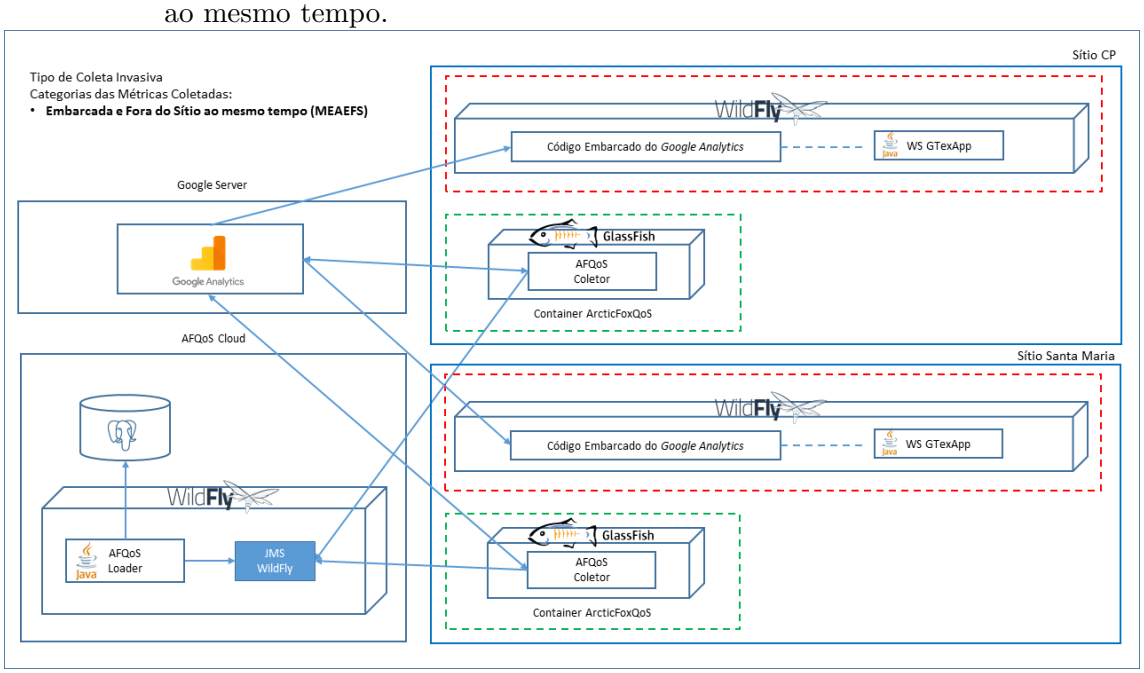
na nuvem, foi criado um agente *AFQoSPublisherAnalyticsReporting* que implementa o protocolo MQTT para o envio dos dados coletados ao broker instalado na nuvem, como ilustrado a seguir.

Figura 6.8 - Pipeline de coleta e envio dos dados coletados pelo AFQoS.



A Figura 6.9 exibe o modelo de arquitetura de implantação do AFQoS, suas interações e as tecnologias utilizadas para a realização da coleta.

Figura 6.9 - Arquitetura do AFQoS coletando métricas do tipo Embarcada e Fora do Sítio ao mesmo tempo.



A coleta das métricas no estudo de caso relatado acima, foi durante o período de 7 dias. A URL monitorada foi de um sistema de teste criado e implantado em um servidor da Amazon AWS utilizando os serviços gratuitos que o mesmo oferece. A Figura 6.10 abaixo mostra o resultado da coleta de QoS dos sítios (dois web sites simulando CP e Santa Maria).

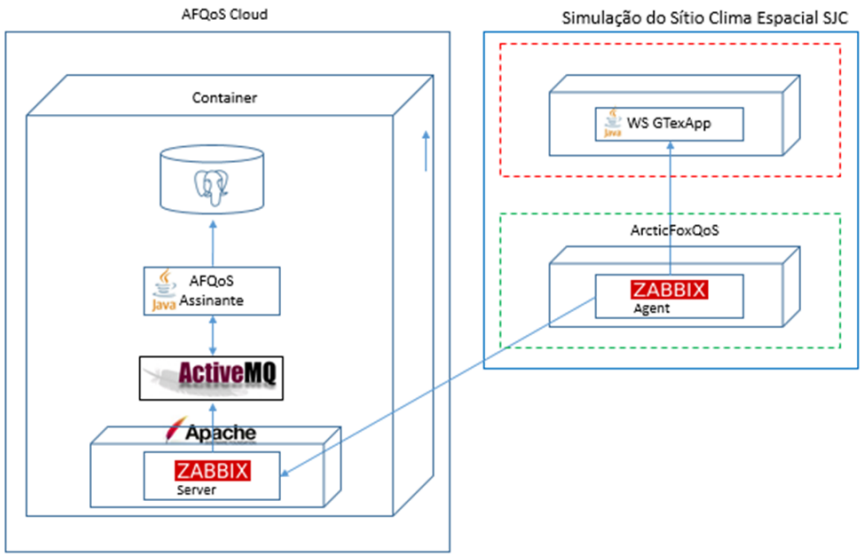
Figura 6.10 - Tabela de comparação entre os sítios de Cachoeira Paulista X Santa Maria.

Métrica	Cachoeira Paulista	Santa Maria
pageViews	21	12
uniquePageViews	6	2
timeOnPage	389	129
bounces	1	0
entrances	6	1
exits	6	1

6.4 Coleta das métricas do tipo embarcadas pelo ArcticFoxQoS - Terceiros

O terceiro e último estudo de caso utilizou-se a ferramenta de monitoramento Zabbix para a realização da coleta de métricas de QoS do sistema alvo. A instalação do Zabbix foi realizada na máquina virtual ubuntu LTS 16 64-bits, 2048MB. Após a instalação e configuração do Zabbix *Server* e *Agent*, um servidor Jboss WildFly foi instalado para implantar a aplicação de *web service* do GTex. A métrica coletada foi o PING do servidor cuja aplicação do GTex está implantada. Basicamente o agente Zabbix coleta a informação e entrega ao Zabbix Server. Como o Zabbix possui integração com o protocolo MQTT, foi possível criar um agente assinante denominado *SubscriberQoSAgentWS* (ver Apêndice A.7) e selecionar as mensagens publicadas pelo Zabbix nos tópicos configurados, processá-las e armazená-las no repositório de métricas.

Figura 6.11 - Arquitetura do AFQoS coletando métricas do tipo Terceiro.



A Figura 6.12 apresenta o tópico criado para gerenciar as mensagens enviadas pelo agente do Zabbix através do *broker ActiveMQ*. Além disso a ferramenta gráfica do Zabbix mostra os valores em tempo real do dado coletado.

Figura 6.12 - Tópico criado para gerenciar os dados coletados do servidor wildfly.

6.4.1 ArcticFoxQoS Implementação

Para que fosse possível registrar as metainformações de QoS dos sistemas que fazem parte da composição de monitoramento do AFQoS, foi necessário criar um sistema de gerenciamento capaz de fornecer recursos ao engenheiro de qualidade. A estrutura de pacotes do *back-end* do sistema possui, o pacote raiz do projeto denominado *arcticfox.qos*, os sub-pacotes bem como as implementações de classes necessárias para a manipulação das metainformações e podem ser vistos no Apêndice A.3.

6.4.1.1 Estrutura do Projeto

- **Pacote Config**: esse pacote contém a classe *AppConfig* na qual é responsável pela configuração dos serviços REST. Nela são relacionadas todas as classes que deverão ser expostas como serviço.

- **Pacote Data**: as classes deste pacote são as entidades que representam o modelo de dados da Figura B.1. Além disso as mesmas possuem *Annotations* de persistência para a manipulação dos dados utilizando ORM *Object-relational mapping)* [2].

[2]é uma técnica capaz de fazer uma relação dos objetos das entidades do modelo orientado a objetos com o modelo de dados relacional

- **Pacote Filter**: as classes desse pacote são responsáveis por criar um filtro entre a requisição e a resposta que o serviço irá entregar ao cliente. Questões como serialização e deserialização de objetos e tratamento de requisições *CORS* (compartilhamento de recursos de origem cruzada) são alguns exemplos de funcionalidades das classes contidas neste pacote.

- **Pacote Services**: classes responsáveis por expor os métodos de acesso criando um canal de comunicação entre o cliente e o serviço a partir de uma URI *Uniform Resource Identifier*.

- **Pacote Resource**: são as classes responsáveis por representar a resposta que será entregue ao serviço consumidor. Para cada serviço exposto, haverá uma classe resource. Exemplo: um serviço que entrega todas as métricas coletadas nas últimas 24h, então pode-se criar uma classe *MetricFrequency-Resource.class* que corresponde ao formatos dos dados requisitados.

- **Pacote SessionBean**: são as classes responsáveis pela lógica da aplicação.

- **Pacote SessionBeanLocal**: são as interfaces que definem como as classes de negócio devem implementar de forma concreta suas regras de negócio.

O *front-end* do sistema de gerenciamento do AFQoS responsável pela visualização e manipulação dos dados pelo engenheiro de qualidade, possui e seguinte estrutura:

- **Pacote Module**: Responsável pela subdivisão da aplicação. Cada **Entidade** do sistema possui um arquivo de modulo com as configurações necessárias para fazer a integração entre as camadas de *controller / view*. Permitindo o completo desacoplamento entre os elementos arquiteturais.

- **Pacote Config**: Indica quais módulos serão adicionados à aplicação;

- **Pacote Controller**: Responsável pelo comportamento das *Views*;

- **Pacote Service**: Cliente consumidor dos serviços disponibilizados pelo *back-end*; e (exemplo do código fonte no Apêndice A.9)

- **Pacote View**: Neste pacote estão localizados todos os arquivos HTML responsáveis pela visualização das páginas.

Este capítulo apresentou 3 formas de implantação do AFQoS para a realização da coleta de dados de QoS. Primeiramente foi realizado o cadastro das metainformações

sobre os parâmetros de configuração dos dispositivos/serviços bem como as métricas, seus atributos e por fim o SLA. Em seguida, o primeiro estudo de caso foi através do agente coletor *ArcticFoxQoSCollector* desenvolvido em JAVA e embarcado na própria arquitetura do cliente. A simulação resultou em uma tabela comparativa em relação à média do tempo de resposta do serviço GTex instalado em Cachoeira Paulista/SP e Santa Maria/RS respectivamente.

O segundo experimento foi utilizando a ferramenta do Google Analytics onde foi possível coletar métricas sobre os serviços do GTex localizados em Cachoeira Paulista/SP e Santa Maria/SP respectivamente. Ao final da simulação obteve-se uma tabela comparativa com as métricas de QoS coletadas do serviço GTex instalado nos referidos municípios respectivamente.

Finalmente o último estudo de caso foi utilizando a ferramenta de monitoramento Zabbix e com ela foi possível obter métricas do servidor Wildfly onde estava instalada a aplicação do GTex.

Apesar da implementação do AFQoS apresentar a viabilidade da arquitetura de referência proposta, os testes realizados nos sistemas alvo foram através de ambientes redundantes e simulados, tanto para os sistemas monitorados quanto para o parque computacional dos sítios (como os servidores e os contêineres para as aplicações web, instalação e configuração dos sistemas operacionais, entre outros). Visto que, ao implantar uma arquitetura desse porte em um ambiente operacional, algumas políticas de segurança devem ser levadas em consideração por parte das organizações dispostas a permitir a implantação da arquitetura. Dessa forma não foi possível medir o impacto da coleta em ambientes operacionais. Entretanto as ferramentas analíticas utilizadas no estudo de caso são consolidadas no mercado e não possuem relatos negativos a respeito da sua aplicabilidade para a realização da coleta de dados de QoS. Assim, podemos dizer que possivelmente a implantação do AFQoS em sistemas reais não causará impacto em relação às características não funcionais da arquitetura.

7 CONCLUSÃO

Este trabalho propõe uma arquitetura para a avaliação da qualidade (*Quality of Service - QoS*) de serviço de sistemas de segmento solo e de seus componentes. A principal contribuição desse estudo é a criação de um padrão de avaliação da QoS para esses sistemas e seus componentes através da arquitetura de referência proposta. Ainda, ser utilizada em arquiteturas autoadaptáveis e estender a coleta de QoS para diversos tipos de dispositivos, como satélites, celulares, sistemas embarcados, e muitos outros.

Para a implementação do modelo arquitetural foi desenvolvido o sistema *Arctic-FoxQoS* (AFQoS) desenvolvido com a tecnologia JAVA. Os agentes de coleta apresentados na arquitetura proposta e no AFQoS mostraram que é possível coletar os dados dos sistemas alvo independente da tecnologia, seja ela por código fonte desenvolvido apenas para este fim ou por meio de ferramentas de monitoramento e análise de QoS.

O padrão arquitetural utilizado foi o de camadas, permitindo que cada componente da estrutura fosse construído de forma independente. A arquitetura encontra-se dividida nas seguintes camadas: Dispositivo/Serviços; Agente Embarcado; Repositório Embarcado; Comunicação; Agente de Coleta Não Embarcado; Repositório; Metainformação; Processador de Eventos; Sistema de Entregas API; Segurança; e Aplicação.

A camada de dispositivos e serviços está baseado em um modelo de arquitetura de internet das coisas (*Internet of Things* - IoT) proposto por (BASSI et al., 2013). Dessa forma é possível coletar QoS de qualquer elemento que possua algum tipo de interface de comunicação. As camadas de comunicação foram baseadas visando a interoperabilidade entre os sistemas viabilizando a transferência dos dados coletados para o repositório. Todas as informações sobre a QoS ficam centralizadas e disponíveis em repositórios de metainformações. A elaboração do sistema de gerenciamento de metainformação implementado no AFQoS, permitiu o registro dos recursos necessários para o monitoramento das métricas, como os dispositivos, serviços e seus parâmetros de configuração, acordos de níveis de serviço a fim de criar uma garantia de qualidade aos serviços de quem está provendo, os atributos, métricas, frequência de coleta, status dos serviços e ainda questões de segurança, como a autorização e a autenticação ao implantar um novo sistema alvo. Também, foi criada uma API REST com o propósito de criar um contrato de uso dos serviços, proporcionando uma interface padronizada aos interessados em obter os dados coletados. Além disso,

o serviço de entrega também pode ser implementado utilizando outros protocolos como o SOAP.

Com a criação de um banco de dados de conhecimento gerado pela implementação da arquitetura, a geração dos indicadores de QoS permitirá a criação de medidas preventivas e corretivas dos serviços prestados, bem como a redução de custos ao implantar recursos autônomos baseados na avaliação da QoS. Outras medidas também podem ser tomadas como aprimoramento da infraestrutura, como o aperfeiçoamento de processos, serviços, sistemas e produtos, além da replicação ou redundância de ambientes. O último identificará quais os serviços dentre aqueles prestados estão em melhores condições de qualidade e quais os requisitos não funcionais dos sistemas não estão atendendo de forma satisfatória os critérios de QoS considerados. Isso permitirá a seleção do serviço com melhores índices de QoS daqueles oferecidos pela arquitetura. Como impacto principal do emprego de QoS, Silva (2009) em sua "2ª Pesquisa sobre o Perfil dos Usuários das Imagens do Satélite Sino-Brasileiro de Recursos Terrestres (CBERS)" indicou que o uso gratuito dessas imagens de satélite gerou diversos benefícios a seus usuários, como a geração de economia e a modernização e melhorias de suas infraestruturas de máquinas e equipamentos, elevando a importância da prestação de serviços com qualidade. Portanto, expande-se a possibilidade de angariar recursos financeiros para a manutenabilidade desses sistemas provedores de serviços.

Como sugestão de trabalhos futuros, propõe-se a implantação de sistemas que realizam a autoadaptação em tempo real baseados no conhecimento do contexto situacional dos sistemas monitorados. Assim, requisitos como autoconfiguração, autocura, identificação de problemas arquiteturais, indisponibilidade de sistemas de controle, acordos de níveis de serviço bem definidos, reusabilidade e outros, poderão ser aplicados em qualquer ambiente computacional. Contudo, a implementação do AFQoS em um ambiente operacional permitirá a validação completa da arquitetura e certamente novos requisitos aparecerão. Outra proposta interessante é a diversificação dos tipos de sistemas alvo a serem monitorados. Como a arquitetura proposta permite qualquer elemento avaliar a QoS, tornar-se-á interessante monitorar dispositivos de hardware como controladores, sensores, satélites e muitos outros.

Para finalizar ressalta-se que o AFQoS foi implementado utilizando as tecnologias citadas no decorrer do trabalho. Entretanto outras formas de implementação poderiam ser adotadas, como diferentes tecnologias e outros modelos de acordo de níveis de serviço, protocolos para realizar a comunicação entre os sistemas alvo e os agen-

tes de coleta e utilização de outras soluções para a autorização e a autenticação dos dispositivos. Portanto, conclui-se que a arquitetura de referência proposta é independente de tecnologia, pois seguindo os padrões arquiteturais explicitados, diversas formas de avaliação da QoS podem ser desenvolvidas e implementadas.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, C. **A pattern language:**: towns, buildings, construction. New York: Oxford University Press, 1977. 50

ANGELOV, S.; GREFEN, P.; GREEFHORST, D. A classification of software reference architectures: Analyzing their success and effectiveness. In: INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, Cambridge. **Proceedings...** Robinson College Cambridge, United Kingdom: IEEE, 2009. p. 141–150. 46

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **ABNT NBR ISO 9001**: Sistemas de gestão de qualidade-requisitos: quality management systems-requirements. Rio de Janeiro, 2008. 27

BAKALOS, N.; KYRIAZIS, D.; PROTONOTARIOS, E.; VARVARIGOU, T.; BARRETO, O. **SLA specification and reference model-a D3. 2**. [S.l.: s.n.], 2015. 5, 8, 36, 37, 71, 72, 73, 90

BARBOSA, M. N. A importância estratégica da cooperação internacional na área do espaço. **Parcerias Estratégicas**, v. 4, n. 7, p. 131–136, 2009. 1

BARESI, L.; GUINEA, S. Architectural styles for adaptive systems: A tutorial. In: IEEE INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS (SASO), 6., 2012, Lyon, France. **Proceedings...** [S.l.]: IEEE, 2012. p. 237–238. 61

BARESI, L.; NITTO, E. D.; GHEZZI, C. Toward open-world software: Issues and challenges. **Computer**, IEEE, v. 39, n. 10, p. 36–43, 2006. 60

BASSI, A.; BAUER, M.; FIEDLER, M.; KRAMP, T.; KRANENBURG, R. V.; LANGE, S.; MEISSNER, S. **Enabling things to talk**. [S.l.]: Springer, 2013. 163–211 p. 4, 7, 60, 107

BERNS, A.; GHOSH, S. Dissecting self-* properties. In: IEEE INTERNATIONAL CONFERENCE ON SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS, 3., 2009, San Francisco, CA. **Proceedings...** [S.l.]: IEEE, 2009. p. 10–19. 62

BOSCH, J. **Design and use of software architecture:**: adopting and evolving a product line architecture. [S.l.]: Addison-Wesley, 2000. 45

BRAY, T. The javascript object notation (json) data interchange format. 2014. 5

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. C. **Pattern-oriented software architecture, on patterns and pattern languages**. [S.l.]: John wiley & sons, 2007. 50

CAMPOS, V. F. **TQC: controle da qualidade total**. [S.l.: s.n.], 1992. v. 11. 27

CARDOSO, P. E.; BARRETO, J. P.; DOBROWOLSKI, K. M.; CARDOSO, L. S.; HOFFMANN, L. T. A ground control system for cbers 3 and 4 satellites. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS WILL BE HOSTED BY THE, 9TH., 2006, Rome, Italy. **Proceedings...** [S.l.]: ASI, 2006. p. 8. CD-ROM. (INPE-14071-PRE/9240). 7, 14, 17

CARVALHO, F.; MEIRA, S. R.; FREITAS, B.; CARVALHO, N. A quality model for embedded software component. In: BRAZILIAN WORKSHOP ON REAL-TIME AND EMBEDDED SYSTEMS, 11., 2009, Recife. **Proceedings...** [S.l.], 2009. 9

CARVALHO, M. J. M.; LIMA, J. S. S.; JOTHA, L. S.; AQUINO, P. S. Conasat-constelação de nano satélites para coleta de dados ambientais. In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 16. (SBSR)., 2013, Fox do Iguaçu. **Anais...** São Jose dos Campos: INPE, 2013. p. 9108–9115. ISBN 978-85-17-00066-9. Disponível em: <http://urlib.net/3ERPFQRTRW34M/3E7GL7J>. 2

CEREJA JUNIOR, M. G. Arctic fox project - a framework to provide self-adaptive support to space ground systems: A proof of concept. In: INTERNATIONAL ASTRONAUTICAL CONGRESS, 67., 2016, Guadalajara, Mexico. **Proceedings...** IAF, 2016. Disponível em: <http://www.iac2016.org/pdf/IAC2016_FP_FULL.pdf>. 65, 66

_____. A framework to provide self-adaptive support to scientific software systems. In: CONFERENCE OF COMPUTATIONAL INTERDISCIPLINARY SCIENCE, 4th., 2016, São José dos Campos. **Proceedings...** CCIS, 2016. Disponível em: <http://www.epacis.net/ccis2016/papers/paper_98.pdf>. 66

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, 2007, Hong Kong, China. **Proceedings...** [S.l.]: IEEE, 2007. p. 622–625. DOI: 10.1109/ICEBE.2007.107. 4, 8, 27, 29, 30, 71, 95

CLEMENTS, P.; GARLAN, D.; BASS, L.; STAFFORD, J.; NORD, R.; IVERS, J.; LITTLE, R. **Documenting software architectures:**: views and beyond. [S.l.]: Pearson Education, 2002. ISBN 0201703726. 46

CLEMENTS, P. C. **Software architecture in practice**. Tese (Doutorado) — Software Engineering Institute, 2002. 46

COMMISSION., E. Cloud computing service level agreements. 2013. 36

COMPRESSION, L. D. Report concerning space data system standards. CCSDS, 2013. 13

COSTA L. L.; DURÃO, O. S. C. S. N. J. Projeto de um aplicativo de bordo para missão nanosatc-br. In: SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DO INPE (SICINPE), 2011, São José dos Campos. **Anais...** São José dos Campos: INPE, 2011. p. 64. Disponível em: <http://urlib.net/8JMKD3MGP7W/3AGF5BE>. 2

DEITEL, H. M.; DEITEL, P. J. **Como programar em C**. [S.l.]: LTC, 1999. 43

DINIZ, E. H. Internet das coisas. **GVexecutivo**, v. 5, n. 1, p. 59, 2006. 7

EELES, P. **Capturing architectural requirements**. 2005. Disponível em: <http://itculiacan.webs.com/descargas/z_FURPS%20Requisites.pdf>. 65

ERL, T. **Service-oriented architecture**: a field guide to integrating xml and web services. [S.l.]: Prentice Hall PTR, 2004. 58

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **ECSS-E-ST-70C**: Ground systems and operations (31 july 2008). Noordwijk, The Netherlands, 2008. 12

FERREIRA, M. G. V. **Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao software de controle de satélites**. 2001. 244 p. INPE-8602-TDI/787. Tese (Doutorado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2001. Disponível em: <http://urlib.net/83LX3pFwXQZ5Jpy/wMD3P>. 14

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. Tese (Doutorado) — University of California, Irvine, 2000. 53

FREEMAN, E.; FREEMAN, E.; SIERRA, K.; BATES, B. **Padrões de projeto**. [S.l.]: Alta Books Rio de Janeiro, 2005. 49

GALLO, M. A.; HANCOCK, W. S.; SILVA, F. S. C. da; CARNEIRO, M. R. de F.; MELO, A. C. V. de. **Comunicação entre computadores e tecnologias de rede**. [S.l.]: Pioneira Thomson Learning, 2003. 54

GAMMA, E. **Design patterns**: elements of reusable object-oriented software. [S.l.]: Pearson Education India, 1995. 50

GARCIA, D. Z. G.; TOLEDO, M. B. F. Semantics-enriched qos policies for web service interactions. In: WEBMEDIA '06 PROCEEDINGS OF THE 12TH BRAZILIAN SYMPOSIUM ON MULTIMEDIA AND THE WEB, 2006, Natal, Rio Grande do Norte, Brazil. **Proceedings...** New York: ACM, 2006. Disponível em: <http://doi.acm.org/10.1145/1186595.1186601>. 30, 31

GARCIA, J. R. M.; MOREIRA, J. L. Bdi-banco de dados de imagens de satélite. In: CONGRESSO BRASILEIRO DE METEOROLOGIA, 14. (CBMET)., 2006, Florianópolis, SC. **Proceedings...** 2006. (INPE-14306-PRE/9394). Disponível em: <http://www.sbmet.org.br/congresso/port/index.html>. 25

GARLAN, D.; SHAW, M. An introduction to software architecture. **Advances in software engineering and knowledge engineering**, Singapore, v. 1, n. 3.4, 1993. 52

GROUP, I. et al. **An architectural blueprint for autonomic computing**. 2006. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.1011&rep=rep1&type=pdf>. 61, 62

GUERRA, E. **Design patterns com java**: projeto orientado a objetos guiado por padrões. [S.l.]: Editora Casa do Código, 2014. 49

HERZUM, P.; SIMS, O. **Business Components Factory**: a comprehensive overview of component-based development for the enterprise. [S.l.]: John Wiley & Sons, Inc., 2000. 48

HOFMEISTER, C.; NORD, R. L.; SONI, D. Describing software architecture with uml. In: DONOHOE, P. e. (Ed.). **Software Architecture**. [S.l.]: Springer, 1999. p. 145–159. 48

IBM. **Get it right the first time**: writing better requirements. [S.l.]: Copyright IBM Corporation, 2009. 65

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC. IEEE. **Std 1471-2000**: Ieee recommended practice for architectural description of software-intensive systems. New York, 2000. 46

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). **ISO/IEC19086**: Information technology cloud computing - service level agreement (sla) framework. Geneva, 2016. 29, 90

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. 32

JONES, S. Toward an acceptable definition of service [service-oriented architecture]. **IEEE software**, IEEE, v. 22, n. 3, p. 87–93, 2005. 27

KRUCHTEN, P. B. The 4+ 1 view model of architecture. **IEEE software**, IEEE, v. 12, n. 6, p. 42–50, 1995. 47

LAMPKIN, V.; LEONG, W. T.; OLIVERA, L.; RAWAT, S.; SUBRAHMANYAM, N.; XIANG, R.; KALLAS, G.; KRISHNA, N.; FASSMANN, S.; KEEN, M. et al. **Building smarter planet solutions with mqtt and ibm websphere mq telemetry**. [S.l.]: IBM Redbooks, 2012. 58

LARSON, W. J.; WERTZ, J. R. **Space mission analysis and design**. Third edition. [S.l.: s.n.], 1999. 1, 11, 12, 13

LUDWIG, H.; KELLER, A.; DAN, A.; KING, R. P.; FRANCK, R. **Web service level agreement (WSLA) language specification**. 2003. 815–824 p. 37, 38

MATTIELLO-FRANCISCO, M. F. M. **Sistemas computacionais em aplicações espaciais**. INPE, 2003. Disponível em: <<http://urlib.net/sid.inpe.br/jeferson/2003/10.13.15.25>>. Acesso em: 11 Dez. 2017. 12

MCCARTNEY, S. **ENIAC**: the triumphs and tragedies of the world's first computer. [S.l.]: Walker & Company, 1999. 43

MIRANDOLA, R.; POTENA, P. A qos-based framework for the adaptation of service-based systems. **Scalable Computing: Practice and Experience**, v. 12, n. 1, 2011. 8

O'BRIEN, L.; MERSON, P.; BASS, L. Quality attributes for service-oriented architectures. In: INTERNATIONAL WORKSHOP ON SYSTEMS DEVELOPMENT IN SOA ENVIRONMENTS, SDSOA '07, 2007, MINNEAPOLIS, MN, USA. **Proceedings...** IEEE, 2007. p. 3. ISBN 0-7695-2960-7. Disponível em: <http://dx.doi.org/10.1109/SDSOA.2007.10>. 3, 29

OREIZY, P.; GORLICK, M. M.; TAYLOR, R. N.; HEIMBIGNER, D.; JOHNSON, G.; MEDVIDOVIC, N.; QUILICI, A.; ROSENBLUM, D. S.; WOLF, A. L. An architecture-based approach to self-adaptive software. **IEEE Intelligent systems**, IEEE Educational Activities Department, v. 14, n. 3, p. 54–62, 1999. 60, 61

ORLANDO, V.; KUGA, H. K. Os satélites scd1 e scd2 da missão espacial completa brasileira-mecb. In: WINTER; C., O.; PRADO, A. F. B. A. e. (Ed.). **A conquista do espaço:**. do sputnik à missão centenário. São Paulo: Editora Livraria da Física, 2007. p. 151–176. ISBN 978-85-88325-89-0. 1, 11

PARASURAMAN, A.; ZEITHAML, V.; BERRY, L. Servqual: a multiple-item scale for measuring consumer perceptions of service quality. **Retailing: critical concepts**, v. 64, n. 1, p. 140, 2002. 28

PATEL, P.; RANABAHU, A. H.; SHETH, A. P. **Service level agreement in cloud computing**. [S.l.]: CORE Scholar, 2009. 32

PHILIP, P. B.; LEWIS, G.; MERSON, P. **Service Level Agreements in Service-Oriented Architecture Environments**. 2008. 32

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. **ACM Transactions on Autonomous and Adaptive Systems (TAAS)**, ACM, v. 4, n. 2, p. 14, 2009. 61

SANT'ANNA, 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: SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO. ,2014, Londrina. **Anais...** [S.l.], 2014. p. 125–136. 6, 21, 23

SANTOS, M. A. F.; MATTIELO-FRANCISCO, M. F. M.; YAMAGUTI, W. O sistema nacional de dados ambientais e a coleta de dados por satélite. In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 16. (SBSR)., 2013, Foz do Iguaçu. **Anais...** São José dos Campos: INPE, 2013. p. 9116–9123. ISBN 978-85-17-00066-9. Disponível em: <http://urlib.net/3ERPFQRTRW34M/3E7GKDH>. 6, 23, 24

(SEI)., S. E. I. **Architecture Evaluation at the SEI**. 2016. Disponível em: <http://resources.sei.cmu.edu/asset_files/Brochure/2016_015_001_484457.pdf>. Acesso em: 10 Jan. 2017. 45

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. 9, 27, 29

SILVA, L. T. d. **Perfil dos Usuários CBERS - 2ª Pesquisa - 2009**. Instituto Nacional de Pesquisas Espaciais, 2009. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m17@80/2009/10.14.15.46>. 2, 108

SOMMERVILLE, I. **Engenharia de software**. [S.l.: s.n.], 2012. 27, 44, 46, 48, 49, 52, 65

SORENSEN, T. A university-developed comprehensive open-architecture space mission operations system (cosmos) to operate multiple space vehicles. In: SPACEOPS 2012 CONFERENCE, 2012, Stockholm, Sweden. **Proceedings...** 2012. p. 1296468. Disponível em: <http://dx.doi.org/10.2514/MSPOPS12>. 17, 18, 19, 20

SOUZA, P. B.; MONICO, J. F. G.; TAKAHASHI, H.; GUEDES, F. B. Processamento de dados de rádio ocultação da constelação de satélites cosmic-2. In: INTERNATIONAL CONGRESS OF THE BRAZILIAN GEOPHYSICAL SOCIETY, 14., 2015, Rio de Janeiro, RJ. **Proceedings...** [S.l.]: SBGf, 2015. 2

TAKAHASHI, H.; PADILHA, A. L.; SAWANT, H. S.; COSTA, J. E. R.; CECATTO, J. R.; GONZALEZ, W. D.; PAULA, E.; VITORELLO, I.; VELHO, H. C.; SILVA, J. D. S.; ABDU, M. A.; JR., O. M.; LAGO, A. D.; NARDIN, C. M. D.; SANTANNA, N.; ALVES, M. V.; SOUZA, J. R. The first landmark of the brazilian space weather program. In: BRAZILIAN DECIMETRIC ARRAY WORKSHOP, 2008, INPE. **Proceedings...** São José dos Campos: INPE, 2009, INPE. p. 155–160. Disponível em: <http://urlib.net/8JMKD3MGP8W/35LT3U8>. 20

TANENBAUM, A. S. et al. **Computer networks**. 4. ed. [S.l.: s.n.], 2003. 28

TAYLOR, R. N.; MEDVIDOVIC, N.; DASHOFY, E. M. **Software architecture**: foundations, theory, and practice. [S.l.]: Wiley Publishing, 2009. 60

THIO, N.; KARUNASEKERA, S. Automatic measurement of a qos metric for web service recommendation. In: AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE (ASWEC'05), 2005, Brisbane, Australian. **Procedings...** Whashington: IEEE, 2005. p. 202–211. 27

TOSIC, V.; MOORSEL, A. van; WONG, R. Quality of service middleware for web services: Achieved results and challenges for the future-panel. In: EDOC MIDDLEWARE FOR WEB SERVICES WORKSHOP (MWS'05), 2005. **Proceedings...** [S.l.], 2005. p. 19–23. 30

TRIENEKENS, J. J.; BOUMAN, J. J.; ZWAN, M. V. D. Specification of service level agreements: Problems, principles and practices. **Software Quality Journal**, Springer, v. 12, n. 1, p. 43–57, 2004. 32

APÊNDICE A: Documentação da implementação do sistema Arctic-FoxQoS

A.1 Requisitos

A listagem abaixo apresenta os requisitos arquiteturais e funcionais do Arctic-FoxQoS.

A.1.1 Requisitos Arquiteturais - RA

- **RA-01** - A arquitetura de referência deve permitir a integração de qualquer sistema de segmento solo e de seus componentes;
- **RA-02** - A arquitetura de referência deve permitir a implementação de gateway de comunicação para cada qualquer tipo de sistema de segmento solo e de seus componentes;
- **RA-03** - A arquitetura de referência deve permitir a implementação de um broker publish/subscribe (pub/sub) para a troca de mensagens entre qualquer tipo de sistema de segmento solo e de seus componentes;
- **RA-04** - A arquitetura de referência deve permitir a coleta de métricas de QoS dos sistemas de segmento solo e de seus componentes;
- **RA-05** - A arquitetura de referência deve permitir o processamento das métricas de QoS coletadas dos sistemas de segmento solo e de seus componentes;
- **RA-06** - A arquitetura de referência deve permitir o armazenamento das métricas de QoS coletadas dos sistemas de segmento solo e de seus componentes;
- **RA-07** - A arquitetura de referência deve permitir a disponibilização das métricas de QoS coletadas dos sistemas de segmento solo e de seus componentes;
- **RA-08** - A arquitetura de referência deve permitir o monitoramento das métricas coletadas dos sistemas de segmento solo e de seus componentes independente da tecnologia utilizada;
- **RA-09** - A arquitetura de referência deve permitir o reuso entre suas camadas de referência;

- **RA-10** - A arquitetura de referência deve permitir a interoperabilidade entre suas camadas de referência;
- **RA-11** - A arquitetura de referência deve permitir a criação de um banco de dados local para armazenar todas a métricas de QoS coletadas dos sistemas de segmento solo e de seus componentes;
- **RA-12** - A arquitetura de referência deve possuir um banco de dados central na nuvem para armazenar todas a métricas de QoS coletadas dos sistemas de segmento solo e de seus componentes;
- **RA-13** - A arquitetura de referência deve permitir um sistema de gerenciamento de acordo de nível de serviço;
- **RA-14** - A arquitetura de referência deve possuir uma camada de segurança para gerenciar as autenticações nos sistemas alvo;

A.1.2 Requisitos Funcionais do ArcticFoxQoS - RFA

- **RFA-01** - O ArcticFoxQoS deve permitir o gerenciamento dos atributos de QoS;
- **RFA-02** - O ArcticFoxQoS deve permitir o gerenciamento das métricas de QoS;
- **RFA-03** - O ArcticFoxQoS deve permitir o gerenciamento das regras do acordo de nível de serviço;
- **RFA-04** - O ArcticFoxQoS deve permitir o gerenciamento das consequências de violação de um SLA;
- **RFA-05** - O ArcticFoxQoS deve permitir o gerenciamento dos sistemas de segmento solo e de seus componentes que serão monitorados;
- **RFA-06** - O ArcticFoxQoS deve permitir o gerenciamento dos tipos de recursos de um dispositivo;
- **RFA-07** - O ArcticFoxQoS deve permitir o gerenciamento de categorização das métricas;
- **RFA-08** - O ArcticFoxQoS deve permitir o gerenciamento dos parâmetros das métricas cadastradas;

- **RFA-09** - O ArcticFoxQoS deve permitir o gerenciamento dos status dos contratos cadastrados;
- **RFA-10** - O ArcticFoxQoS deve permitir o gerenciamento das dependências que as métricas cadastradas possuem;
- **RFA-11** - O ArcticFoxQoS deve permitir o gerenciamento do acordo de nível de serviço entre provedores e consumidores;
- **RFA-12** - O ArcticFoxQoS deve permitir a criação de um banco de dados para armazenar dados das métricas coletadas localmente;
- **RFA-13** - O ArcticFoxQoS deve permitir a criação de um ou mais agentes coletores embarcados dentro do nó onde estará localizado o sistema alvo.
- **RFA-14** - O ArcticFoxQoS deve permitir a criação de um ou mais agentes publicadores embarcados dentro do.
- **RFA-15** - O ArcticFoxQoS deve permitir a criação de um ou mais agentes coletores fora do nó onde estará localizado o sistema alvo.
- **RFA-16** - O ArcticFoxQoS deve permitir a coleta de métricas de sistemas alvo que não fazem parte do domínio de monitoramento.
- **RFA-17** - O agente coletor deve ser responsável por armazenar as métricas coletadas no banco de dados local;
- **RFA-18** - O agente coletor deve colocar a métrica coletada em um serviço de troca de mensagens localizado em um servidor na nuvem;
- **RFA-19** - O ArcticFoxQoS deve permitir a criação de agentes assinantes responsáveis por colocar a métrica coletada em um serviço de troca de mensagens localizado em um servidor na nuvem;
- **RFA-20** - O ArcticFoxQoS deve permitir a criação de um serviço de troca de mensagens na nuvem;
- **RFA-21** - O ArcticFoxQoS deve permitir a criação de um ou mais agentes consumidores responsáveis em obter as métricas coletadas, inseridas na fila de mensagens.
- **RFA-22** - Os agentes consumidores devem ser os responsáveis em armazenar as métricas coletadas no banco de dados central, Metric Repository.

- **RFA-23** - O ArcticFoxQoS deve permitir a criação de um ou mais agentes assinantes a fim de armazenar as métricas consumidas da fila em uma base de dados central na nuvem;

- **RFA-24** - Os agentes consumidores assinantes ser os responsáveis em armazenar as métricas coletadas no banco de dados central, Metric Repository.

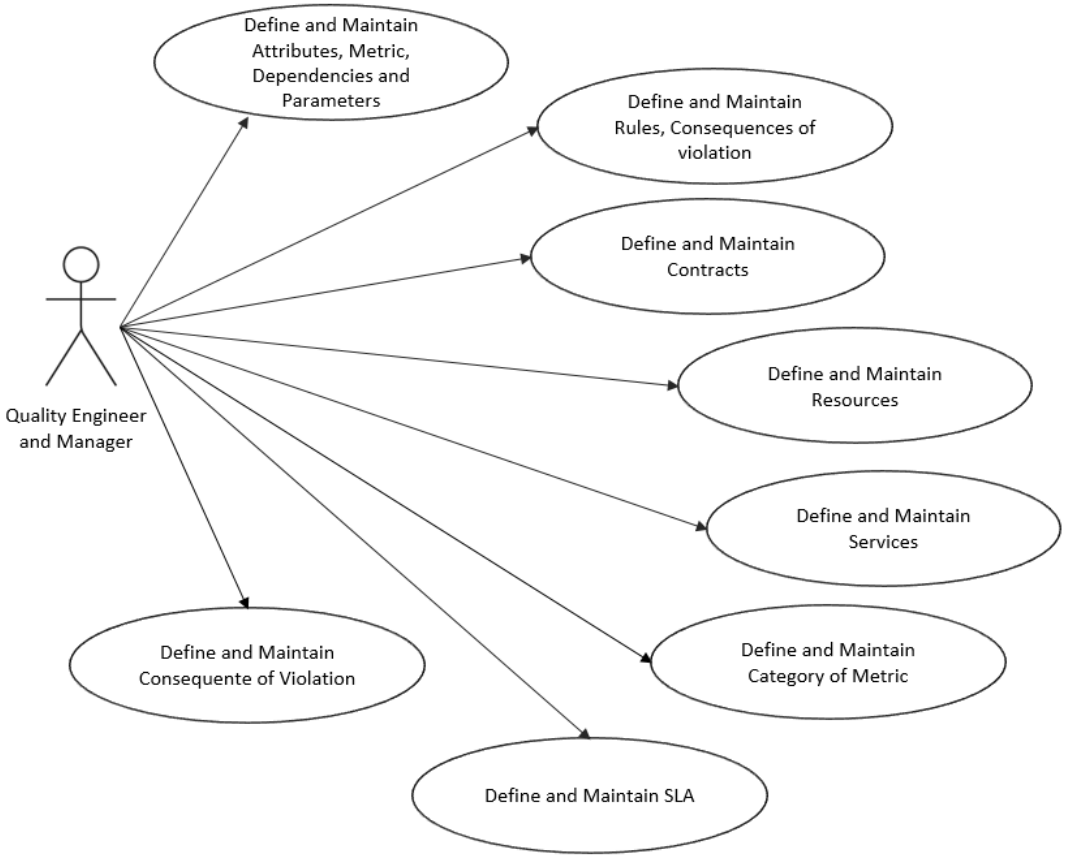
- **RFA-25** - O ArcticFoxQoS deve possuir um ou mais agentes entregadores *(Deliver)* que quando solicitado o mesmo busca o dado coletado no banco de dados entrega ao um cliente;

- **RFA-26** - O ArcticFoxQoS deve disponibilizar as métricas coletadas dos sistemas alvo em um formato de arquivo de intercâmbio de dados que seja legível por máquinas;

A.2 Casos de uso do ArcticFoxQoS

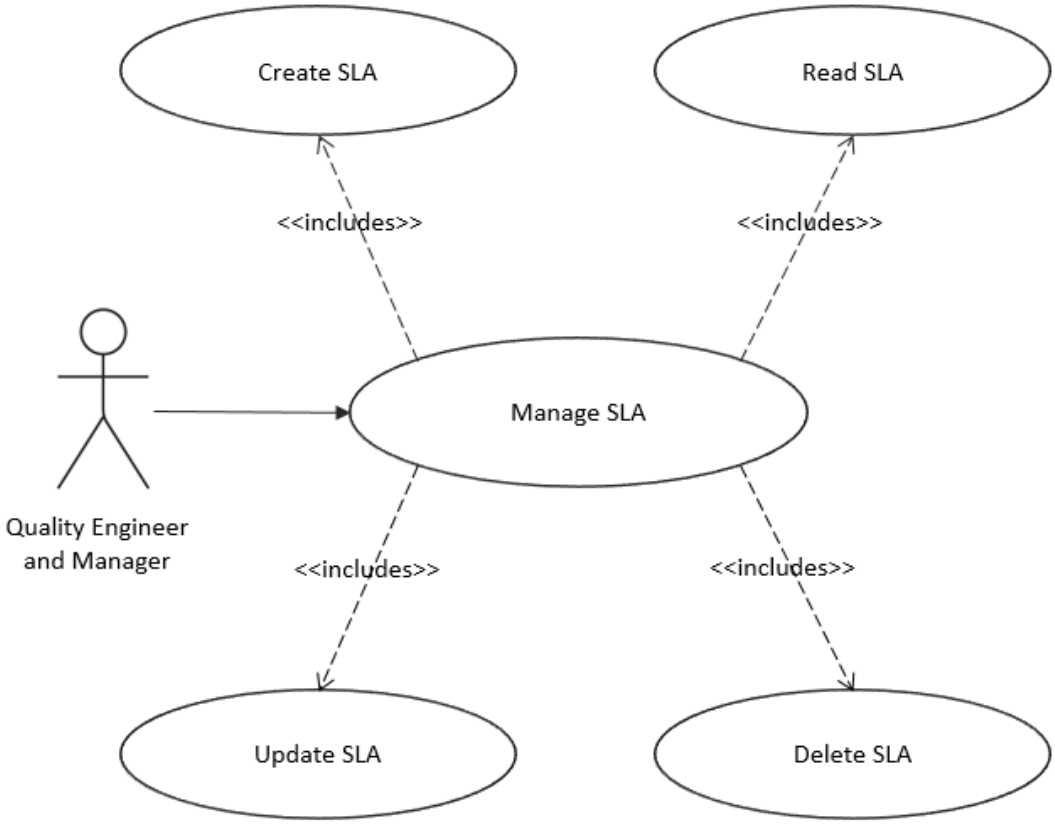
A.2.1 Gerenciamento da Qualidade

Figura A.1 - Caso de uso do engenheiro da qualidade.



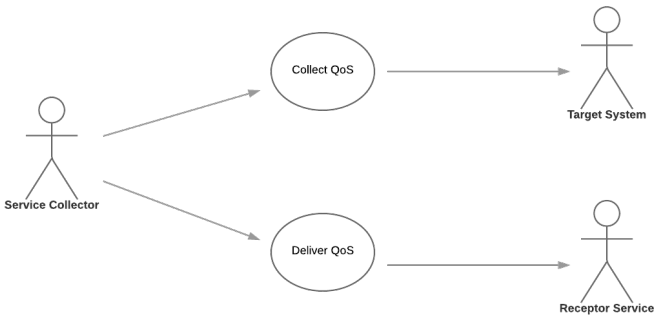
A.2.2 Gerenciamento de SLA

Figura A.2 - Caso de uso de um gerenciamento de SLA.



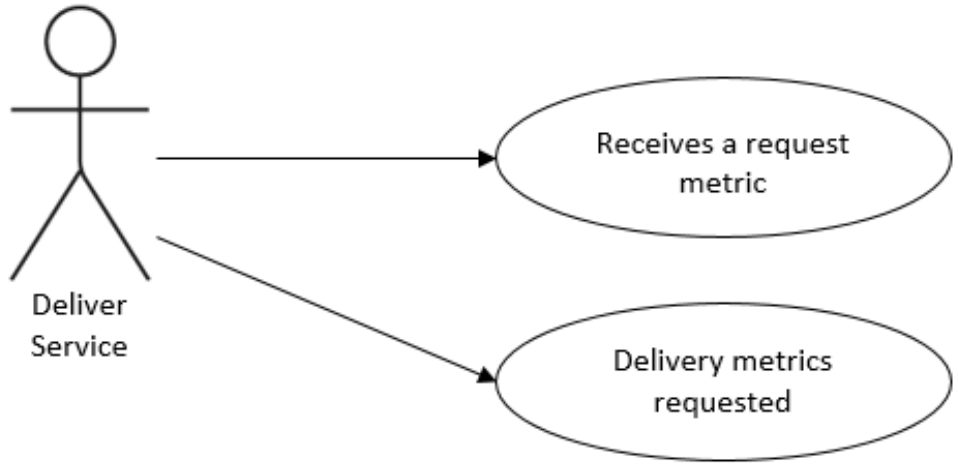
A.2.3 Agentes de Coleta

Figura A.3 - Caso de uso de um agente de coleta.



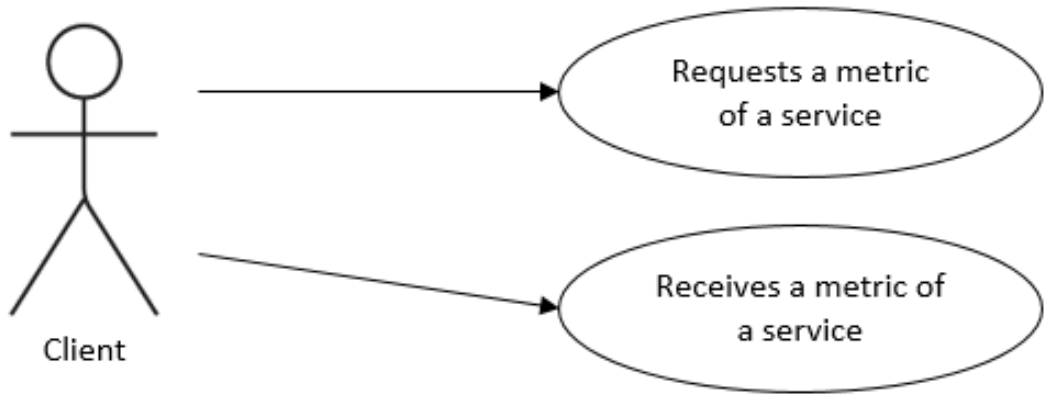
A.2.4 Deliver

Figura A.4 - Caso de uso de um serviço de entrega.



A.2.5 Cliente

Figura A.5 - Caso de uso de um cliente requisitando e recebendo os dados solicitados.



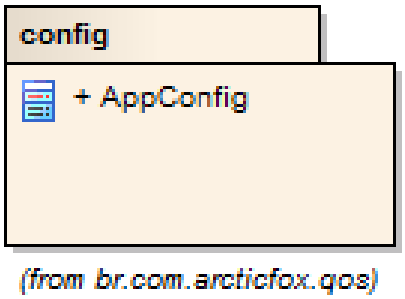
A.3 Diagramas de Pacotes do ArcticFoxQoS

A.3.1 Pacote Principal

Figura A.6 - Estrutura principal dos pacotes do ArcticFoxQoS.

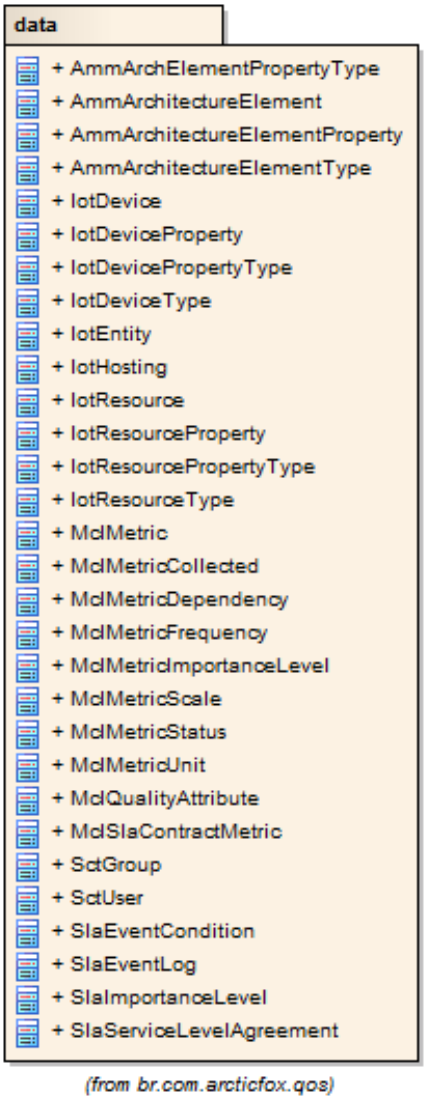
A.3.2 Pacote Config

Figura A.7 - Estrutura do pacote config e sua classe de configuração AppConfig.



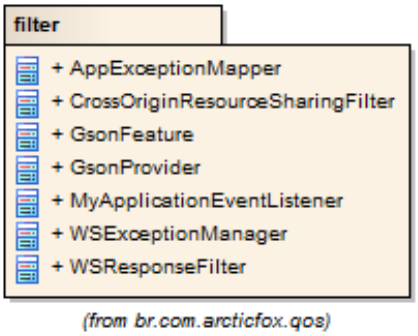
A.3.3 Pacote Data

Figura A.8 - Estrutura do pacote data e suas respectivas classes.



A.3.4 Pacote Filter

Figura A.9 - Estrutura do pacote filter e suas respectivas classes.

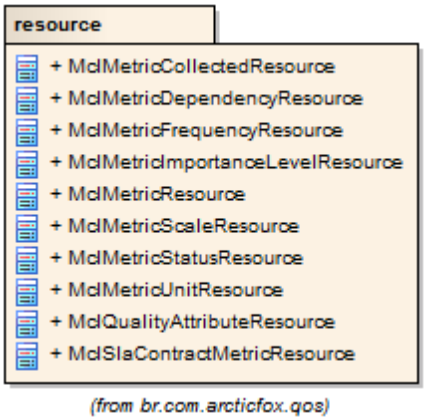


A.3.5 Pacote Services

Figura A.10 - Estrutura do pacote service e suas respectivas classes.

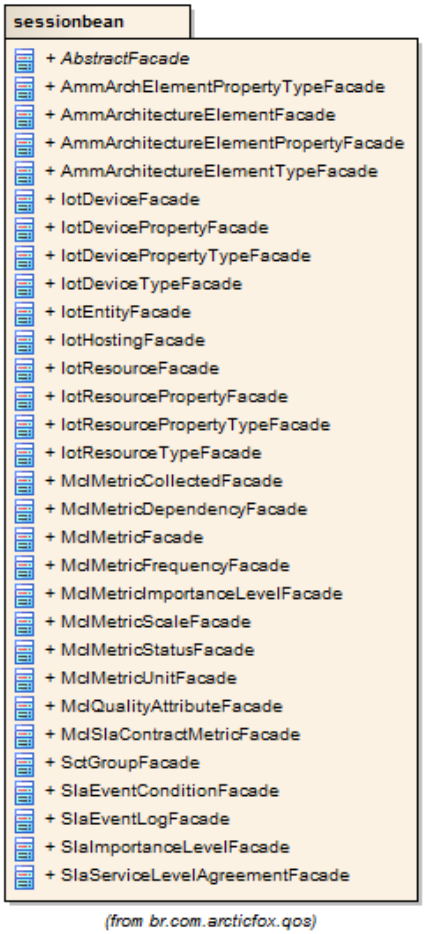
A.3.6 Pacote Resource

Figura A.11 - Estrutura do pacote resource e suas respectivas classes.



A.3.7 Pacote SessionBean

Figura A.12 - Estrutura do pacote sessionbean e suas respectivas classes.



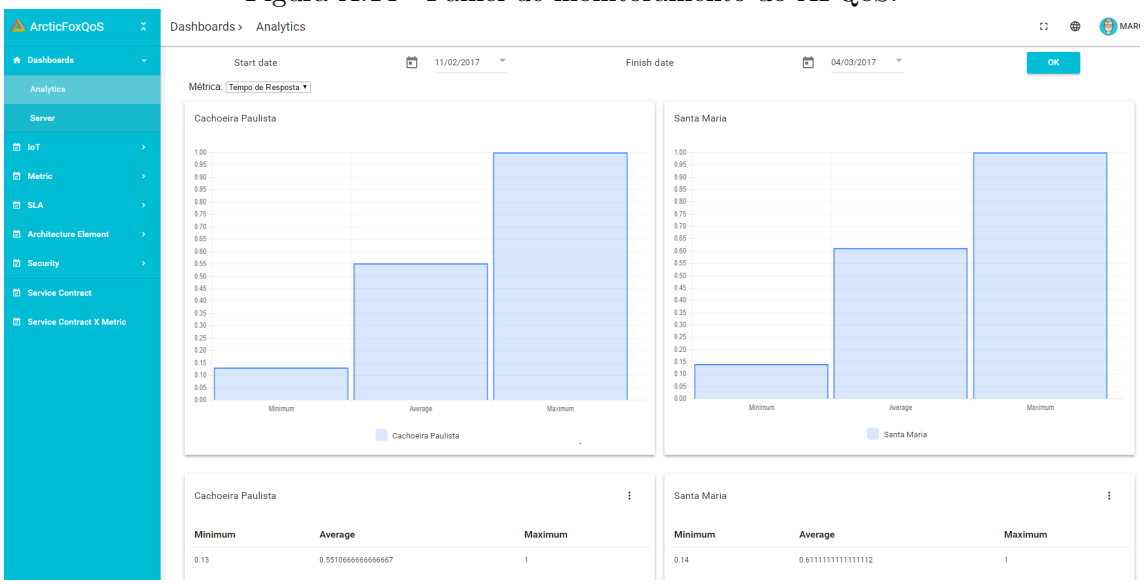
A.3.8 Pacote SessionBeanLocal

Figura A.13 - Estrutura do pacote sessionbeanlocal e suas respectivas interfaces.

A.4 Telas do sistema de cadastro

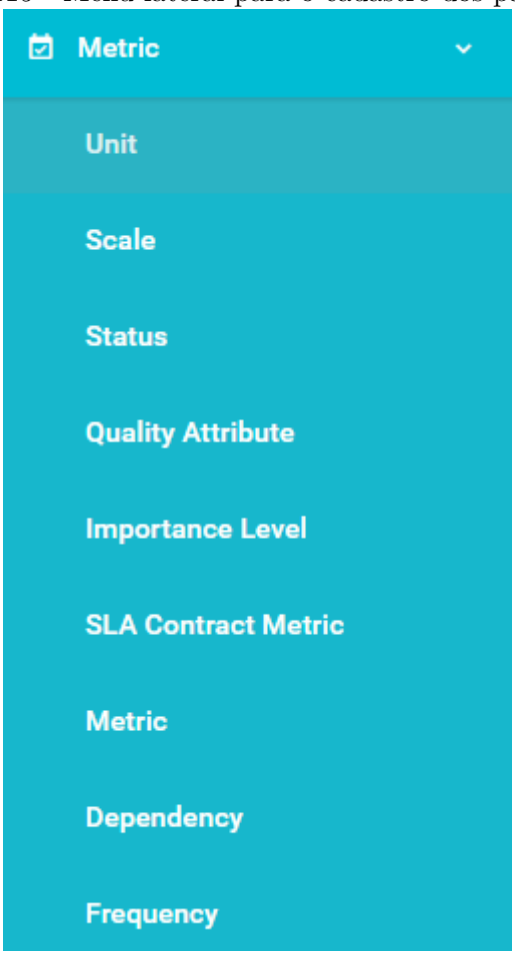
A Figura A.14 apresenta o painel de monitoramento *(Dashboard)*. Nele é possível selecionar o tipo da métrica, período de coleta e a exibição dos dados através de um gráfico. Além disso uma tabela exibindo os valores máximo, média, mínimo coletados no período pré selecionado para cada sítio monitorado.

Figura A.14 - Painel de monitoramento do AFQoS.



A Figura A.15 apresenta o menu que possibilita o cadastro dos parâmetros necessários para a composição da métrica a ser coletada.

Figura A.15 - Menu lateral para o cadastro dos parâmetros.



Nas figuras seguintes é apresentada as telas de cadastro dos parâmetros necessários para definição da métrica.

Figura A.16 - Cadastro do atributo de qualidade.

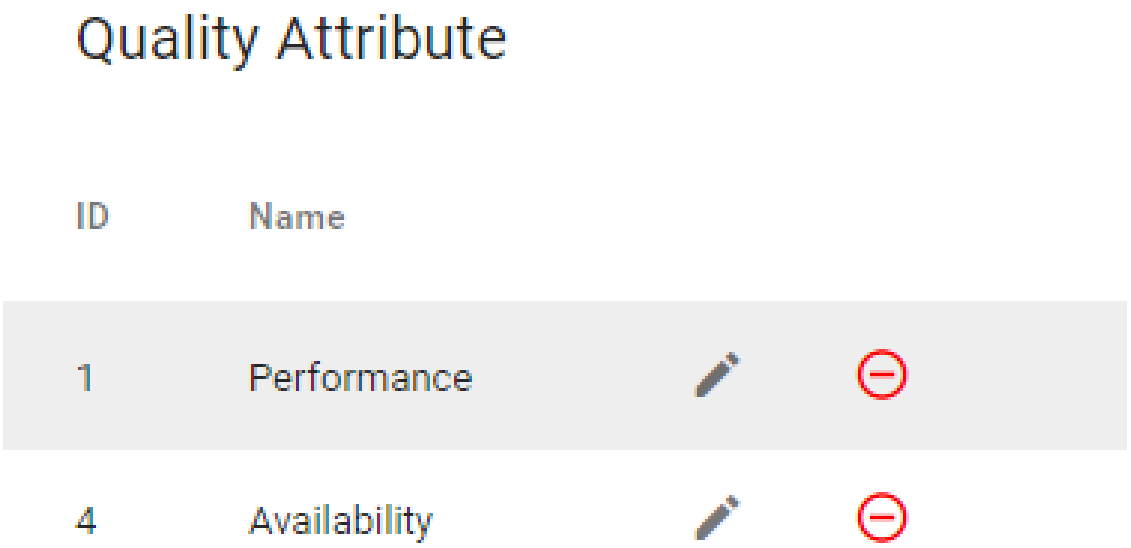


Figura A.17 - Cadastro das métricas dependentes.

Dependency

Name	Dependency Expression	Note
Simultaneous Connections	Availability = totalErrors/NumberOfConnections	Higher availability is achieved with more simultaneous connections
No dependency	No dependency	No dependency

Figura A.18 - Cadastro da unidade de medida da métrica.

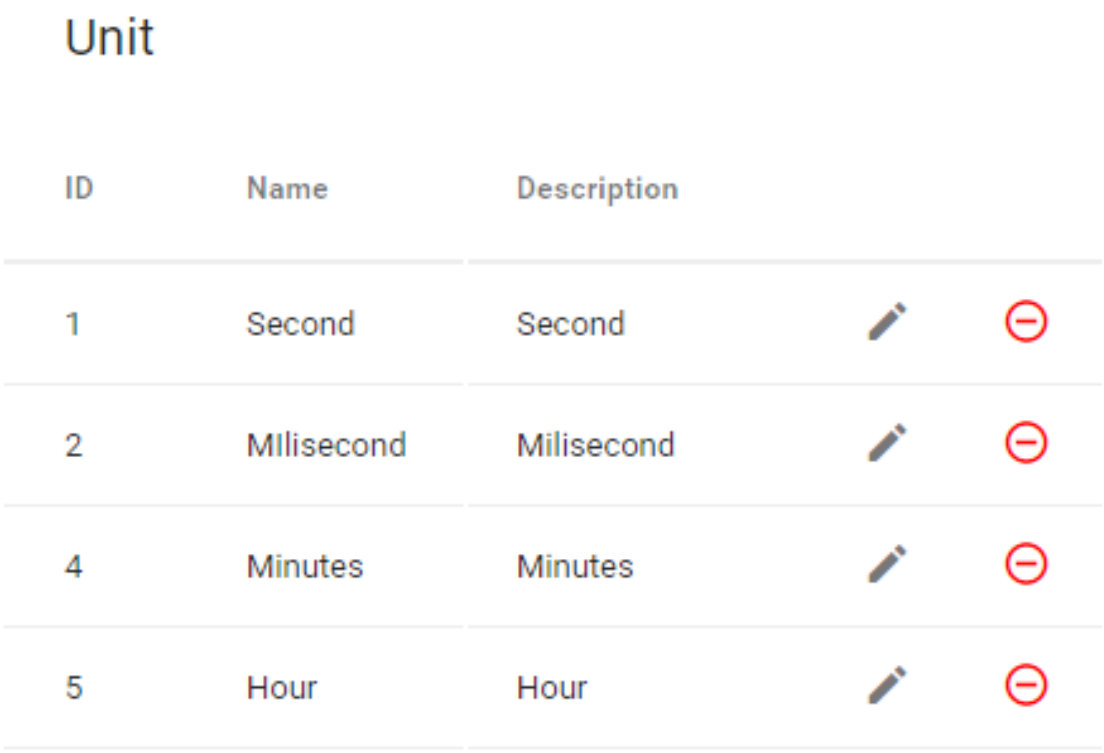


Figura A.19 - Cadastro dos tipos de escala da métrica.

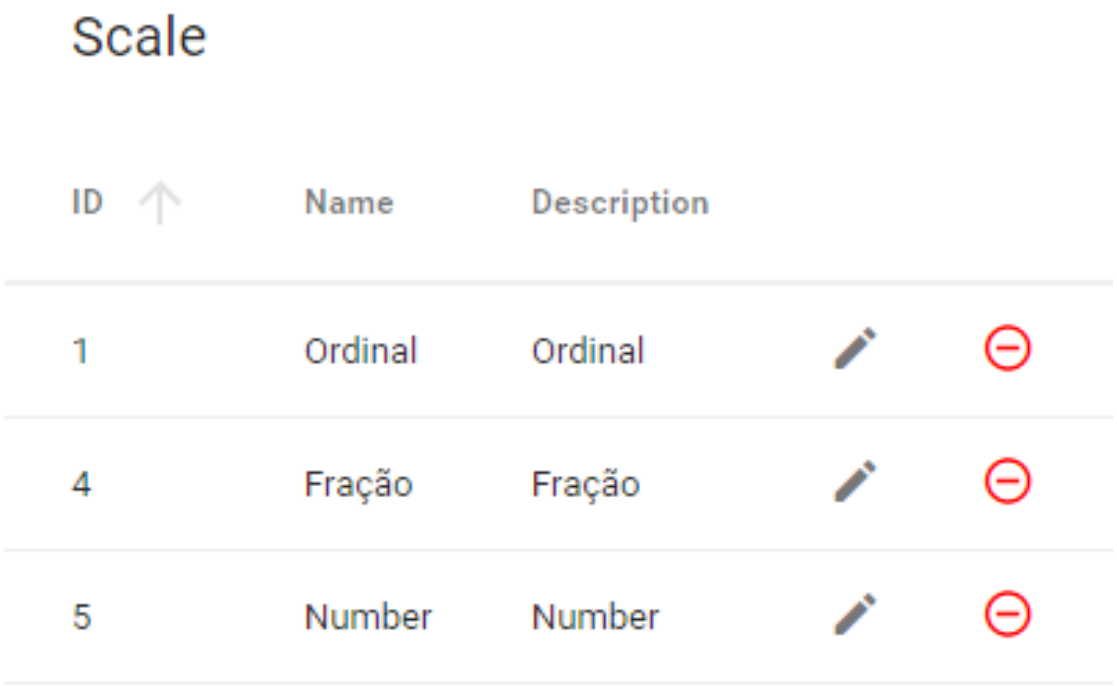
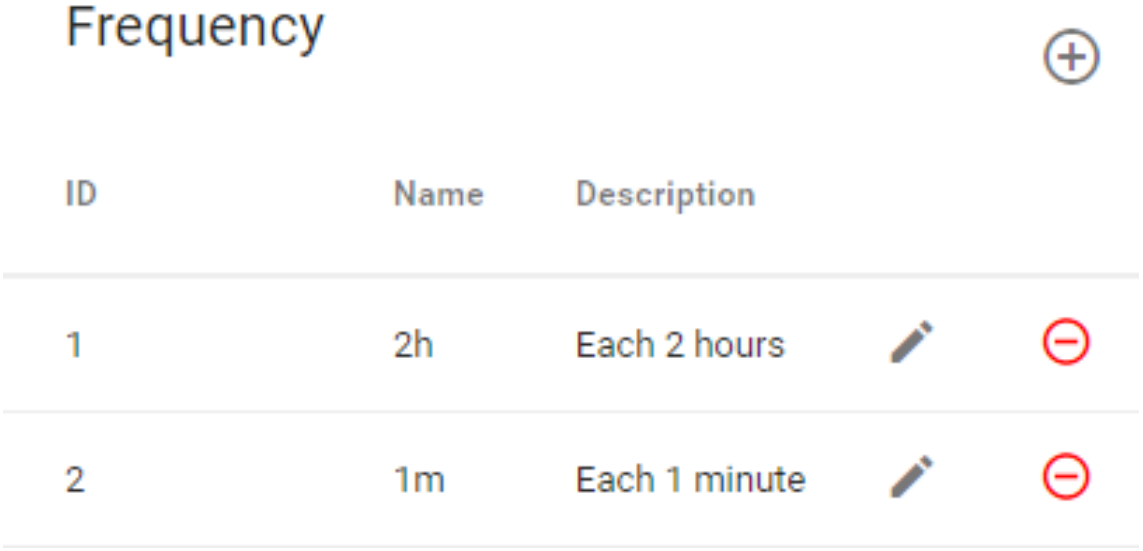


Figura A.20 - Definição da parâmetro da frequência de coleta da métrica.



A Figura A.21 apresenta os acordos de níveis de serviços definidos para cada métrica. Assim, todas as métricas coletadas possuem um SLA previamente cadastrado.

Figura A.21 - Acordo de nível de servico por métricas.

Service Contract X Metric

ID	Service Contract	Metric	Level	Status	Frequency	Default Value
1	Contrato de serviço GTex no Servidor HP 01 - Cachoeira Paulista	ResponseTime	Highly Important	Active	Each 2 hours	1.5
2	Contrato de serviço GTex no Servidor Dell 01 - STA Maria	ResponseTime	Highly Important	Active	Each 2 hours	1.8

A.5 Código fonte do CollectorQoSAgentWS

A figura A.22 e A.23 mostra os códigos responsáveis pela cálculo e a resposta do Collector.

Figura A.22 - Código fonte do cálculo do atributo de qualidade *Performance*.

```
public List<SlaMetricCollected> performanceCalculator(String url) {
    int ncsr = 0;
    int ncsrnot = 0;
    double ut = 5000.0;
    double start, finish = 0;
    start = System.currentTimeMillis();
    for (int i = 0; i < 100000; i++) {
        try {
            String result = ConnectionWS.connectionWSAPP(url);
            if (result != null) {
                (new Thread()).start();
                ncsr++;
            } else {
                ncsrnot++;
            }
        } catch (Exception ex) {
            log.error(ex);
        }
    }
    double tp_srv = (ncsr / ut);
    log.info("Number of completed service requests = " + ncsr);
    log.info("Number of not completed service requests = " + ncsrnot);
    double responseTime = (finish - start);
    createPerformance(tp_srv, responseTime);
    List<SlaMetricCollected> list = new ArrayList<>();
    SlaMetricCollected slaMetricCollected = new SlaMetricCollected();
    slaMetricCollected.setCreateDate(
            new DateTime(DateTimeZone.UTC).toDate());
    slaMetricCollected.setSlaMetricFk(new SlaMetric(1L));
    list.get(0).setValue(tp_srv);
    list.get(1).setValue(responseTime);
    return list;
}
```

Figura A.23 - Código fonte do método responsável por entregar a métrica coletada.

```
@GET
@Path("/performance")
@Produces(MediaType.APPLICATION_JSON)
public Response collectorPerformance() {
    return Response.status(Response.Status.OK)
            .entity(performanceCalculator(URIClient.URL_CONTEXT))
            .build();
}
```

A.6 Código fonte do ReceptorService

Figura A.24 - Código fonte do método responsável por enfileirar a métrica coletada.

```
@Path("/receptor")
@GET
public Response receptorService() {
    try {
        String metricAttributeValueJson = ConnectionWS
                .connectionGeneral("http://localhost:46856/CollectorQoSAgent/performance");
        System.out.println(metricAttributeValueJson);
        ObjectMapper mapper = new ObjectMapper();

        List<MetricValue> enums = mapper.readValue(metricAttributeValueJson,
                new TypeReference<List<MetricValue>>() {
        });

        for (MetricValue aEnum : enums) {
            System.out.println(aEnum.getValue());
        }
        sendMessage(enums);
        ConnectionWS.connectionGeneral("http://localhost:46856/CollectorQoSAgent/garbage");
        return Response.ok().entity(metricAttributeValueJson).build();
    } catch (Exception ex) {
        log.error(ex);
    }
    return Response.serverError().build();
}

public void sendMessage(List<MetricValue> metricAttributeValues) {
    try {
        context.createProducer().send(qosMetricQueue, (Serializable) metricAttributeValues);
    } catch (Exception exc) {
        log.error(exc);
    }
}
```

A.7 Código fonte do SubscriberQoSAgentWS

Figura A.25 - Código fonte do agente assinante.

```
public class SubscriberQoSAgentWS {

    String USER = env("ACTIVEMQ_USER", "admin");
    String PWD = env("ACTIVEMQ_PASSWORD", "password");
    String HOST = env("ACTIVEMQ_HOST", "localhost");
    int PORT = Integer.parseInt(env("ACTIVEMQ_PORT", "1883"));

    public Response listener(String[] params) throws URISyntaxException {
        final String DESTINATION = arg(params, 0, "/topic/event");
        MQTT mqtt = new MQTT();
        mqtt.setHost(HOST, PORT);
        mqtt.setUserName(USER);
        mqtt.setPassword(PWD);

        final CallbackConnection connection = mqtt.callbackConnection();
        connection.listener(new org.fusesource.mqtt.client.Listener() {
            long count = 0;
            long start = System.currentTimeMillis();

            public void onConnected() {
            }

            public void onDisconnected() {
            }

            public void onFailure(Throwable value) {
                value.printStackTrace();
                System.exit(-2);
            }

            public void onPublish(UTF8Buffer topic, Buffer msg, Runnable ack) {
                String body = msg.utf8().toString();
                if ("SHUTDOWN".equals(body)) {
                    long diff = System.currentTimeMillis() - start;
                    System.out.println(String.format("Received %d in %.2f seconds", count, (1.0 * diff / 1000.0)));
                    connection.disconnect(new Callback<Void>() {
                        @Override
                        public void onSuccess(Void value) {
                            System.exit(0);
                        }

                        @Override
                        public void onFailure(Throwable value) {
                            value.printStackTrace();
                            System.exit(-2);
                        }
```

A.8 API do Sistema de Entregas

Para que cliente possa acessar os serviços do AFQoS a fim de obter todas as métricas coletadas, acordos de nível de serviço e diversas outras metainformações, uma API de entrega baseada em serviços RESTful foi desenvolvida. Dessa forma garantir a interoperabilidade entre sistemas distintos. A listagem abaixo apresenta os principais serviços disponíveis.

- Buscar métricas coletadas por data e ContractMetric. Exemplo do código fonte do serviço e o formato de saída nas Figuras A.26 e A.27 respectiva-

mente;

getMetricCollectedByServiceContractAndPeriod(contractMetricId, initialDate, String finalDate)

- Buscar métricas coletadas por data e Architecture Element.

getMetricCollectedByAEAndPeriod(architectureElementId, initialDate, String finalDate)

- Buscar todos as métricas ativas por ContractMetric;

getActiveMetricBySla(contractMetricId, status)

- Buscar métrica por nível de importância;

getMetricsByImportanceLevel(importanceLevelId)

- Buscar métrica por nível de importância e ServiceContract;

getMetricsByImportanceLevel(importanceLevelId, contractMetricId)

- Buscar todas as métricas de um determinando serviço de contrato por frequência;

getMetricsByContractMetricAndFrequency(contractMetricId,frequencyId)

- Buscar todas as métricas coletadas por SLA;

getMetricsBySla(slaId)

- Buscar qual o valor *default* de uma métrica do serviço provedor. (Ex: Qual a disponibilidade do GTex WS?)

getDefaultValueByContractMetric(slaId, metricId)

- Qual a frequência de coleta da métrica X por Resource?

getMetricFrequencyByResource(resourceId)

- Buscar todos os contratos de serviço de um Resource

getServiceContractByResource(resourceId)

- Buscar todos os contratos de serviço de um Resource por status

getServiceContractByResourceAndStatus(resourceId, status)

- Buscar contrato de serviço entre Resource e AE

getServiceContractByResourceAndAE(resourceId, aeId)

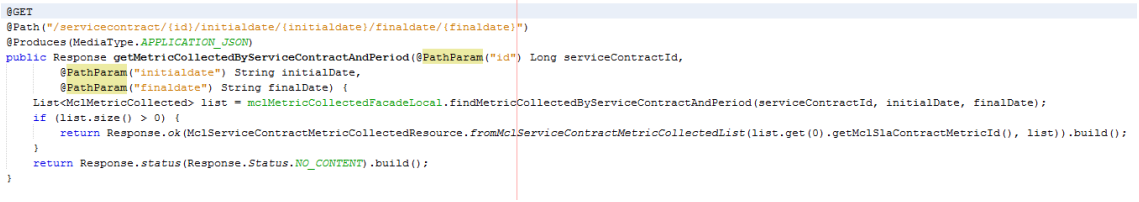
- Buscar todos os contratos de serviço de um Resource por nivel de importância

getServiceContractByResourceAndImportanceLevel(resourceId, importanceLevelId)

- Buscar métrica coletada de um AE por tipo de Métrica. (Qual o tempo de resposta do serviço GTex WS)

getMetricCollectedByAEAndMetric(aeId, metricId)

Figura A.26 - Código fonte do serviço de entrega das métricas coletadas por contrato.



```
@GET
@Path("/servicecontract/{id}/initialdate/{initialdate}/finaldate/{finaldate}")
@Produces(MediaType.APPLICATION_JSON)
public Response getMetricCollectedByServiceContractAndPeriod(@PathParam("id") Long serviceContractId,
        @PathParam("initialdate") String initialDate,
        @PathParam("finaldate") String finalDate) {
    List<MclMetricCollected> list = mclMetricCollectedFacadeLocal.findMetricCollectedByServiceContractAndPeriod(serviceContractId, initialDate, finalDate);
    if (list.size() > 0) {
        return Response.ok(MclServiceContractMetricCollectedResource.fromMclServiceContractMetricCollectedList(list.get(0).getMclSlaContractMetricId(), list)).build();
    }
    return Response.status(Response.Status.NO_CONTENT).build();
}
```

Figura A.27 - Resposta do serviço *getMetricCollectedByServiceContractAndPeriod* no formato Json.


```
{
-"slaContractMetricResource": {
    "id": 1,
    "creationDate": "2017-01-01 17:00:00",
   -"mclMetricId": { ... }
   -"mclMetricImportanceLevelId": { ... }
   -"slaServiceContractId": {
        "id": 1,
      -"slaServiceLevelAgreemntId": {
            "id": 1,
            "name": "Space Weather Service Level Agreement GOES Application Service",
            "description": "This Space Weather GOES Application Service Service Level Agreement
            "creationDate": "2017-01-01 16:00:00",
            "expirationDate": "2017-12-31 23:59:59",
            "updateDate": null,
          -"mclMetricStatusId": { ... }
        },
      -"iotResourceId": { ... }
      -"ammArchitectureElementId": { ... }
      -"slaImportanceLevelId": { ... }
       "name": "Contrato de serviço GTex no Servidor HP 01",
       "creationDate": "2017-02-14 09:31:51",
       "status": 1
    }
},
-"mclMetricCollectedResource": [Array[60]
   -0:  {
        "id": 399,
        "value": 0.41,
        "collectedTimestamp": "2017-02-12 23:55:20"
    },
   -1:  {
        "id": 383,
        "value": 0.82,
        "collectedTimestamp": "2017-02-12 23:55:20"
    },
   -2:  {
        "id": 386,
        "value": 0.74,
        "collectedTimestamp": "2017-02-12 23:55:20"
    },
```

A.9 Código fonte do Service Client do *Front-End*

Figura A.28 - Código fonte da camada de serviço do Sistema de Gerenciamento do AFQoS.

```
function addItem(event) {
    $mdDialog
        .show({
          clickOutsideToClose: true,
              controller: 'ServiceContractItemController as vm',
              focusOnOpen: false,
              targetEvent: event,
              templateUrl: 'app/service-contract/add.item.dialog.tmpl.html',
          locals: {
            title: vm.title,
            content: {},
                                service: 'UNIT',
            type: 'POST'
          },
        })//.then(activate());
        .finally(function() {
            activate();
        });
}

function editContent(event, content) {
    event.stopPropagation();
    $mdDialog.show({
      clickOutsideToClose: true,
      controller: 'ServiceContractItemController as vm',
      focusOnOpen: false,
      targetEvent: event,
      templateUrl: 'app/service-contract/add.item.dialog.tmpl.html',
      locals: {
        title: vm.title,
        content: content,
                        service: 'UNIT',
        type: 'UPDATE'
      },
    }).then(console.log('getAll'));
}
```

APÊNDICE B: Diagramas de Entidade e Relacionamento relacionados à base de dados do ArcticFoxQoS

B.1 Diagrama Entidade Relacionamento

O diagrama de entidade e relacionamento reflete à implementação da arquitetura de referência do ArcticFoxQoS para a realização do monitoramento da QoS. O diagrama está dividido em 5 grupos: Iot, Metric, Architecture Element, Security e Sla.

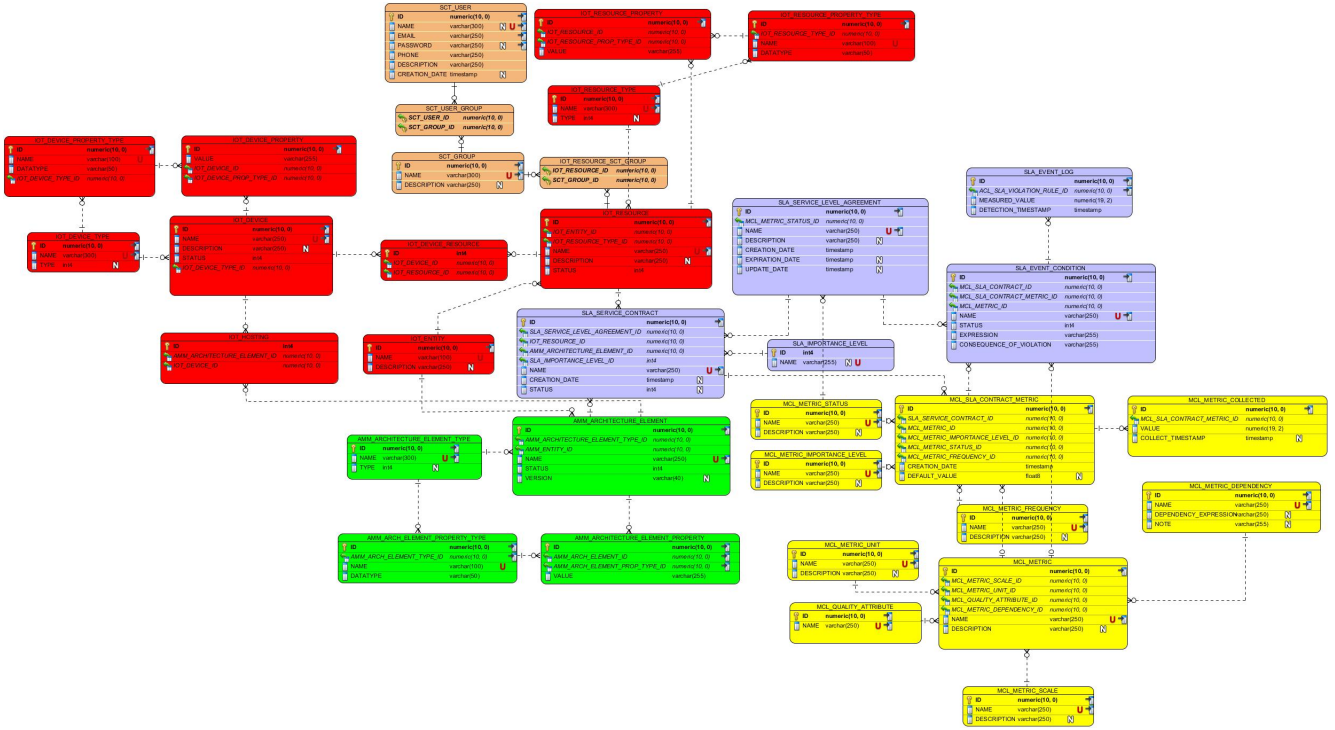


Figura B.1 - Diagrama Entidade Relacionamento do AFQoS.

B.1.1 DER - IoT

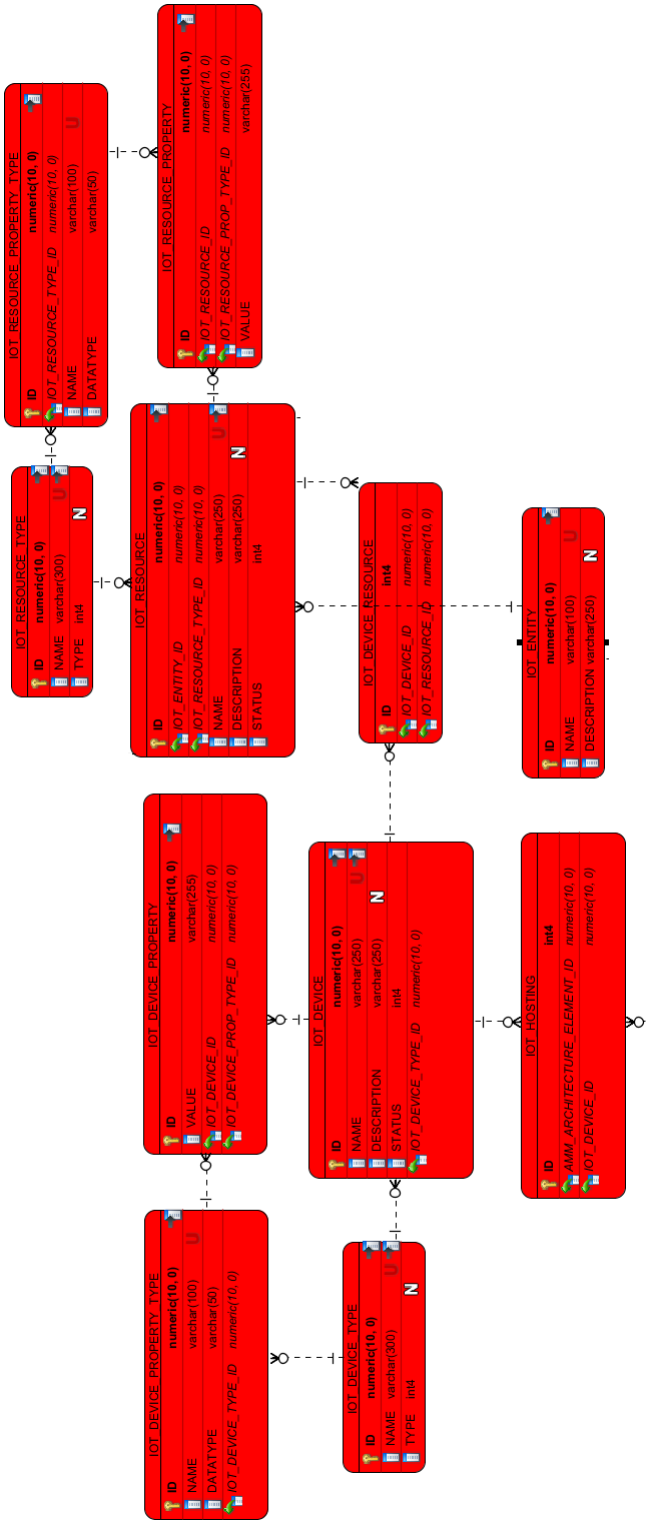


Figura B.2 - Diagrama Entidade Relacionamento do AFQoS - Parte Iot.

B.1.2 DER - Metric

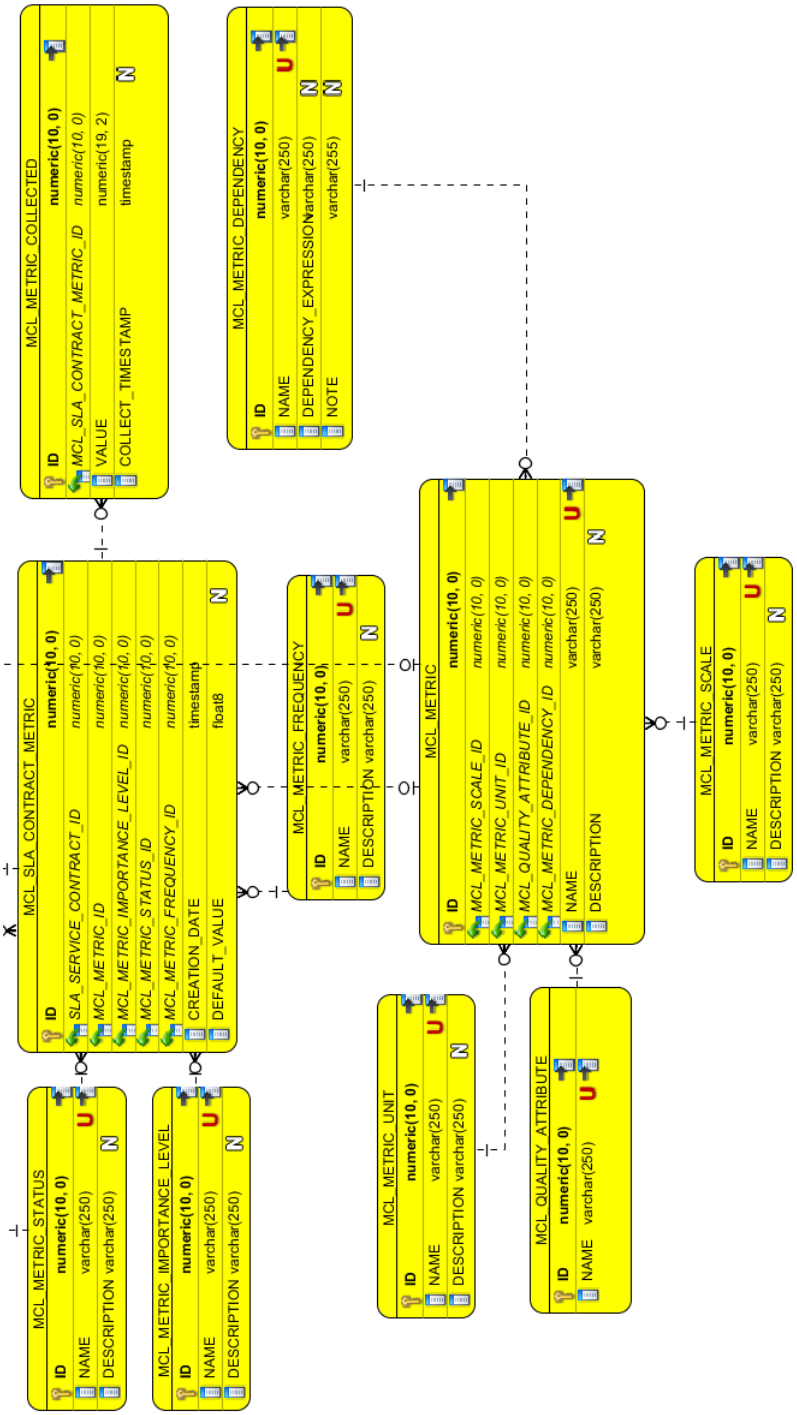


Figura B.3 - Diagrama Entidade Relacionamento do AFQoS - Parte Metric.

B.1.3 DER - Architecture Element

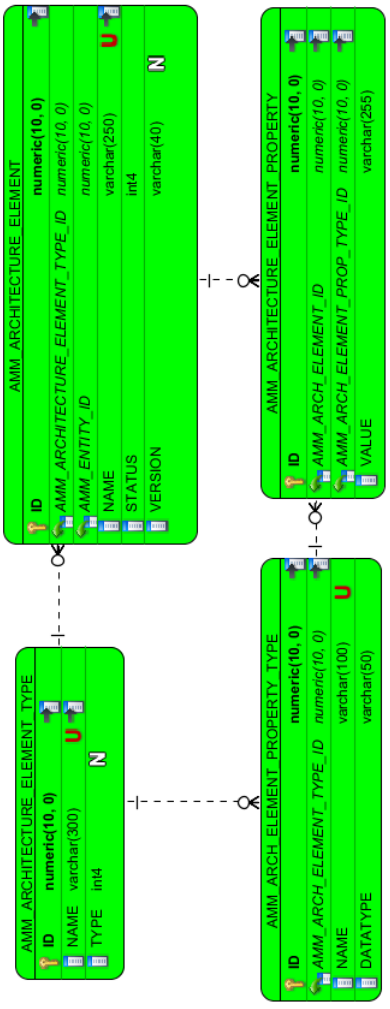


Figura B.4 - Diagrama Entidade Relacionamento do AFQoS - Parte Architecture Element.

B.1.4 DER - Security

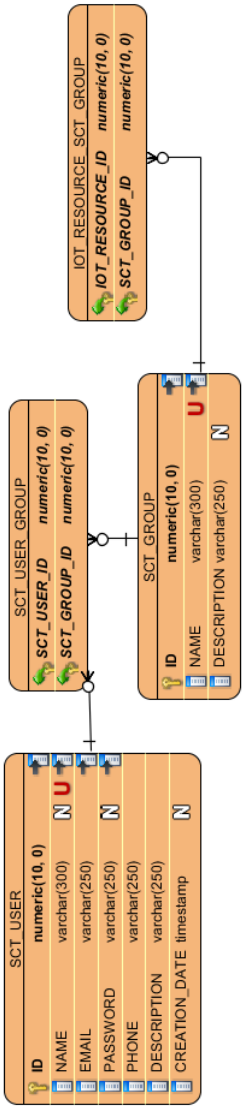


Figura B.5 - Diagrama Entidade Relacionamento do AFQoS - Parte Security.

B.1.5 DER - SLA

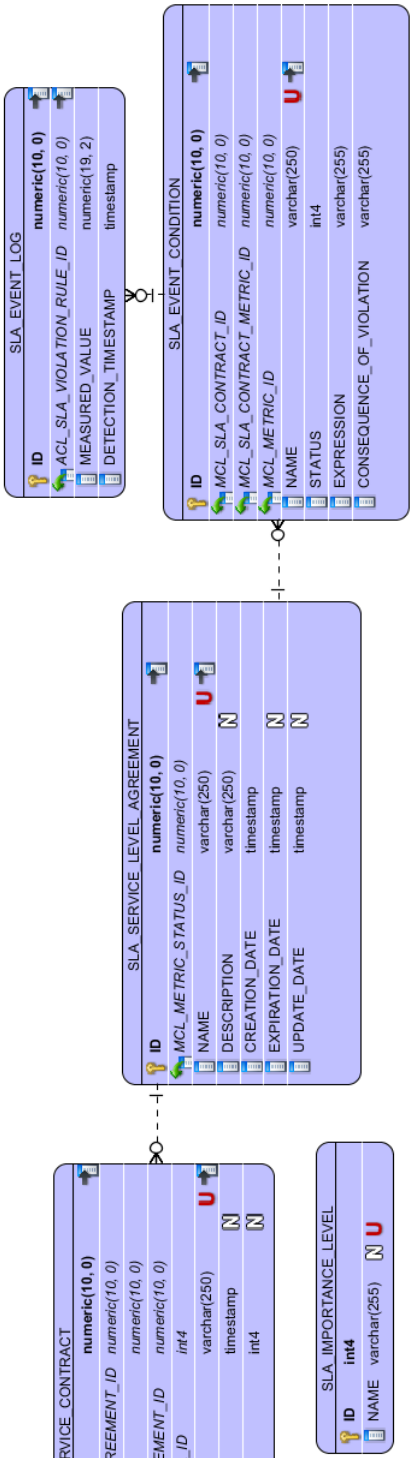


Figura B.6 - Diagrama Entidade Relacionamento do AFQoS - Parte Sla.