

**Universidade Federal de Uberlândia**  
**Faculdade de Computação**

**Uma Arquitetura com Suporte a Módulos Dinâmicos  
para WebLab no Domínio de Redes de Computadores**

**Autor: Daniel Vieira de Souza**

**Orientador: Prof. Dr. Luís Fernando Faina**

**Dissertação de Mestrado** apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciência da Computação. Área de Concentração: **Redes de Computadores**.

**Fevereiro de 2011**  
**Uberlândia, MG - Brasil**

Dados Internacionais de Catalogação na Publicação (CIP)  
Sistema de Bibliotecas da UFU

---

S729a Souza, Daniel Vieira de, 1977 -  
Uma arquitetura com suporte a módulos dinâmicos para WebLab no  
domínio de redes de computadores [manuscrito] / Daniel Vieira de Souza. - 2011  
90 f. : il.

Orientador: Luís Fernando Faina.

Dissertação (mestrado) - Universidade Federal de Uberlândia,  
Programa de Pós-Graduação em Ciência da Computação.  
Inclui bibliografia.

1. Redes de computadores - Protocolos. - Teses. I. Faina, Luís Fernando.
- II. Universidade Federal de Uberlândia. Programa de Pós-Graduação  
em Ciência da Computação. III. Título.

CDU: 681.3.02

---

**Universidade Federal de Uberlândia - UFU**  
**Faculdade de Computação - FACOM**

# **Uma Arquitetura com Suporte a Módulos Dinâmicos para WebLab no Domínio de Redes de Computadores**

**Autor: Daniel Vieira de Souza**

**Orientador: Prof. Dr. Luís Fernando Faina**

**Dissertação de Mestrado** apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciência da Computação. Área de Concentração: **Redes de Computadores**.

## **Banca Examinadora**

---

Prof. Dr. Luís Fernando Faina – FACOM/UFU (orientador)

---

Prof. Dr. Cairo Lúcio Nascimento Júnior – ITA

---

Prof. Dr. Pedro Frosi Rosa – FACOM/UFU

---

Prof. Dr. Renan Gonçalves Cattelan – FACOM/UFU

**Fevereiro de 2011**  
**Uberlândia, MG - Brasil**

# Resumo

Souza, D.V., “Uma Arquitetura com Suporte a Módulos Dinâmicos para WebLab no Domínio de Redes de Computadores”. Dissertação de Mestrado - FACOM/UFU, Uberlândia, MG. Fevereiro 2011.

Este trabalho apresenta uma arquitetura com suporte a módulos dinâmicos para laboratórios de acesso remoto (WebLab), no domínio de redes de computadores. A arquitetura segue o paradigma da Arquitetura Orientada a Serviços (*Service Oriented Architecture - SOA*). Os experimentos são disponibilizados como módulos, seguindo a especificação OSGi (*Open Services Gateway Initiative*). Os serviços de cada módulo de experimento são publicados automaticamente como serviços Web, seguindo os princípios REST (*Representational State Transfer*). Requisitos funcionais para WebLabs de propósitos gerais, bem como para o domínio de redes de computadores, são contemplados nesta proposta a fim de permitir a criação de um WebLab mais robusto que atenda à demanda de uso por parte dos alunos e instituições de ensino e de forma escalável. Por meio da criação de um novo experimento, que fez uso da implementação de referência da arquitetura proposta, possibilitou-se demonstrar a facilidade na concepção e disponibilização dos experimentos em forma de módulos, além de contribuir para o WebLab com mais um experimento, que permite a configuração de VLANs (*Virtual Local Area Network*).

Palavras-chave: WebLab, REST, Web Service, OSGi, VLAN.

# Abstract

Souza, D.V., “An Architecture with Support for Dynamic Modules to WebLab in the Domain of Computer Networks”. Master Thesis - FACOM/UFU, Uberlândia, MG. February 2011.

This work presents an architecture that supports dynamic modules for Remote Access laboratories (WebLab) in the domain of computer networks. The architecture follows the paradigm of Service Oriented Architecture (SOA). The experiments are available as modules, following the OSGi specification (Open Services Gateway Initiative). The services of each module of the experiment are automatically published as Web services, following the REST (Representational State Transfer) principles. Functional requirements for general-purpose WebLabs, as well for the domain of computer networks, are covered in this proposal in order to allow the creation of more robust WebLab that meets the demand of use on the part of students and educational institutions, and in a scalable manner. Through the creation of a new experiment, which made use of the reference implementation of the proposed architecture, it was allowed to demonstrate the facility in designing and making available the experiments in the form of modules, besides contributing to the WebLab with one more experiment that allows setting up VLANs (Virtual Local Area Network).

Keywords: WebLab, REST, Web Service, OSGi, VLAN.

*Para meu filho Gabriel e meu sobrinho Neto (in memoriam)*

*“Agradeço todas as dificuldades que enfrentei; não fosse por elas, eu não teria saído do lugar. As facilidades nos impedem de caminhar. Mesmo as críticas nos auxiliam muito.”*

Chico Xavier

# Agradecimentos

A Deus por ter me iluminado e permitido a conclusão deste trabalho.

Ao meu orientador Prof. Dr. Luís Fernando Faina, por estar sempre disposto a ajudar e pela orientação firme que propiciou o desenvolvimento deste trabalho, meus sinceros agradecimentos.

Ao Prof. Paulo Rodolfo da Silva Leite Coelho, por sua participação nas reuniões e ajuda nos artigos.

A minha mãe Maria e meu pai Walter pelo apoio incondicional nos bons e nos maus momentos.

A minha querida esposa Tahis, por me apoiar e aceitar os momentos de ausência, principalmente na fase final desta dissertação. Ao meu filho Gabriel que com seu sorriso puro e cativante, me deu forças para continuar em frente motivado e contagiado pela sua alegria.

Aos meus familiares, minha irmã Raquel, minha sobrinha Mayara, meu querido sobrinho Neto.

Aos companheiros de projetos, Paulo Eduardo, Célio Domingues, Ricardo Bortolatto, Fabíola Bento, João Euripedes, pelas conversas, pelas informações e conhecimentos compartilhados.

Ao amigo Lúcio Agostinho pela disposição em ajudar sempre que precisei de informações sobre o projeto.

A Faculdade de Computação (FACOM) da Universidade Federal de Uberlândia (UFU) pela infraestrutura e suporte aos alunos do curso.

Aos amigos da secretaria, Erisvaldo e André pela dedicação em ajudar os estudantes.

A Sankhya Tecnologia em Sistemas pela liberação parcial, que, sem ela, não seria possível ter realizado este trabalho. Meus sinceros agradecimentos aos colegas de trabalho, Manoel Prudêncio Menezes, que sem seu apoio incondicional esse trabalho não seria possível, Cláudio Gualberto pelas conversas sobre arquitetura de *software* e a flexibilidade nos horários de trabalho, Wender Oliveira, pela disposição em sempre querer ajudar, Rodrigo e Jorlano pelas conversas sobre a experiência deles no mestrado e Marcos Paulo Bonatti pela ajuda com as imagens e ícones.

# Sumário

|                                                         |             |
|---------------------------------------------------------|-------------|
| <b>Resumo</b>                                           | <b>iv</b>   |
| <b>Abstract</b>                                         | <b>v</b>    |
| <b>Lista de Figuras</b>                                 | <b>x</b>    |
| <b>Lista de Tabelas</b>                                 | <b>xii</b>  |
| <b>Glossário</b>                                        | <b>xiii</b> |
| <b>1 Introdução</b>                                     | <b>1</b>    |
| 1.1 Motivações . . . . .                                | 1           |
| 1.2 Contexto do Trabalho . . . . .                      | 2           |
| 1.3 Objetivos do Trabalho . . . . .                     | 2           |
| 1.4 Organização da Dissertação . . . . .                | 3           |
| <b>2 Laboratórios de Acesso Remoto</b>                  | <b>4</b>    |
| 2.1 Arquitetura do WebLab-Deusto . . . . .              | 4           |
| 2.1.1 Experimentos Contemplados . . . . .               | 6           |
| 2.1.2 Análise da Arquitetura do WebLab-Deusto . . . . . | 6           |
| 2.2 Arquitetura do WebLab-SIT . . . . .                 | 7           |
| 2.2.1 Experimentos Contemplados . . . . .               | 8           |
| 2.2.2 Análise da Arquitetura do WebLab-SIT . . . . .    | 10          |
| 2.3 Arquitetura do iLab . . . . .                       | 11          |
| 2.3.1 Experimentos Contemplados . . . . .               | 13          |
| 2.3.2 Análise da Arquitetura do iLab . . . . .          | 14          |
| 2.4 Arquitetura do WebLab-DACSE . . . . .               | 15          |
| 2.4.1 Experimentos Contemplados . . . . .               | 17          |
| 2.4.2 Análise da Arquitetura do WebLab-DACSE . . . . .  | 17          |
| 2.5 Arquitetura do NetLab WebLab . . . . .              | 18          |
| 2.5.1 Experimentos Contemplados . . . . .               | 20          |
| 2.5.2 Análise da Arquitetura do NetLab WebLab . . . . . | 21          |
| 2.6 Requisitos Funcionais para WebLabs . . . . .        | 22          |
| 2.7 Considerações Finais . . . . .                      | 24          |
| <b>3 Arquitetura Proposta para WebLab</b>               | <b>25</b>   |
| 3.1 Modelo de referência para WebLab . . . . .          | 25          |
| 3.1.1 Sessão de Acesso . . . . .                        | 25          |
| 3.1.2 Sessão de Interação . . . . .                     | 26          |

|          |                                                                |           |
|----------|----------------------------------------------------------------|-----------|
| 3.1.3    | Sessão de Comunicação . . . . .                                | 27        |
| 3.2      | Arquitetura com Suporte a Módulos Dinâmicos . . . . .          | 27        |
| 3.3      | Descrição dos Componentes da Arquitetura Proposta . . . . .    | 29        |
| 3.3.1    | Componente <i>Dispatcher</i> . . . . .                         | 29        |
| 3.3.2    | Componente <i>ServiceManager</i> . . . . .                     | 31        |
| 3.3.3    | Componente <i>Core</i> . . . . .                               | 33        |
| 3.3.4    | Framework OSGi . . . . .                                       | 35        |
| 3.4      | Interação Entre os Componentes . . . . .                       | 39        |
| 3.4.1    | Inicialização dos Componentes . . . . .                        | 39        |
| 3.4.2    | Disponibilização e Remoção de Módulos . . . . .                | 41        |
| 3.4.3    | Execução de Serviço . . . . .                                  | 43        |
| 3.5      | Considerações Finais . . . . .                                 | 44        |
| <b>4</b> | <b>Implementação e Resultados</b>                              | <b>45</b> |
| 4.1      | Aspectos de Implementação da Arquitetura Proposta . . . . .    | 45        |
| 4.1.1    | Componente <i>Dispatcher</i> . . . . .                         | 47        |
| 4.1.2    | Componente <i>ServiceManager</i> . . . . .                     | 48        |
| 4.1.3    | Componente <i>Core</i> . . . . .                               | 50        |
| 4.2      | Cenário para Validação da Arquitetura Proposta . . . . .       | 52        |
| 4.2.1    | Infraestrutura do Laboratório . . . . .                        | 52        |
| 4.2.2    | Cadastro do Experimento de Configuração de VLAN . . . . .      | 55        |
| 4.2.3    | Reserva do Experimento . . . . .                               | 56        |
| 4.2.4    | Execução do Experimento . . . . .                              | 57        |
| 4.3      | Cenário #1 Criação de VLAN . . . . .                           | 63        |
| 4.4      | Cenário #2 Configuração de 3 VLANs Inter-conectadas . . . . .  | 64        |
| 4.5      | Cenário #3 Configuração de 3 VLANs sem Inter-conexão . . . . . | 67        |
| 4.6      | Considerações Finais . . . . .                                 | 69        |
| <b>5</b> | <b>Conclusão e Trabalhos Futuros</b>                           | <b>71</b> |
| 5.1      | Retrospectiva das Contribuições . . . . .                      | 71        |
| 5.2      | Avaliação das Contribuições . . . . .                          | 71        |
| 5.3      | Trabalhos Futuros . . . . .                                    | 73        |
|          | <b>Referências Bibliográficas</b>                              | <b>75</b> |

# Lista de Figuras

|      |                                                                                  |    |
|------|----------------------------------------------------------------------------------|----|
| 2.1  | Visão Geral da Arquitetura do WebLab-Deusto [10]. . . . .                        | 5  |
| 2.2  | Servidores e Protocolos Utilizados para Comunicação [10]. . . . .                | 6  |
| 2.3  | Experimento WebLab-FPGA [7]. . . . .                                             | 7  |
| 2.4  | Visão Geral da Arquitetura [8]. . . . .                                          | 9  |
| 2.5  | Aplicação Cliente Executando um Experimento Virtual [8]. . . . .                 | 9  |
| 2.6  | Experimento Remoto do Tipo Real em Execução [8]. . . . .                         | 10 |
| 2.7  | Visão Geral da Execução de um Experimento em Lote [8]. . . . .                   | 11 |
| 2.8  | Visão Geral da Arquitetura do iLab [12]. . . . .                                 | 12 |
| 2.9  | Arquitetura Compartilhada do iLab. [6]. . . . .                                  | 12 |
| 2.10 | Estrutura para Execução de Experimentos em Lote [6]. . . . .                     | 13 |
| 2.11 | Arquitetura para Experimentos com Interação do Aluno [6]. . . . .                | 14 |
| 2.12 | Visão Geral da Arquitetura Genérica [9]. . . . .                                 | 16 |
| 2.13 | Estrutura dos Componentes do Servidor de Aplicação [9]. . . . .                  | 17 |
| 2.14 | Diagrama para a Criação de Laboratórios [13]. . . . .                            | 18 |
| 2.15 | Visão Geral da Arquitetura do NetLab WebLab [4]. . . . .                         | 19 |
| 2.16 | Composição de Serviços do NetLab WebLab [4]. . . . .                             | 20 |
| 2.17 | Aplicação Cliente do Experimento de Configuração de NIC [2]. . . . .             | 21 |
| 2.18 | Experimento de Configuração de Rotas [15]. . . . .                               | 22 |
| 3.1  | Diagrama de Pacotes Representando os Serviços de Cada Sessão [2]. . . . .        | 26 |
| 3.2  | Diagrama Bloco dos Componentes que Constituem a Arquitetura Proposta. . . . .    | 28 |
| 3.3  | Arquitetura Proposta com Suporte a Módulos Dinâmicos para WebLabs. . . . .       | 28 |
| 3.4  | Diagrama Pacotes com as Classes dos Componentes. . . . .                         | 29 |
| 3.5  | Camadas do <i>Framework</i> OSGi [18]. . . . .                                   | 35 |
| 3.6  | <i>Class Loader Delegation model</i> [18]. . . . .                               | 37 |
| 3.7  | <i>Class Space</i> [18]. . . . .                                                 | 37 |
| 3.8  | Diagrama de Estados do <i>framework</i> [18]. . . . .                            | 38 |
| 3.9  | Inicialização dos Componentes da Arquitetura Proposta. . . . .                   | 40 |
| 3.10 | Sequência de Disponibilização de Módulo . . . . .                                | 42 |
| 3.11 | Sequência de Remoção de Módulo . . . . .                                         | 43 |
| 3.12 | Sequência de Execução de Serviços . . . . .                                      | 44 |
| 4.1  | Divisão dos Pacotes da Implementação. . . . .                                    | 46 |
| 4.2  | Diagrama das Interfaces e Classes de Implementações dos Componentes. . . . .     | 46 |
| 4.3  | Diagrama de Classes Separados por Pacotes. . . . .                               | 47 |
| 4.4  | Atendimento de Requisições REST pelo <i>HttpServiceDispatcher</i> . . . . .      | 48 |
| 4.5  | Interação entre o <i>HttpServiceDispatcher</i> e <i>ServiceManager</i> . . . . . | 49 |
| 4.6  | Interação entre o <i>Core</i> e <i>ServiceManager</i> . . . . .                  | 50 |
| 4.7  | Interação entre o <i>Core</i> e <i>Framework</i> OSGi. . . . .                   | 51 |

|      |                                                                               |    |
|------|-------------------------------------------------------------------------------|----|
| 4.8  | Infra-estrutura dos <i>Hosts</i> e <i>Switchs</i> do NetLab. . . . .          | 53 |
| 4.9  | Portal Administrativo do NetLab parte do Projeto GigaBOT. . . . .             | 54 |
| 4.10 | Marcação no Cadastro de Experimento de VLAN. . . . .                          | 56 |
| 4.11 | Marcação no Cadastro Recurso. . . . .                                         | 56 |
| 4.12 | Listagem de Experimentos do Portal Netlab. . . . .                            | 57 |
| 4.13 | Visão Geral das Interações Relacionadas a Execução do Experimento. . . . .    | 58 |
| 4.14 | Aplicação Cliente do Experimento. . . . .                                     | 61 |
| 4.15 | Aplicação Cliente Após Carregadas as VLANs. . . . .                           | 62 |
| 4.16 | Representação gráfica e um <i>Host</i> e suas NICs. . . . .                   | 62 |
| 4.17 | Painel de Mapa da Rede. . . . .                                               | 63 |
| 4.18 | Painel de Criação de VLANs. . . . .                                           | 64 |
| 4.19 | Painel de Inclusão de Portas a uma VLANs. . . . .                             | 65 |
| 4.20 | Distribuição dos Hosts por VLANs. . . . .                                     | 66 |
| 4.21 | Interligações Entre as Redes. . . . .                                         | 67 |
| 4.22 | Execução do Teste de Conectividade de Gaia para Poseidon. . . . .             | 67 |
| 4.23 | Resultado do Teste de Conectividade ente Gaia e Poseidon Cenário #2. . . . .  | 68 |
| 4.24 | <i>Zeus</i> Fora da Rede Lab B. . . . .                                       | 69 |
| 4.25 | Mapa da Rede Cenário #3. . . . .                                              | 70 |
| 4.26 | Resultado do Teste de Conectividade ente Gaia e Poseidon. Cenário #3. . . . . | 70 |

# Lista de Tabelas

|     |                                                                      |    |
|-----|----------------------------------------------------------------------|----|
| 2.1 | Requisitos Gerais por WebLab . . . . .                               | 23 |
| 2.2 | Requisitos Específicos do Domínio de Redes de Computadores . . . . . | 23 |
| 3.1 | Exemplo de ambientes de execuções [18] . . . . .                     | 36 |
| 4.1 | Configurações dos Computadores do NetLab. . . . .                    | 53 |
| 4.2 | Configurações dos Switchs. . . . .                                   | 53 |
| 4.3 | Características das Redes e Hosts do Cenário #2. . . . .             | 66 |
| 4.4 | Características das Redes e Hosts do Cenário #3. . . . .             | 68 |

# Glossário

|      |                                             |
|------|---------------------------------------------|
| AJAX | <i>Asynchronous Javascript And XML</i>      |
| FPGA | <i>Field Programmable Gate Array</i>        |
| GPIB | <i>General Porpouse Instrumentation Bus</i> |
| HTML | <i>HyperText Markup Language</i>            |
| HTTP | <i>HyperText Trasnfer Protocol</i>          |
| IP   | <i>Internet Protocol</i>                    |
| JDK  | <i>Java Development Kit</i>                 |
| JDOM | <i>Java Document Object Model</i>           |
| JNLP | <i>Java Network Launching Protocol</i>      |
| JWS  | <i>Java Web Start</i>                       |
| MVC  | <i>Model View Controller</i>                |
| NIC  | <i>Network Interface Card</i>               |
| OSGi | <i>Open Services Gateway Initiative</i>     |
| REST | <i>Representational State Transfer</i>      |
| RMI  | <i>Remote Method Invocation</i>             |
| RPC  | <i>Remote Procedure Call</i>                |
| SNMP | <i>Simple Network Management Protocol</i>   |
| SOA  | <i>Service Oriented Architecture</i>        |
| SOAP | <i>Simple Object Access Protocol</i>        |
| SSH  | <i>Secure Shell</i>                         |
| TCP  | <i>Transmission Control Protocol</i>        |
| UML  | <i>Unified Modeling Language</i>            |
| URI  | <i>Uniform Resource identifier</i>          |

|      |                                          |
|------|------------------------------------------|
| URL  | <i>Uniform Resource Locator</i>          |
| USB  | <i>Universal Serial Bus</i>              |
| VLAN | <i>Virtual Local Area Network</i>        |
| VRML | <i>Virtual Reality Modeling Language</i> |
| XML  | <i>Extensible Markup Language</i>        |
| XWL  | <i>eXensible WebLab</i>                  |

# Capítulo 1

## Introdução

Este capítulo apresenta as principais motivações para a concepção de uma arquitetura com suporte a módulos dinâmicos para laboratórios de acesso remoto e contextualiza a avaliação desta no domínio de redes de computadores.

### 1.1 Motivações

Laboratórios de Acesso Remoto, também conhecidos como WebLabs, oferecem por meio da rede de conexão, acesso a infraestruturas de *hardware* e *software*, permitindo a execução de experimentos que manipulam com frequência equipamentos reais. O uso de WebLabs por instituições de ensino superior tem mostrado bons resultados, pois maximizam o uso dos equipamentos e possibilitam estender as aulas práticas [1]. Os alunos não ficam limitados aos laboratórios e horários das instituições para pôr em prática o que aprendem nas aulas teóricas.

Para que possam ser utilizados e difundidos nas instituições de ensino, os WebLabs devem atender a um conjunto de requisitos que propiciem aos alunos interagir com os equipamentos dos laboratórios. Esses requisitos podem variar conforme o domínio do laboratório, por isso é apresentado um conjunto de requisitos (Seção 2.6) para a criação de laboratórios tidos como gerais e específicos.

Entre os requisitos funcionais típicos de WebLabs [2, 3], destacam-se: a) disponibilidade de experimentos bem como a disponibilidade do próprio laboratório (7 dias por semana e 24 horas por dia); b) utilização de padrões de desenvolvimento baseados em serviços; c) interoperabilidade por meio da utilização de protocolos não bloqueados por *firewall*; d) padronização para a criação dos experimentos; e) dinamicidade e adaptabilidade dos experimentos, possibilitando ao usuário a utilização de topologias de redes dinâmicas e adaptáveis aos experimentos usados; f) independência de um experimento em relação a outros experimentos para garantir que nenhum experimento interfira no funcionamento do laboratório.

Embora vários laboratórios sejam disponibilizados a cada dia na Internet, a abrangência e a forma como estes e outros requisitos funcionais são contemplados variam e, assim, constituem espaço para novos estudos, bem como para a extensão de outros. Nesse contexto, este trabalho propõe e implementa uma arquitetura que atende aos requisitos destacados, mas acrescenta mecanismos que facilitam enormemente a introdução de novos experimentos.

Nesse sentido, este trabalho descreve uma arquitetura que utiliza conceitos de OSGi (*Open Services Gateway Initiative*), para disponibilizar os experimentos em forma de módulos dinâmicos, atendendo aos requisitos disponibilidade, padrões de desenvolvimento e padronização na criação dos experimentos. Requisições REST (*Representational State Transfer*) entre a aplicação cliente e servidor, atendendo aos requisitos de interoperabilidade e desempenho na comunicação. Por fim, validou-se a

arquitetura implementada por meio de experimentos de configuração de VLANs (*Virtual Local Area Networks*).

## 1.2 Contexto do Trabalho

O WebLab descrito nesse trabalho, o NetLab, é um projeto da Faculdade de Computação - FACOM da Universidade Federal de Uberlândia - UFU. É um laboratório de acesso remoto para o domínio de redes de computadores, sendo inicialmente apresentado por Farias [2] e Rocha [4]. O NetLab integra o projeto GigaBOT - Laboratórios de Acesso Remoto Sobre Redes Avançadas [5] desenvolvido em conjunto com o Centro de Pesquisas Renato Archer - CenPRA e a Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas - FEEC Unicamp. Além dessas duas instituições, ainda fazem parte do projeto: Universidade Federal do Rio de Janeiro - UFRJ; Pontifícia Universidade Católica do Rio Grande do Sul - PUC/RS; e Instituto Tecnológico de Aeronáutica - ITA.

O NetLab segue o modelo conceitual apresentado pelo projeto GigaBOT na sua arquitetura. A gestão do laboratório é mantida pelo projeto *AccessService* que faz parte da implementação de referência do GigaBOT. A arquitetura do NetLab foi proposta de modo a acomodar experimentos para o domínio de redes de computadores.

O NetLab é o projeto utilizado como base para a implementação da arquitetura proposta nesta dissertação. Este trabalho foi selecionado por seguir o paradigma da computação orientada a serviços e possuir experimentos para o domínio de redes de computadores, que é o domínio pelo qual o trabalho apresenta o experimento para prova de conceito.

## 1.3 Objetivos do Trabalho

Dentre os principais objetivos deste trabalho, destacam-se:

1. utilizar suporte a módulos dinâmicos para aumentar a disponibilidade de experimentos, bem como a disponibilidade do próprio laboratório (7 dias por semana e 24 horas por dia);
2. utilizar padrões de desenvolvimento baseados em serviços sem que se comprometa a escalabilidade do laboratório e de seus experimentos;
3. propiciar interoperabilidade por meio da utilização de protocolos não bloqueados por *firewalls*;
4. aumentar a dinamicidade e adaptabilidade dos experimentos, possibilitando ao usuário o emprego de topologias de redes dinâmicas e adaptáveis aos experimentos utilizados;
5. proporcionar independência de um experimento em relação a outros experimentos, para garantir que nenhum experimento interfira no funcionamento do laboratório, contribuindo, assim, com o aumento da disponibilidade;
6. facilitar a criação de novos experimentos, bem como a disponibilização de experimentos mediante o controle de versões sem perda da disponibilidade.

## 1.4 Organização da Dissertação

O restante deste trabalho constitui-se de 4 capítulos. A ordem em que são apresentados elucida a sequência em que as atividades foram desenvolvidas.

O Capítulo 2 apresenta os principais trabalhos que originaram na criação de WebLabs, que têm como base uma Arquitetura Orientada a Serviços (*Service Oriented Architecture - SOA*). Ao final do capítulo, é apresentada uma análise dos trabalhos apresentados e levantados os requisitos para laboratórios, independente de domínio e específicos para o domínio de redes de computadores.

O Capítulo 3 descreve em detalhes, a arquitetura proposta para atender aos principais requisitos levantados. O Capítulo 4, apresenta os aspectos de implementação da arquitetura proposta, bem como a avaliação por meio da criação de um experimento de configuração de VLANs.

Por fim, as conclusões são apresentadas no Capítulo 5, tendo como base a arquitetura proposta, sua implementação e utilização por meio da criação de um novo experimento. Propostas para continuidade dos trabalhos também são apresentadas.

## Capítulo 2

# Laboratórios de Acesso Remoto

O WebLabs, de modo geral, estão voltados para a área de robótica ou de eletrônica [6, 7, 8, 9], assim, poucos são os WebLabs para redes de computadores. Entre os trabalhos discutidos, destacamos aqueles cuja arquitetura é orientada a serviços (SOA), o que permite a execução de seus experimentos por meio de chamadas de serviços Web. Trabalhos que definem uma arquitetura genérica, que podem ser implementadas para domínio de quaisquer laboratórios, também são destacados.

### 2.1 Arquitetura do WebLab-Deusto

O WebLab-Deusto [7] é um laboratório de experimentação remota desenvolvido por um grupo de pesquisadores da Universidade de Deusto Bilbao, Espanha. Teve seu projeto iniciado no final de 2001 e vem evoluindo ao passar do tempo. Na primeira versão, v0.1, a interação entre a aplicação cliente e o laboratório era realizada por meio de uma aplicação *desktop*, a versão seguinte, v1.0, utilizava *Applets* Java para a interação, já as versões 2.0 e a atual 3.0 passaram a utilizar aplicações Web com a tecnologia AJAX (*Asynchronous Javascript And XML*).

O WebLab-Deusto possui uma arquitetura distribuída e dividida em camadas. Essas camadas estendem o modelo MVC (*Model View Controller*), por incluir mais camadas distribuídas no lado do servidor. A Fig.2.1 apresenta a visão geral da arquitetura.

Os WebLabs Servers são compostos por um conjunto de servidores que são referenciados como camadas. Para o projeto, foram definidas camadas como:

- *Model* - camada composta pelo banco de dados, denominada *database*;
- *View* - camada de apresentação no domínio do cliente;
- *Controller* - camada responsável pelas regras, denominada *logic*;
- *Authentication* - camada que acomoda os serviços relacionados à autenticação;
- *Laboratory Server* - camada da rede interna do laboratório que provê a comunicação da camada de lógica com os experimentos;
- *Experiment Server* - camada mais acoplada ao experimento, com implementação específica do *hardware*.

Os servidores de experimento, denominados *MicroServers*, são responsáveis por prover a interação com o *hardware* do laboratório, por isso são dependentes da plataforma suportada pelo dispositivo. Contudo, a comunicação entre um WebLab Server com o *MicroServer* pode ser realizada por algum dos cinco tipos de protocolos disponíveis na implementação.

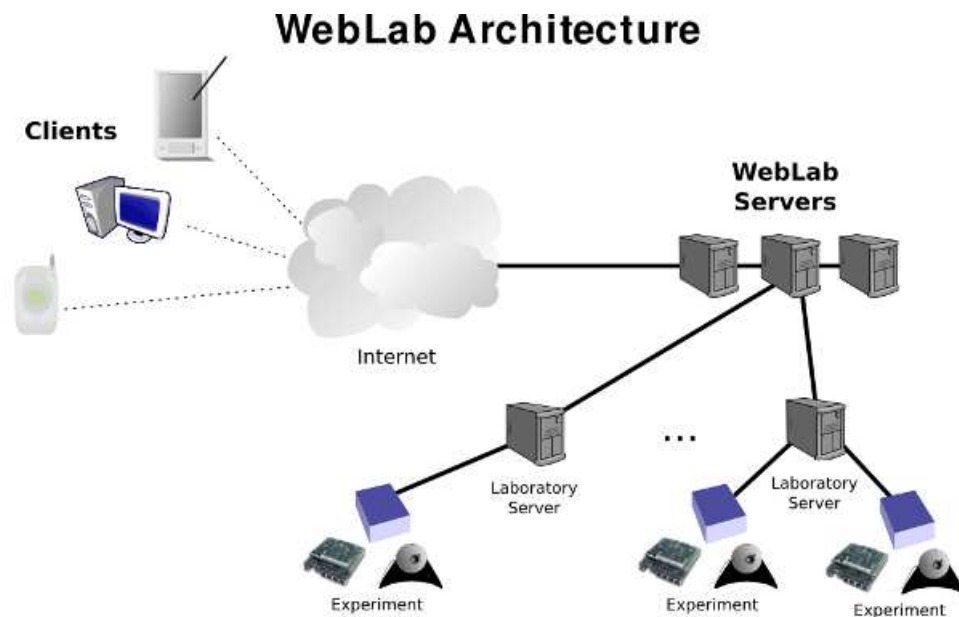


Fig. 2.1: Visão Geral da Arquitetura do WebLab-Deusto [10].

A arquitetura define cinco tipos de protocolos para a comunicação entre as camadas [7], cuja seleção fica a critério do desenvolvedor do experimento ou servidor. Os protocolos definidos, bem como uma breve descrição de utilização de cada um, são apresentados a seguir:

1. *"Direct"* - quando os servidores estão em execução em um mesmo processo com instâncias de objetos diferentes;
2. *TCP sockets* - utilizado quando dois servidores estão localizados na mesma rede e em processos diferentes;
3. *UNIX sockets* - utilizado quando dois servidores estão localizados na mesma máquina *Unix* em diferentes processos;
4. *Python-dependent SOAP* - utilizado quando o servidor de destino está atrás de um *firewall*, sendo que nesse caso, o servidor pode estar dentro ou fora da rede do laboratório;
5. *XML-RPC* - utilizado para comunicação com os servidores de experimento.

A Fig.2.2 expõe os vários servidores e os protocolos utilizados para comunicação entre eles.

Para as aplicações clientes, executadas pelo navegador na máquina do usuário, foi definido um *framework*, que implementa uma arquitetura denominada XWL (eXensible WebLab) [7]. Essa arquitetura abstrai a carga do experimento, é também a comunicação entre a aplicação cliente e o servidor. A interação da aplicação Web é realizada por meio de *JavaScript*, o que possibilita a utilização de tecnologias como *Applets Java*, *Adobe Flash*, *HTML* ou qualquer tecnologia que dê suporte a *XML-RPC*.

A implementação do XWL é responsável por controlar toda a parte administrativa do laboratório, como fila de espera para execução de experimento e autenticação. A implementação da arquitetura XWL resulta em uma aplicação que, quando solicitada a execução de um experimento, ela é carregada no navegador do usuário e, então, carrega o experimento. O experimento que faz uso do *framework* deve utilizar as rotinas disponibilizadas para comunicação por meio das interfaces definidas por ele.

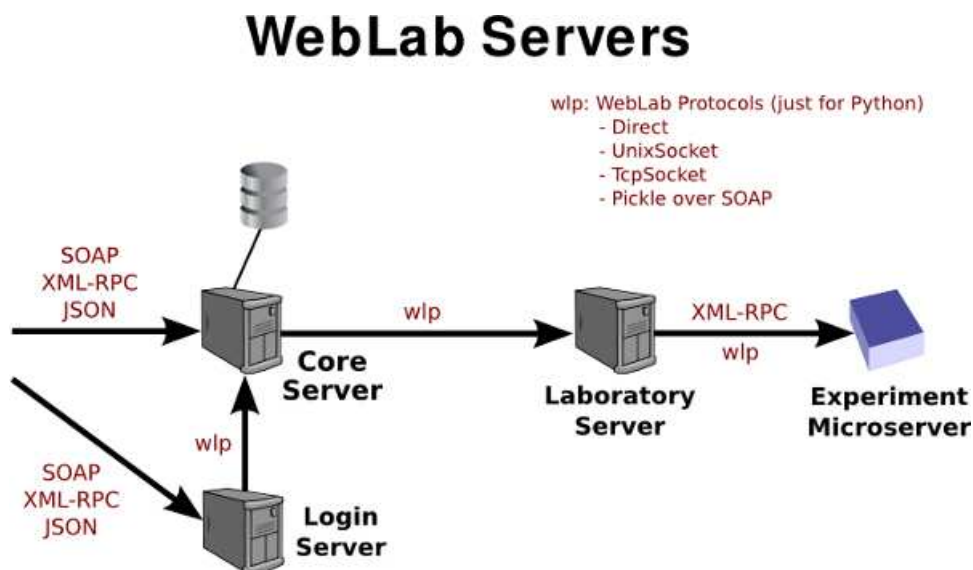


Fig. 2.2: Servidores e Protocolos Utilizados para Comunicação [10].

Para experimentos que utilizam o XWL, é necessária a criação da aplicação cliente e de sua contrapartida no servidor, a aplicação no servidor é responsável em comunicar-se diretamente com o *hardware* do experimento. O protocolo utilizado para a comunicação entre a aplicação cliente e o servidor é o HTTP por meio de AJAX, já do servidor do laboratório para o do experimento é *XML-RPC*.

### 2.1.1 Experimentos Contemplados

O WebLab-Deusto provê experimentos para o curso de Engenharias Elétrica e Automação da Universidade de Deusto. Os experimentos vêm sendo utilizados pela instituição desde 2004. Um dos experimentos disponibilizados que faz parte do projeto é o WebLab-FPGA, que permite que o usuário interaja com uma placa FPGA (*Field Programmable Gate Array*). Esse experimento é composto por um equipamento externo (placa FPGA) conectado por meio de um cabo USB com o microservidor, responsável por controlá-lo e uma *webcam*.

A Fig.2.3 demonstra o experimento em execução. Ao ser carregado pelo navegador na máquina do usuário, é apresentado um conjunto de botões e caixa de texto que o usuário pode selecionar ou entrar com valores para interagir com o equipamento. Ao selecionar alguns desses botões é possível visualizar a mudança de estado do equipamento pela imagem exibida. No caso desse experimento, não é apresentado o vídeo e, sim, imagens, que são capturadas pela câmera.

Outro experimento disponibilizado é de GPIB (*General Purpose Instrumentation Bus*), criado como WebLab-GPIB. Esse experimento é composto por dois equipamentos: um de análise de espectro e outro de gerador de sinais. Permite que o usuário gere sinais e possa visualizar, por meio de um vídeo a análise do sinal feita pelo analisador de espectro.

### 2.1.2 Análise da Arquitetura do WebLab-Deusto

O modelo de servidores separados por camadas dificulta a manutenção do laboratório, pois, torna-se uma carga muito grande para o administrador manter atualizados todos esses servidores.

O uso de protocolos que acessam diretamente os servidores do laboratório pode possibilitar que experimentos sejam executados sem prévia reserva ou controle, por isso, a falta de um componente

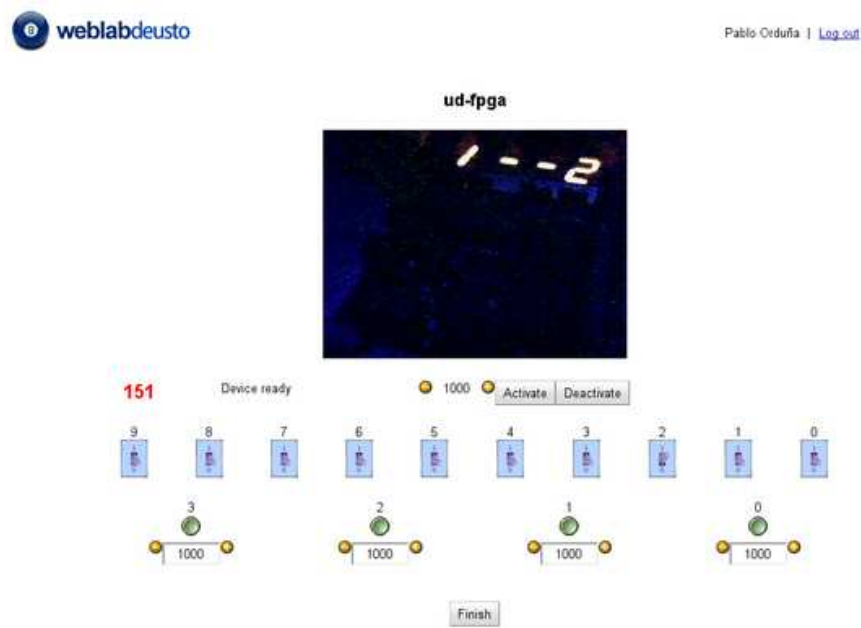


Fig. 2.3: Experimento WebLab-FPGA [7].

da arquitetura para centralizar as requisições no lado do servidor facilita o acesso não autorizado, pois fica por conta do desenvolvedor incluir, na aplicação, as regras de autenticação, autorização e reservas.

O projeto permite o uso de várias tecnologias para criar os experimentos. Essa característica também exige muito de administradores do laboratório para mantê-las constantemente atualizadas. O XWL, embora facilite a comunicação entre o cliente e o servidor, por ser uma camada de abstração, traz para a aplicação cliente regras que deveriam ser aplicadas somente no servidor. Além de replicação de código, pode comprometer a segurança do laboratório, no caso de algum usuário alterar, por meio de alguma ferramenta os fontes em *JavaScripts*.

## 2.2 Arquitetura do WebLab-SIT

É um WebLab desenvolvido pelo SIT (*Stevens Institute of Technology*) para os cursos de engenharia. O projeto propicia a criação de laboratórios de experimentação real e virtual. Os laboratórios reais permitem a interação com equipamentos utilizados nos experimentos. Já o laboratório virtual não tem acesso a *hardware*, sendo este, simulado por *software*.

O WebLab-SIT possibilita a criação de experimentos de três tipos:

1. *Batch-Mode* - experimento de lote em que o usuário informa os dados de entrada por meio de uma página no navegador Web. Uma vez enviados os dados para o experimento, esses são incluídos em uma fila de execução, e após executado, envia um email para o usuário com o resultado. Nesse modelo de experimento, o usuário não pode interagir durante a execução;
2. *Real-Time* - experimento de tempo real, permite que o usuário acompanhe por meio de vídeo e interaja com o experimento durante sua execução;
3. *Virtual* - experimento que simula o *hardware* por meio de *software*.

A arquitetura do WebLab-SIT [8] foi projetada de forma a prover maior escalabilidade, com foco na integração do *software* com o *hardware* disponíveis nos laboratórios.

A Fig.2.4 ilustra a visão geral da arquitetura do laboratório. A interação com o experimento é realizada por meio de um navegador Web, que, dependendo do experimento, pode utilizar *Applets* Java ou simples HTML. O servidor de aplicação é responsável em prover a parte administrativa do laboratório e receber a requisições de execução dos experimentos.

Para laboratório virtual, a arquitetura provê dois tipos de aplicações:

1. *server based* - executada no navegador do usuário e pode conter uma *applet* java embutido, que são os programas que simulam um determinado *hardware*. Em casos em que *applets* Java são utilizados, esses *Applets* provêm acesso a modelos 3D de VRML (*Virtual Reality Modeling Language*), que simulam ambientes para a execução do experimento. Nesse modelo, todo o processamento da simulação é realizado no servidor;
2. *stand-alone* - são aplicações executadas na máquina do usuário e se conectam no servidor para acesso ao banco de dados, a fim de obter informações acerca do experimento. Essas aplicações podem ser desenvolvidas em Python, wxPython, Vpython, o que proporciona a apresentação de modelos 3D.

No laboratório de experimentação real, o servidor possibilita a interação com câmeras e com outros servidores denominados *Instrument Controllers*, que são responsáveis por controlar os experimentos. Os *Instruments Controllers* possuem *hardware* e *software* proprietários para interagir com os equipamentos e experimentos.

O protocolo utilizado pela aplicação cliente, para comunicar com o servidor, é o HTTP, tanto para o laboratório real quanto virtual. No laboratório real, o protocolo TCP/IP é utilizado para comunicar com o *Instrument Controller* e as câmeras.

Os *Instruments Controllers* necessitam de uma aplicação proprietária o *LabView* [11]. Essa aplicação é responsável em prover a interação do computador com o experimento. Os comandos enviados aos experimentos são realizados por meio de *scripts*, que são carregados pelo *Instrument Controller* no *LabView* e executados cujo resultado é retornado para o servidor Web.

Para fornecer o fluxo de dados dos vídeos capturados, foi incluído um servidor de vídeo, que é responsável por capturar o áudio e vídeo e transmiti-los por *broadcasting* ou *multicasting*. Esse servidor também conta com tecnologia proprietária para prover o serviço.

### 2.2.1 Experimentos Contemplados

O WebLab-SIT é utilizado, inicialmente, nos cursos de engenharia mecânica da instituição. Os experimentos são classificados como virtual, para simulação, ou real, que são executados em equipamentos que fazem parte do laboratório. A maioria dos experimentos são disponibilizados nas duas categorias, o que torna possível que um número maior de alunos executem e simulem a sua execução.

O experimento virtual permite que alunos simulem a execução em um *software* e ainda visualizem sua execução por meio de modelos 3D em VMRL. O objeto em 3D representa o *hardware* do experimento e pode ser visualizado pela aplicação cliente. A Fig.2.5 apresenta a execução de um experimento virtual, denominado Sistema de Vibração Mecânica (*Mechanical Vibration System*), que permite criar vibrações por meio de um equipamento mecânico. Esse experimento é executado por um *applet* Java embutido no HTML.

O mesmo experimento apresentado anteriormente existe como experimento real, a Fig.2.6 apresenta o em execução com acompanhamento em tempo real por meio de vídeo.

Uma outra categoria de experimentos do WebLab-SIT é a execução em lote (*Batch-Mode Remote Experiments*). Para esses experimentos, os alunos acessam o portal do laboratório, selecionam

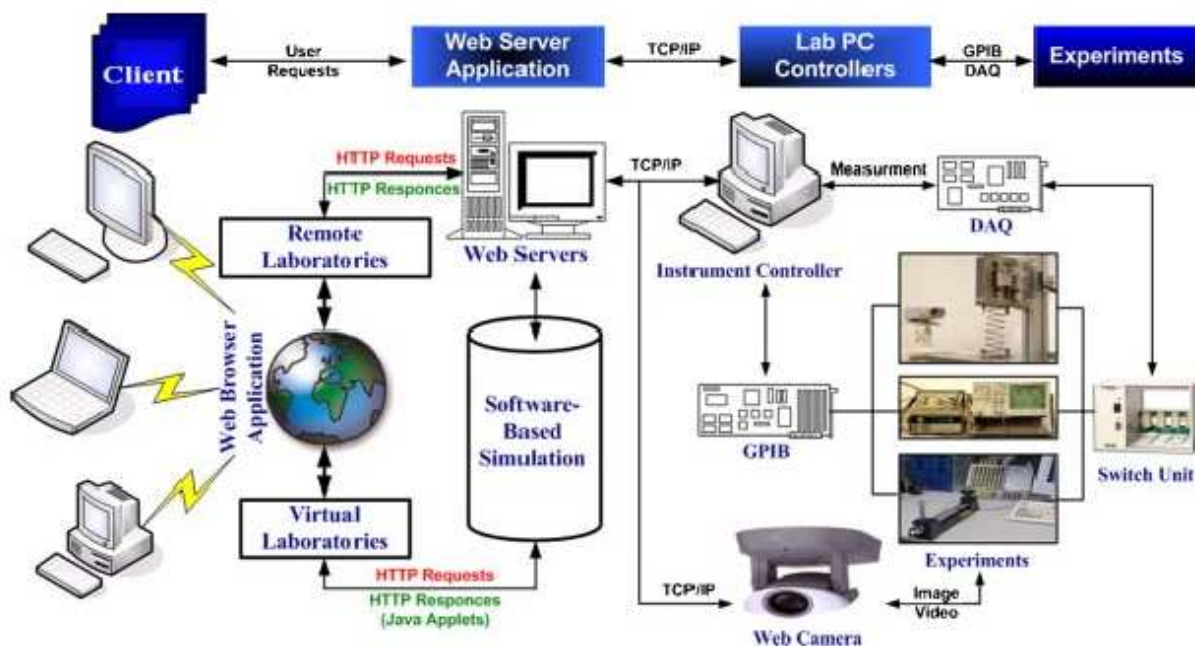


Fig. 2.4: Visão Geral da Arquitetura [8].

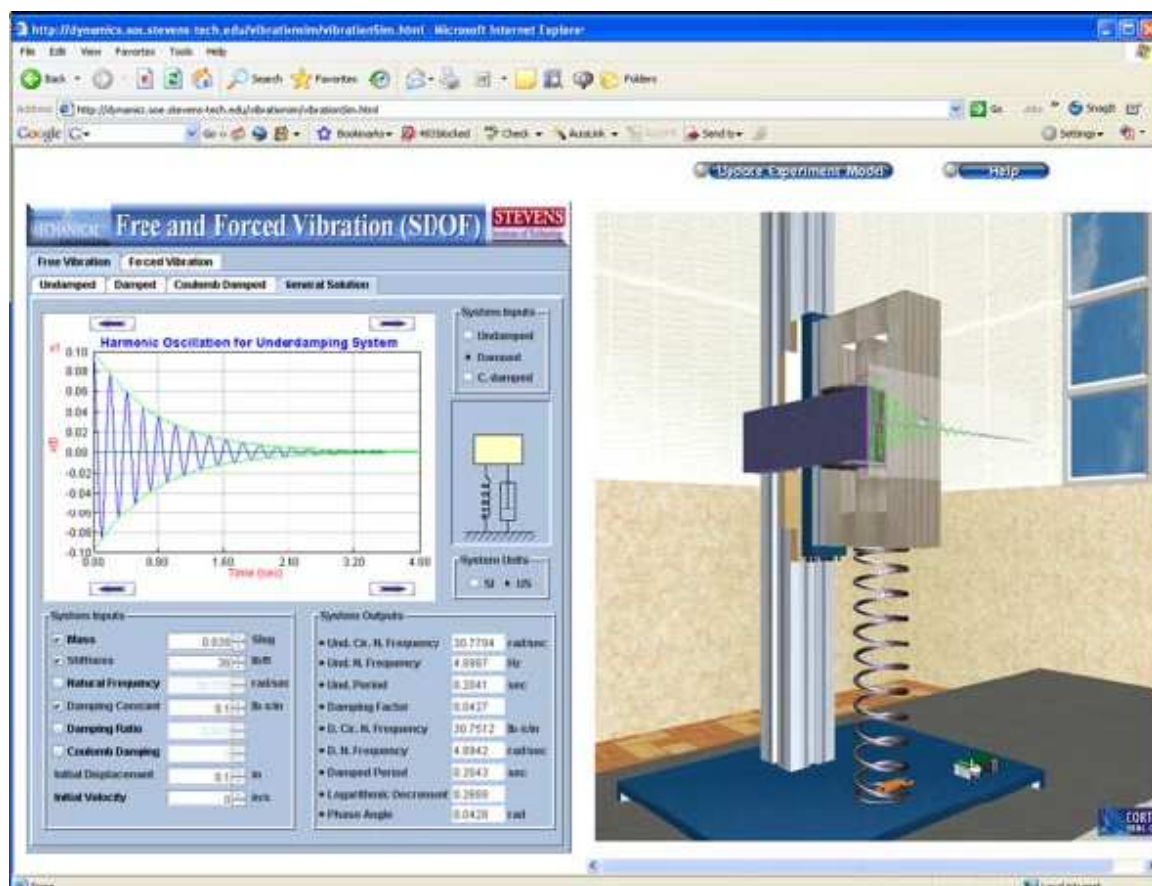


Fig. 2.5: Aplicação Cliente Executando um Experimento Virtual [8].

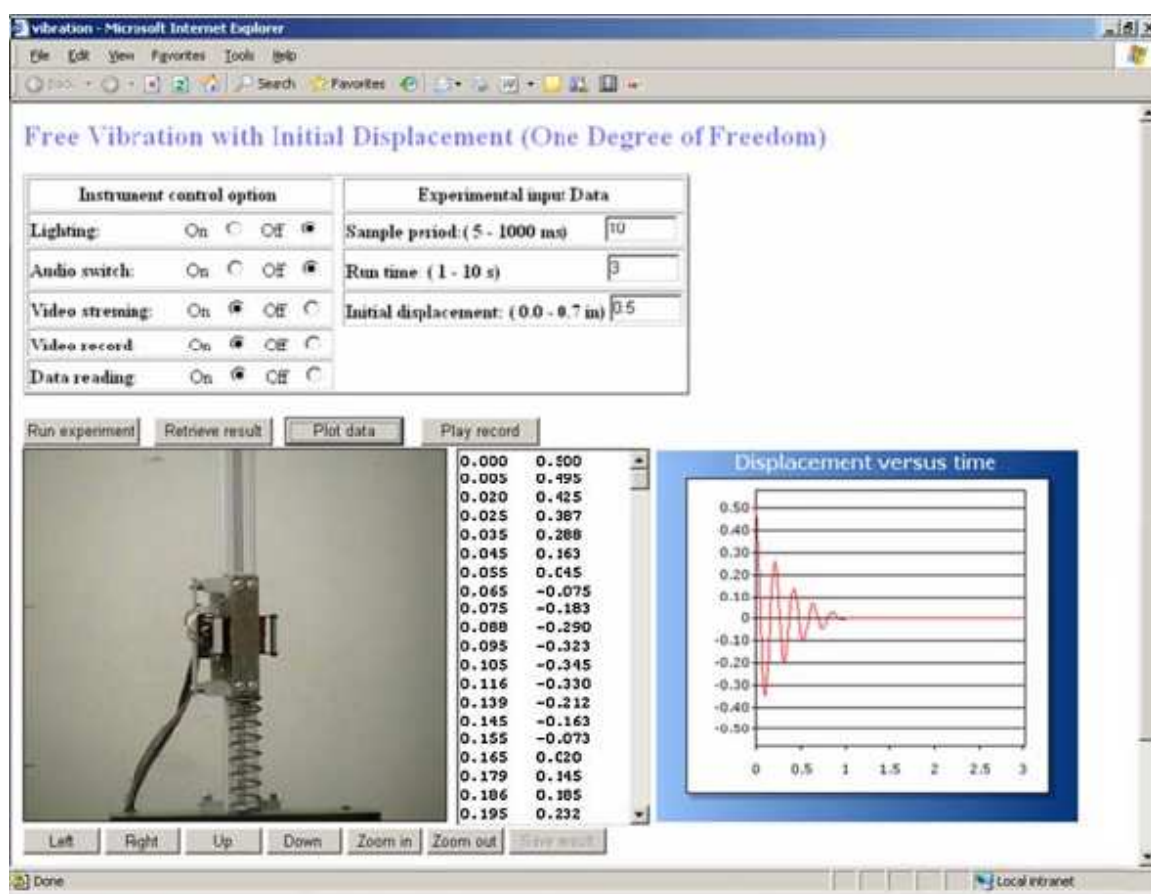


Fig. 2.6: Experimento Remoto do Tipo Real em Execução [8].

o experimento, informam os dados de entrada e enviam o formulário para o servidor, que adiciona a execução em uma fila, que processa um experimento por vez. Após finalizada a execução do experimento, os dados referentes à execução são disponibilizados para *download* e enviado um email para o aluno, informando o final da execução. Os dados de saída de um experimento em lote podem ser dados, áudio ou vídeo. A Fig.2.7 apresenta uma visão geral de como é realizada a execução desse tipo de experimento.

### 2.2.2 Análise da Arquitetura do WebLab-SIT

Embora o projeto tenha boa parte implementada utilizando-se de ferramentas de código aberto, as partes responsáveis em interagir com o experimento são todas proprietárias, tanto *hardware* quanto *software*, o que acaba impactando na implantação e uso do laboratório, pois o custo das licenças e dos equipamentos dificulta a criação de novos laboratórios na instituição.

A estrutura do projeto se mostra um tanto complexa por fazer uso de muitas tecnologias, como Java, Jython, PHP, JSP, Python, LabView, CGI e .NET.

As aplicações *stand-alone*, executadas nas máquinas clientes, podem ser consideradas uma falha de segurança, pois a arquitetura não define um modelo central para a execução dos experimentos virtuais, uma vez que tais aplicações necessitam de acesso ao servidor do laboratório para obter informações acerca dos experimentos a serem simulados.

A arquitetura não define nenhum componente que permita a execução de serviços web, o que diminui a interoperabilidade entre laboratórios.

A falta de padronização na criação dos experimentos possibilita que sejam criados quaisquer

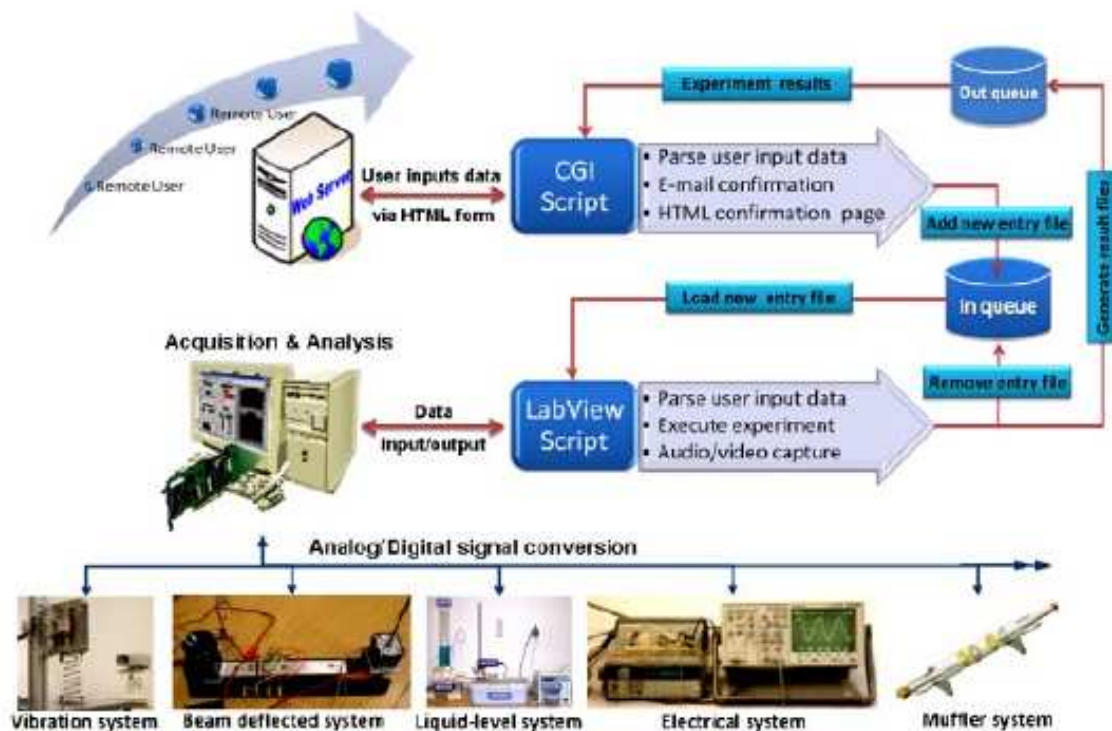


Fig. 2.7: Visão Geral da Execução de um Experimento em Lote [8].

tipos de experimento, o que pode comprometer a segurança do laboratório e, ainda, dificultar sua documentação.

## 2.3 Arquitetura do iLab

O iLab [6] é desenvolvido pelo MIT (*Massachusetts Institute of Technology*), e apresenta uma arquitetura compartilhada (*iLab Shared Architecture - ISA*). O projeto consiste de um *framework*, que provê uma infraestrutura de serviços Web, que fornece suporte à criação de laboratório remoto. Os laboratórios criados podem ser interligados por meio de rede de conexões Internet, de modo que os alunos, uma vez autenticados possam acessar qualquer um dos laboratórios disponíveis.

A arquitetura do iLab possibilita a execução de experimentos remotos pela Internet, em que tanto a interação do cliente com o servidor quanto a comunicação entre os servidores internos do laboratório são realizadas por serviços Web. Foi um dos primeiros projetos de laboratório de experimentação remota a utilizar serviços Web na execução dos experimentos.

A arquitetura do iLab é dividida em três camadas:

1. *Lab Client* - composta pela aplicação cliente, que é executada pelo usuário por meio de um *applet* java ou uma aplicação *desktop* baixada no portal do laboratório;
2. *Service Broker* - camada responsável em manter a parte administrativa e prover integração entre as outras duas camadas. É nessa camada que residem as regras de utilização do laboratório;
3. *Lab Server* - camada composta por servidores que são conectados aos equipamentos do laboratório, esses servidores são responsáveis em prover a comunicação com os equipamentos.

A forma com que a arquitetura foi projetada permite que experimentos entre diferentes cursos ou instituições sejam compartilhados. O componente da arquitetura responsável em prover esse compartilhamento é o *Service Broker*. Esse componente é localizado na instituição e, por meio de serviços Web o *Service Broker*, comunica-se com os *Lab Servers*, que podem estar no campus da instituição do qual o aluno faz parte ou em qualquer outro campus que seja integrado ao laboratório. A autenticação e autorização são de responsabilidade do *Service Broker*, assim, antes de executar algum experimento, o usuário deve antes efetuar autenticação no *Service Broker*. A Fig.2.8 apresenta a visão geral da arquitetura do iLab.

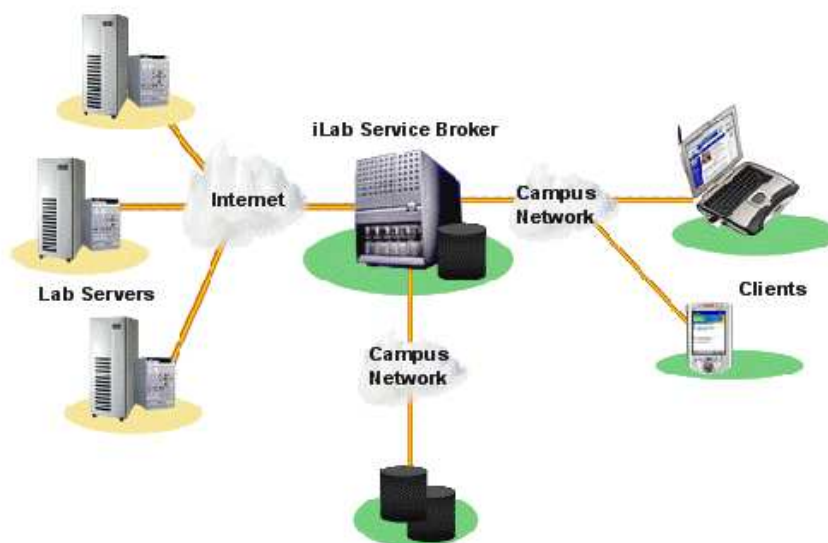


Fig. 2.8: Visão Geral da Arquitetura do iLab [12].

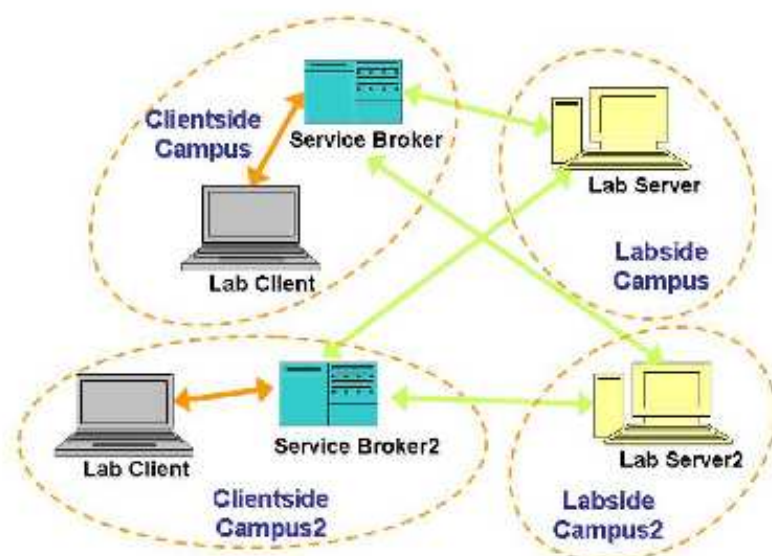


Fig. 2.9: Arquitetura Compartilhada do iLab. [6].

Para o compartilhamento, entre o campus de uma instituição e outro, é necessária a mesma estrutura, ou seja, um *Service Broker* e *Lab Servers* (Fig.2.9). A aplicação cliente no domínio do usuário comunica-se com o *Service Broker* por meio de serviços Web, utilizando o protocolo SOAP.

A arquitetura compartilhada do iLab possibilita a criação de três tipos de experimentos:

1. *Batched* - experimentos de execução em lote, que não permite a interação com o aluno;
2. *Interactive* - experimentos que permitem ao aluno interagir com o equipamento do laboratório;
3. *Sender* - experimentos para monitoramento de sensores em tempo real.

Para cada categoria de experimentos, a arquitetura fornece um conjunto de serviços específicos, que são providos e compartilhados pelo *Service Broker*.

### 2.3.1 Experimentos Contemplados

Os primeiros experimentos criados no iLab foram para a matéria de microeletrônica do curso de engenharia. O principal motivo de prover execução de experimentos remotos era de maximizar o uso dos equipamentos do laboratório para os alunos do curso. Esse princípio deu origem as duas categorias de experimentos, a) execução em lote, em que o aluno não tem necessidade de interagir com o experimento durante sua execução e b) execução de experimentos que necessitam de interação com o aluno.

A estrutura do *Lab Server* para suportar a execução de experimentos em lote é apresentada na Fig.2.10. Os passos para execução do experimento em lote são:

1. efetuar autenticação no *Service Broker*;
2. selecionar o experimento desejado;
3. executar o experimento por meio de página HTML ou baixar, a aplicação cliente;
4. entrar as informações solicitadas pela aplicação cliente do experimento;
5. solicitar a execução.

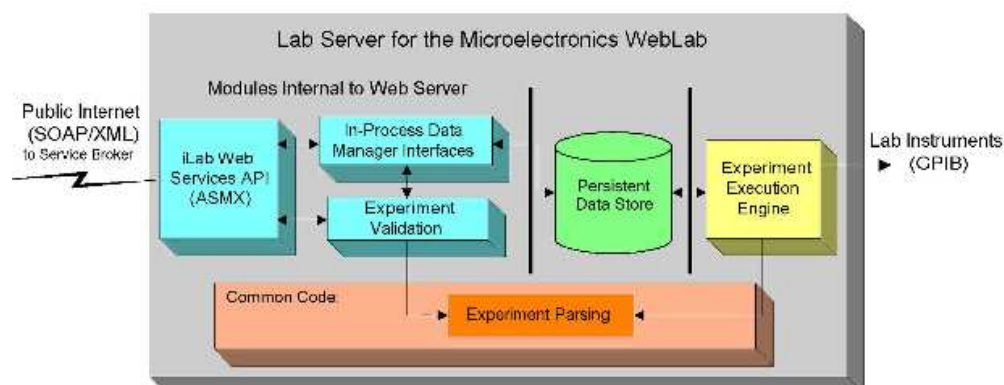


Fig. 2.10: Estrutura para Execução de Experimentos em Lote [6].

Depois de executados os passos acima, com a aplicação cliente já carregada na máquina do aluno, a partir desse ponto, as execuções são realizadas utilizando o protocolo SOAP, tanto do cliente para o *Service Broker* quanto do *Service Broker* para o *Lab Server*. A troca de mensagens é efetuada por meio de arquivos XML.

Ao receber uma solicitação de execução de experimento em lote, o *Lab Server* processa os dados de entrada informados pela aplicação cliente e armazena os dados em um banco de dados. Outro componente o *Experiment Execution Engine* acessa essa mesma base de dados para poder executar o experimento no equipamento do laboratório. Ao término da execução, o resultado é armazenado no banco de dados, que é retornado para o *Service Broker*.

Uma característica dos experimentos em lote é de permitir que vários alunos enviem solicitações de execuções de experimentos. Essas solicitações são armazenadas no banco de dados, e o *Experiment Execution Engine* as trata como uma fila, processando uma por vez.

Os experimentos do tipo *Interactive* permitem ao usuário interagir com o experimento durante sua execução. O uso de um mediador (*Service Broker*) para centralizar a execução de experimentos, embora dê mais segurança ao laboratório causa um aumento de latência na rede, pois, para cada requisição que chega a esse mediador, esta é encaminhada ao *Lab Server*. Por esse motivo, os experimentos que permitem a interação com o aluno podem ser acessados diretamente pela aplicação cliente do experimento. Para possibilitar esse acesso, é necessário que partes das validações e serviços oferecidos pelo *Service Broker* sejam implementadas na aplicação cliente. A Fig.2.11 ilustra como é a arquitetura com suporte a essa categoria de experimento.

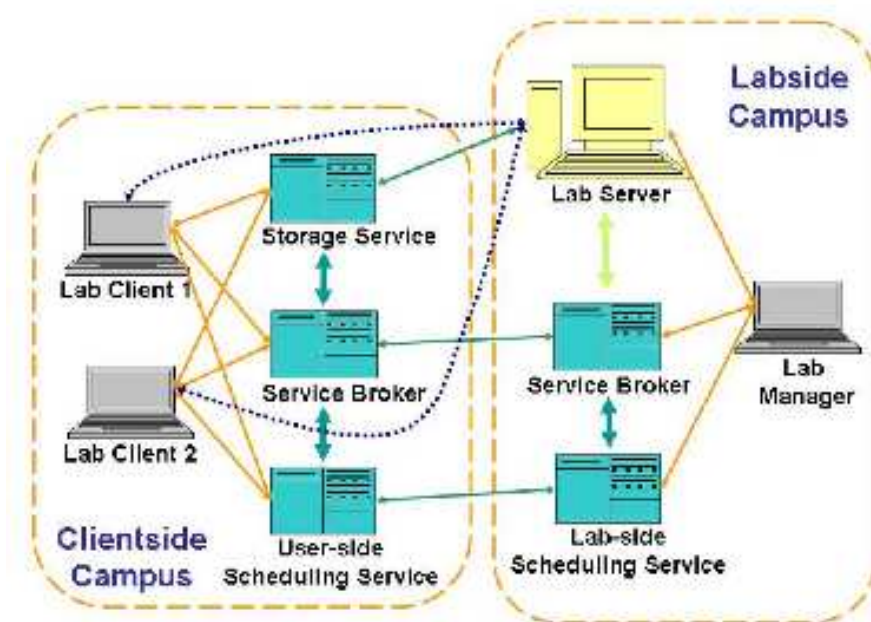


Fig. 2.11: Arquitetura para Experimentos com Interação do Aluno [6].

A estratégia de proporcionar a conexão direta com o servidor do laboratório permite a utilização de ferramentas de terceiros como o LabView ou MatLab para prover a interação com o equipamento.

### 2.3.2 Análise da Arquitetura do iLab

A arquitetura compartilhada do iLab propicia uma boa integração entre cursos, campus e até instituições diferentes. A definição de um componente que centraliza a execução dos experimentos (*ServiceBroker*) contribui para aumentar a interoperabilidade dos laboratórios criados, no entanto a segurança da execução dos experimentos é comprometida devido à possibilidade de utilizar clientes que incluam as funcionalidades desse *ServiceBroker*. Essa característica passa para o desenvolvedor a responsabilidade de incluir tais funcionalidades, que, quando não implementadas, podem comprometer o uso dos experimentos e do laboratório.

Grande parte da arquitetura é implementada em ferramentas de código aberto, mas também inclui ferramentas que necessitam de licenças, tanto na execução dos experimentos quanto nos servidores do laboratório.

## 2.4 Arquitetura do WebLab-DACSE

O trabalho apresentado por Calvo [9] não apresenta um WebLab propriamente dito e sim uma metodologia para a criação de laboratórios remotos. A metodologia propõe um conjunto de componentes que são utilizados na criação de laboratório de acesso remoto, em que esses componentes podem ser reutilizados na criação de outros laboratórios independentes de domínios. O trabalho é desenvolvido pelo Departamento de Controle e Automação (*Department of Automatic Control and System Engineering - DACSE*) da *University of the Basque Country*, Bilbao, Spain.

A arquitetura é projetada para possibilitar que vários laboratórios sejam criados por meio da reutilização dos componentes. É uma arquitetura genérica composta pelos seguintes componentes:

- *Physical Devices* - dispositivos físicos dos experimentos;
- *Application Server* - servidor de aplicação responsável por executar os experimentos nos dispositivos físicos;
- *Remote Application* - aplicação cliente que permite a interação entre o aluno e o experimento.

A aplicação cliente pode ser executada com vários perfis diferentes, esses perfis definem o conjunto de operações que o usuário pode executar no experimento. O servidor de aplicação serve de ponte entre a aplicação cliente e o equipamento do laboratório. Esse servidor é considerado o componente principal da arquitetura. Todas as validações relacionadas às ações executadas no experimento, antes, devem ser interceptadas por esse componente. A Fig.2.12 apresenta uma visão geral da arquitetura proposta.

A estrutura interna do servidor de aplicação segue o modelo MVC (*Model View Controller*), que separa uma aplicação em camadas de Modelo, Apresentação e Controle. Esse modelo possibilita que aplicações remotas de diferentes implementações acessem os mesmos modelos e regras de negócio, sem que seja necessário haver alterações na parte de modelo. Dessa forma, é possível existir várias aplicações acessando os mesmos objetos no servidor de aplicações. A Fig.2.13 ilustra como é estruturado o servidor de aplicações seguindo o modelo MVC.

Os componentes que compõem o servidor de aplicação são listados a seguir:

- *Virtual Plant (VP)* - é composto por uma coleção de objetos que representam os equipamentos físicos do experimento;
- *View Models (VM)* - agrupamento de permissões que são atribuídas e identificadas por perfis como: administrador, aluno, professor. Esse componente é criado a partir de um componente pai, o qual possui um conjunto de funcionalidades padrão;
- *View Model Manager (VMM)* - gerencia as conexões das aplicações clientes. É um centralizador de conexões e qualquer ação executada pela aplicação cliente é recebida por esse componente;

A interação inicia-se pela aplicação cliente que deve realizar a autenticação. Uma vez autenticado, o aluno pode interagir com o laboratório por meio da aplicação cliente. Ao executar alguma ação, esta chega até o VMM, que confirma se há uma autenticação válida, caso positivo, retorna a referência ao

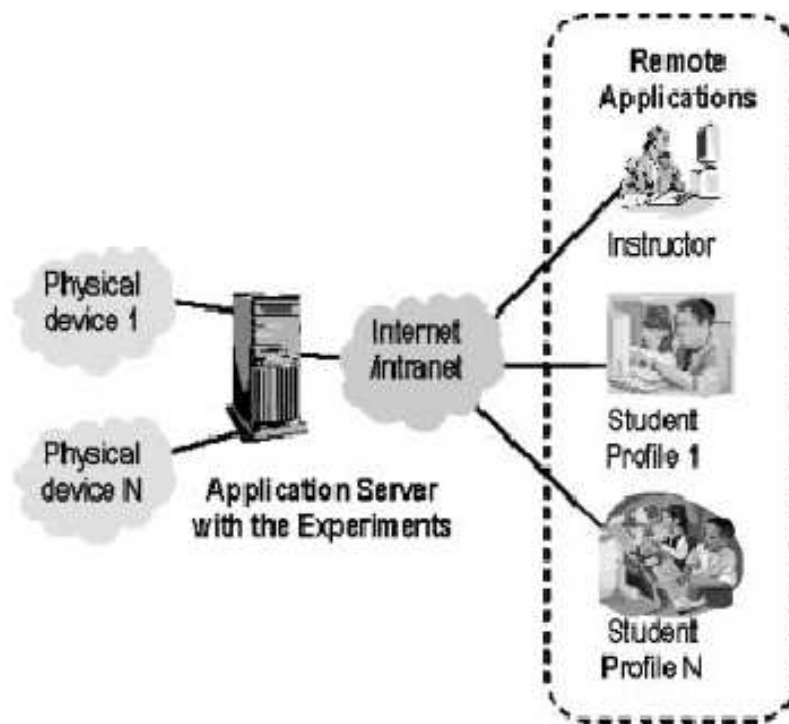


Fig. 2.12: Visão Geral da Arquitetura Genérica [9].

objeto que representa o experimento de acordo com o profile do aluno autenticado. Desse ponto, a aplicação cliente já pode solicitar a interação com o equipamento do laboratório por meio do VM que contém o conjunto de ações que o aluno pode executar no dispositivo.

A execução das ações no dispositivo físico não é realizada diretamente pelo objeto que o representa, essa ação é delegada para um controlador, que a executa e retorna para o objeto.

A metodologia define passos que devem ser seguidos para definir os componentes principais da arquitetura, conforme listados a seguir:

1. definição o laboratório remoto;
  - definição dos objetivos de ensino;
  - desenho da planta física do laboratório;
  - identificação dos perfis;
  - identificação das operações de cada perfil;
2. desenho do sistema de acesso remoto;
  - personalização da arquitetura;
  - seleção das tecnologias;
  - desenho da estrutura interna do servidor de aplicação;
3. implementação do laboratório remoto.

Os passos servem de guia para a definição final da arquitetura, uma vez que a arquitetura inicial é modelada de forma genérica. Ao seguir os passos apresentados, a arquitetura se torna mais específica para o laboratório em questão.

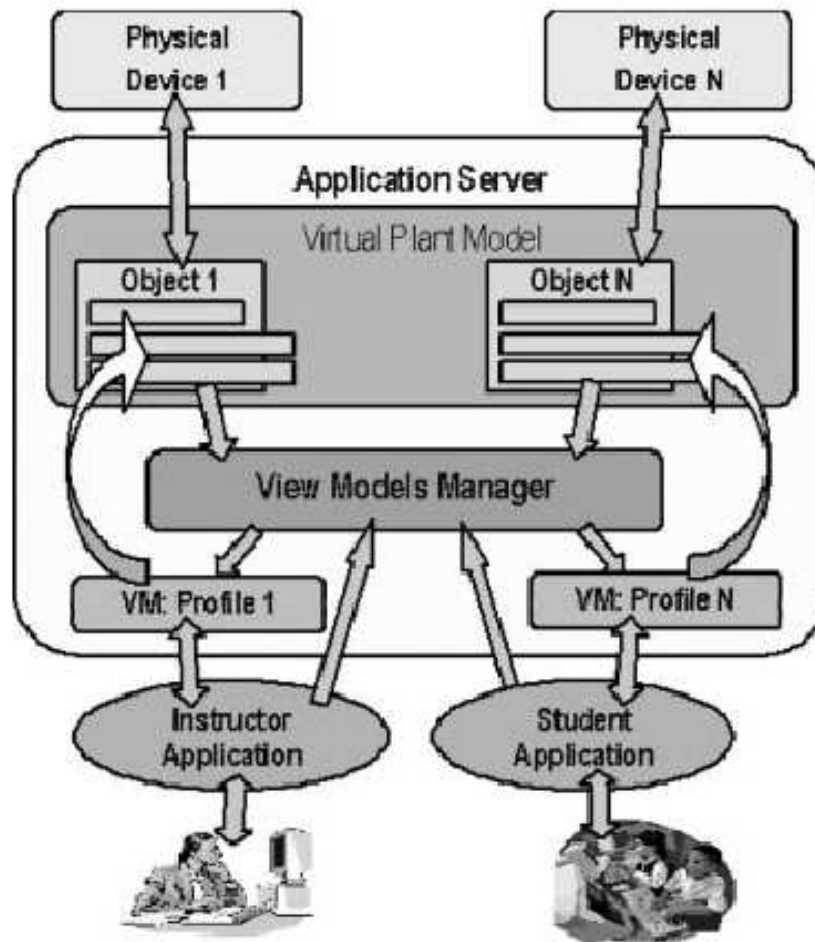


Fig. 2.13: Estrutura dos Componentes do Servidor de Aplicação [9].

Os protocolos empregados entre as camadas são selecionados na definição da arquitetura específica do domínio, por isso, qualquer protocolo como: RMI, TCP/IP ou SOAP pode ser utilizado entre as camadas.

### 2.4.1 Experimentos Contemplados

Os experimentos são disponibilizados segundo o domínio do qual fazem parte. Ao desenvolver um novo laboratório, os dispositivos físicos devem ser mapeados como objetos, que são identificados na etapa de análise. Como o projeto não define um WebLab propriamente dito e, sim, uma metodologia, para sua criação, não existem experimentos criados.

### 2.4.2 Análise da Arquitetura do WebLab-DACSE

A arquitetura genérica pode ser utilizada para criar qualquer tipo de laboratório, contudo a reutilização de um laboratório, bem com seu conjunto de experimentos ficam limitados ao WebLab criado, visto que cada WebLab tem uma implementação diferente, conforme apresentado. Isso impossibilita a reutilização de experimentos já criados em um outro laboratório. Para cada novo WebLab, é necessária uma nova implementação.

Cada WebLab pode ter uma implementação diferente, o que dificulta a manutenção nesses laboratórios.

## 2.5 Arquitetura do NetLab WebLab

O NetLab WebLab [3] é um laboratório de acesso remoto para o domínio de redes de computadores, que segue o paradigma da arquitetura orientada a serviço (SOA). É um projeto desenvolvido pela Faculdade de Computação da Universidade Federal de Uberlândia que tem como objetivo, prover acesso a experimentos relacionados a redes de computadores. Tem como base o modelo conceitual do projeto GigaBOT [13], que permite a criação e integração entre laboratórios.

Por ter como base o GigaBOT, o NetLab utiliza toda a sua parte administrativa como manutenção de usuários, grupos, reservas de experimentos, autorização e autenticação. Os elementos para construção dos laboratórios é apresentado pela Fig.2.14.

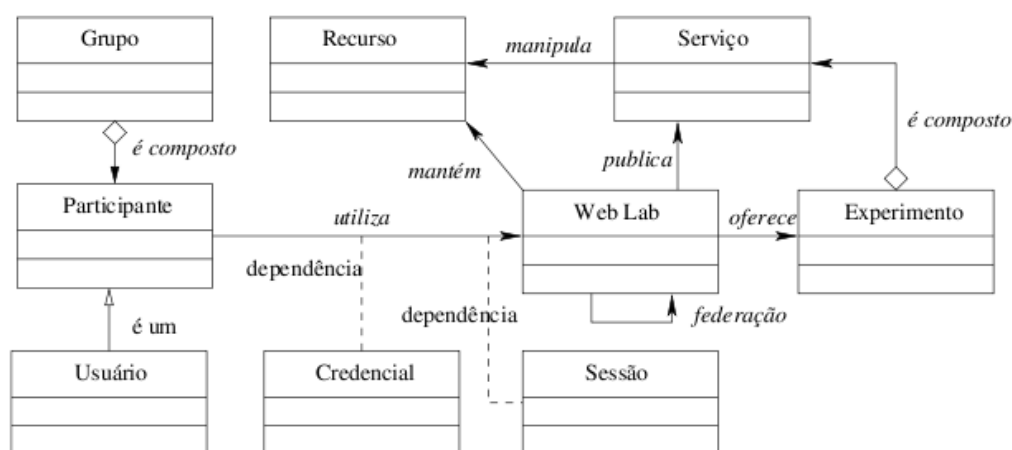


Fig. 2.14: Diagrama para a Criação de Laboratórios [13].

Para acessar o laboratório, é necessário possuir as credenciais, que são atribuídas a um participante que pode ser um usuário ou grupo. Uma vez fornecida a credencial do participante, são criadas sessões que irão possibilitar o uso do WebLab. O WebLab é responsável por manter os recursos e publicar os serviços e experimentos. Para o componente WebLab, pode-se referenciar outro laboratório com a mesma estrutura, o que possibilita a criação de federações de WebLabs. Todo esse modelo referencial é utilizado pelo WebLab, a fim de compor a parte administrativa do laboratório, e faz parte do projeto *AccessService* do GigaBOT.

A arquitetura do NetLab segue o paradigma da computação orientada a serviço, por isso, todos os componentes devem oferecer serviços para possibilitar a interação entre as partes envolvidas. A arquitetura agrupa esses serviços em três sessões: acesso, interação e comunicação, em que cada sessão é responsável em prover um conjunto de serviços específicos de modo que possam ser reutilizados na criação de outros laboratórios. A seguir, são apresentadas as descrições das sessões:

- Sessão de Acesso - fornece serviços para autenticação e autorização dos alunos no laboratório. Compõe a parte gerencial do laboratório;
- Sessão de Interação - provê serviços para interação dos alunos com o experimento ou entre laboratórios;

- Sessão de Comunicação - disponibiliza serviços e mecanismos para a comunicação entre a aplicação de experimento e os equipamentos que fazem parte do WebLab.

A Fig.2.15 ilustra a arquitetura parcial do NetLab e a integração com o *AccessService* no domínio do servidor. Os experimentos são criados em projetos separados do WebLab, contudo são executados no mesmo servidor, o que possibilita a integração entre as aplicações diferentes por meio de serviços da sessão de acesso.

Cada aplicação de experimento é uma aplicação Web e contém os serviços de interação entre o equipamento e o aluno. O protocolo utilizado para comunicação entre a aplicação cliente e o experimento é o SOAP, já a comunicação do experimento com o *Host* do laboratório é realizada por meio de RMI.

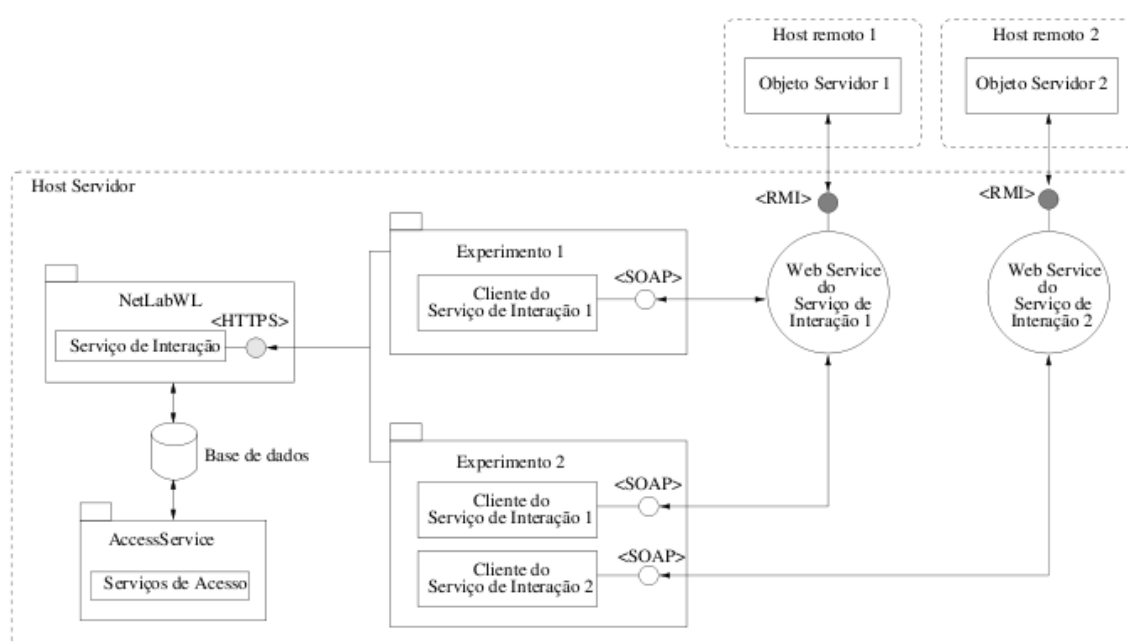


Fig. 2.15: Visão Geral da Arquitetura do NetLab WebLab [4].

Diferentes experimentos podem ser criados por meio de composições de serviços já implementados em outros experimentos. Essa característica permite uma maior reutilização dos serviços e, com isso, maior agilidade na disponibilização de novos experimentos. A Fig.2.16 apresenta a composição dos serviços realizada pela aplicação cliente no domínio do usuário.

A aplicação cliente do experimento é implementada em Java e é carregada para a máquina do aluno por meio da tecnologia *Java Web Start* [14] (JWS). A comunicação da aplicação cliente com o experimento é realizada por meio de requisições SOAP para a aplicação Web referente ao experimento, já a comunicação do serviço do experimento com o *Host* do laboratório é efetuada pelo Protocolo RMI.

Os experimentos no NetLab são compostos de recursos físicos e lógicos. Os recursos físicos são os *Hosts* que fazem parte do laboratório, os lógicos são os serviços que são disponibilizados para o experimento. O NetLab usa essas informações para iniciar e parar os serviços em cada máquina do laboratório. Esses serviços são iniciados por meio de uma fábrica de objeto remoto, que é iniciada ao carregar o Sistema Operacional em cada máquina. O NetLab solicita a criação de um conjunto de objetos remotos referentes a cada experimento. Assim que são criados todos os objetos remotos em todas as máquinas que fazem parte do experimento, o NetLab permite que a aplicação cliente seja

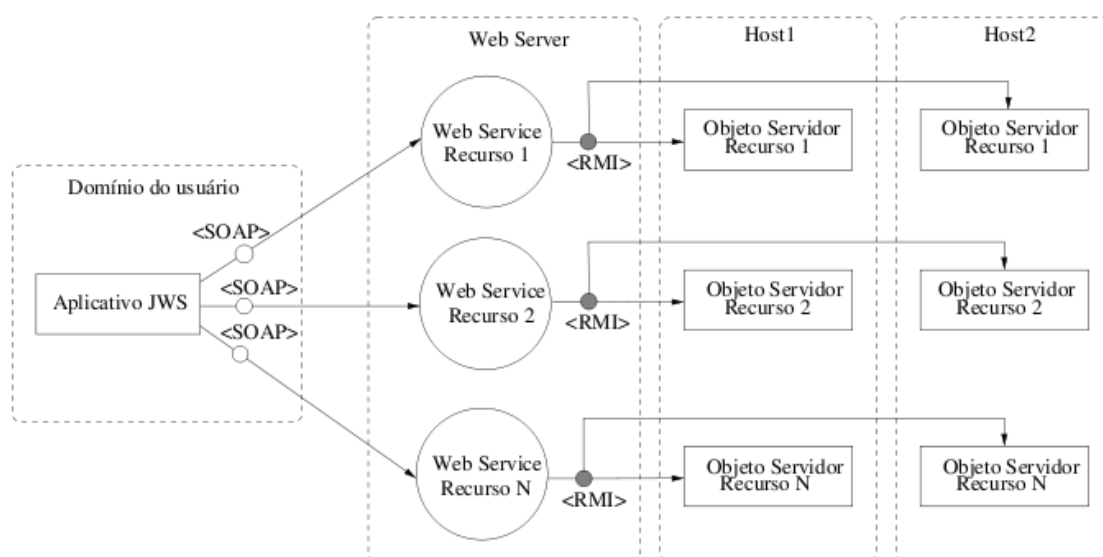


Fig. 2.16: Composição de Serviços do NetLab WebLab [4].

carregada para a máquina do usuário. Os experimentos só podem ser executados mediante prévia reserva, com isso, antes de iniciar o experimento é realizada uma verificação se o aluno tem reserva para executar o experimento selecionado. Com a reserva efetuada, assim que o experimento é iniciado, é criada uma sessão de interação, que é monitorada e, ao seu término, o NetLab solicita a remoção dos objetos remotos do experimento, o que impossibilita a execução dos serviços nas máquinas do laboratório.

### 2.5.1 Experimentos Contemplados

Os experimentos disponibilizados pelo NetLab são todos para o domínio de redes de computadores. Todos os experimentos disponibilizados são reais, ou seja, não são simulações de *software*. São compostos por um conjunto de recursos físicos, que são as máquinas, e os lógicos, que são os serviços dos experimentos.

Na parte gerencial do NetLab, é possível criar vários experimentos mediante composições de recursos diferentes, como exemplo, um determinado experimento pode utilizar todas as máquinas do laboratório, enquanto outro pode utilizar um subconjunto dessas máquinas.

A Fig.2.17 demonstra a execução do experimento de configuração de interface de rede.

Para esse experimento, recorreu-se às as máquinas Gaia e Heilos ambas com duas interfaces de redes. O *host* Gaia está conectado pela *eth1* em Helios na *eth3*. O experimento permite que qualquer interface de rede seja alterada, iniciada ou parada. Esse experimento é composto por outro serviço, o de conectividade (*ping*), com isso, em um único experimento, é possível configurar as interfaces de rede das máquinas e testá-las com o serviço de conectividade. A parte inferior da aplicação é utilizada para apresentar o resultado do teste de conectividade (quando executado).

Outro experimento disponibilizado é o de configuração de rotas. O experimento é composto por três *hosts*, Poseidon, Gaia e Helios. O *host* gaia é o roteador entre os dois outros *hosts*. A Fig. 2.18 ilustra a interface gráfica da aplicação cliente do experimento.

Esse experimento possibilita a utilização dos comandos do Sistema Operacional *Linux*, *ifconfig*, *ping*, *route* e variantes [2]. Cada *host* possui uma tabela de rotas previamente configurada. Ao selecionar o botão Atualizar da aplicação cliente, é enviada uma requisição ao servidor que, por sua vez, solicita as informações de tabela de rotas de cada *host* e retorna para o cliente que, em seguida, exibe

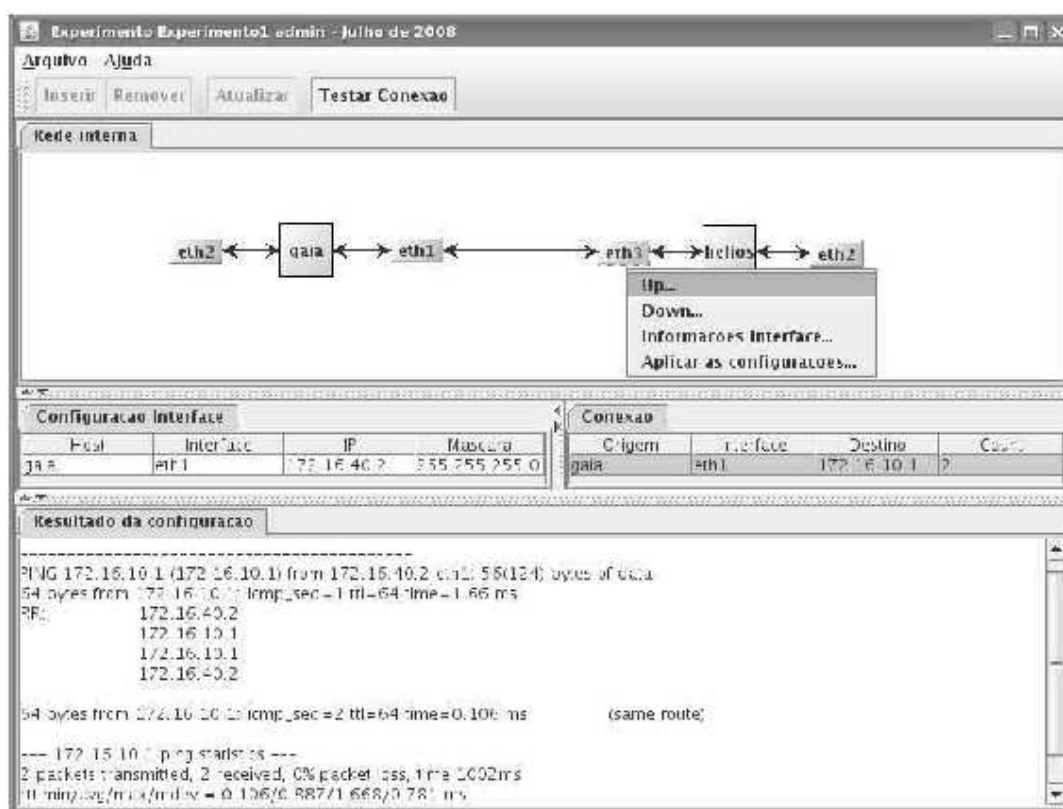


Fig. 2.17: Aplicação Cliente do Experimento de Configuração de NIC [2].

o resultado na aba Rotas. A tabela de rotas pode ser alterada conforme a necessidade na execução do experimento. Isso é possível devido aos serviços de interação que compõem esse experimento, neste caso, o serviço de configuração de tabela de rotas. O propósito desse experimento é a realização de teste de conectividade entre equipamentos de diferentes redes. O teste de conectividade é fornecido pelo serviço de conectividade. Na aba conexão são informados os parâmetros necessários para a execução de teste, e os serviços de conectividade executam o comando *ping* localmente no *host*.

## 2.5.2 Análise da Arquitetura do NetLab WebLab

O NetLab WebLab apresenta uma arquitetura orientada a serviços que acomodam experimentos para o domínio de redes de computadores. A arquitetura possibilita a reutilização dos serviços por meio de composições no domínio do cliente, o que contribui para uma rápida criação de novos experimentos. Agrupar os serviços em aplicações Web diferentes torna a manutenção mais complexa, pois, quando alterado algum serviço que é utilizado em aplicações diferentes, estas devem ser alteradas. O uso do Protocolo SOAP pode comprometer o desempenho por trafegar mais informações do que o necessário (*overhead*).

Embora a maioria dos servidores de aplicações permitam que aplicações Web sejam atualizadas sem que seja necessário reiniciar o servidor, isso na prática nem sempre funciona, causando um problema de indisponibilidade dos serviços, consequentemente, do laboratório.

A disponibilização de um novo experimento implica a criação de uma aplicação Web, e de alterações na fábrica remota de objetos em cada máquina que faça parte do laboratório.

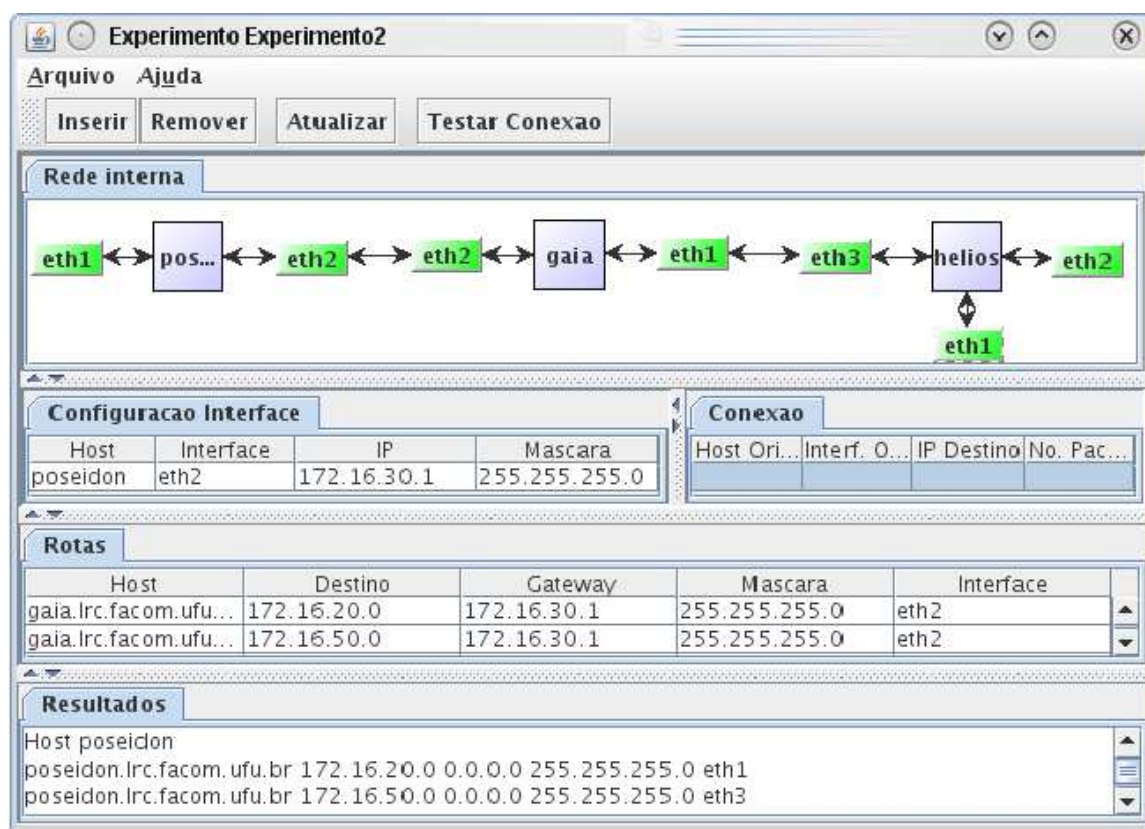


Fig. 2.18: Experimento de Configuração de Rotas [15].

## 2.6 Requisitos Funcionais para WebLabs

Tendo por base a série de trabalhos apresentados, é possível levantar os requisitos essenciais de propósitos gerais e específicos para o domínio de redes de computadores. A Tab. 2.1 apresenta os requisitos de propósitos gerais, sendo comuns para WebLabs independentes do seu domínio. Já a Tab. 2.2 apresenta os requisitos específicos para o domínio de redes de computadores.

Boa parte dos trabalhos analisados utilizam SOA no projeto de seus WebLabs [4, 6, 7, 9], cuja comunicação é baseada na troca de mensagens síncronas ou assíncronas independentes do protocolo de transporte [4]. A sua implementação independe da linguagem de programação ou do sistema operacional. No entanto uma boa forma de prover interoperabilidade e não romper com a política de segurança de um dado domínio é a utilização de protocolos como o HTTP (*Hypertext Transfer Protocol*). Por isso, alguns laboratórios optam por trabalhar com *Web Services* e utilizam tecnologias como JWS (*Java Web Start*), *ActiveX*, *Flash*, *Java Applet* para tornar a interação com o usuário mais amigável e facilitar a utilização dos laboratórios e dos experimentos disponibilizados.

Mesmo a aplicação cliente usando o protocolo HTTP, a infraestrutura interna de um WebLab pode conter diversas máquinas sobre várias plataformas comunicando entre si. Isso implica a utilização de diversos protocolos de comunicação interligando-os (p.ex., Deusto [16]). Alguns laboratórios [6, 8] permitem que os experimentos sejam executados em *background*, evitando a espera pelo término da execução de um dado experimento.

A disponibilização em múltiplos idiomas propicia que outras instituições em outros países acessem o laboratório, possibilitando maior integração entre instituições. Mas, entre aqueles apresentados, apenas o Deusto possui essa funcionalidade.

Há alguns requisitos que são específicos de um WebLab no domínio de redes de computadores

|                                                                  | WebLabs |      |     |       |        |
|------------------------------------------------------------------|---------|------|-----|-------|--------|
| Requisitos Funcionais                                            | NetLab  | iLab | SIT | DACSE | Deusto |
| arquitetura baseada em serviços                                  | X       | X    | X   | X     | X      |
| interoperabilidade                                               | X       | X    | X   | X     | X      |
| tecnologias <i>applet</i> , <i>flash</i> , <i>activeX</i> , etc. | X       | X    | X   | X     | X      |
| suporte a múltiplos protocolos                                   |         |      |     | X     | X      |
| disponibilidade 24x7                                             |         |      |     |       | X      |
| execução de experimentos <i>background</i>                       |         | X    | X   |       |        |
| arquitetura do WebLab em camadas                                 | X       |      |     |       |        |
| padronização dos experimentos                                    | X       |      |     |       |        |
| agendamento de uso do WebLab                                     | X       |      |     |       |        |
| suporte a múltiplos idiomas                                      |         |      |     |       | X      |
| federação entre WebLabs                                          |         |      |     |       | X      |
| maior grau de escalabilidade                                     |         |      |     |       |        |

Tab. 2.1: Requisitos Gerais por WebLab

|                                   | WebLabs |      |     |       |        |
|-----------------------------------|---------|------|-----|-------|--------|
| Requisitos Funcionais             | NetLab  | iLab | SIT | DACSE | Deusto |
| acompanhamento em tempo real      | X       | X    | X   | X     | X      |
| suporte a trabalho colaborativo   | X       | X    |     |       |        |
| configuracao em tempo de execução |         |      |     |       |        |
| compartilhamento de recursos      |         |      |     |       |        |
| multi usuários                    | X       |      |     |       |        |

Tab. 2.2: Requisitos Específicos do Domínio de Redes de Computadores

que não são necessariamente considerados por outros WebLabs. Uma característica contemplada pelos laboratórios descritos é o acompanhamento em tempo real, que permite ao usuário a execução, a manutenção e o gerenciamento do experimento no laboratório em tempo de execução.

O suporte a trabalho colaborativo é uma característica interessante do laboratório, pois possibilita o trabalho em equipe. Enquanto o experimento é executado por um participante, os outros podem assistir a ele. Como nem todos os experimentos utilizam os mesmos recursos, a execução dos experimentos pode ser realizada por vários usuários simultâneos (multi-usuários).

## 2.7 Considerações Finais

Com base na Tabs. 2.1 e 2.2, observam-se algumas tendências com relação aos trabalhos apresentados. Todos os WebLabs possuem uma arquitetura baseada em *Web Services*. O suporte a *Web Services* permite aos WebLabs maior interoperabilidade, sendo esse outro requisito essencial para integrar e criar federações de WebLabs. Outros dois requisitos que também são acomodados por todos os WebLabs são os de acompanhamento em tempo real dos experimentos e uso de tecnologias como *JWS*, *ActiveX*, *Flash*, *Java Applet* na aplicação cliente para proporcionar maior interatividade. O WebLab Deusto acomoda três requisitos, que não são implementados nos outros WebLabs, disponibilidade 24x7, internacionalização e suporte a federação.

## Capítulo 3

# Arquitetura Proposta para WebLab

Este capítulo, apresenta a arquitetura que permite a disponibilização de experimentos de maneira dinâmica e modular, a fim de acomodar os requisitos funcionais levantados no capítulo anterior. Essa arquitetura possibilita que experimentos sejam criados em forma de módulos. Esses módulos seguem a especificação OSGi, por este motivo, a arquitetura define componentes que envolvem o *framework* OSGi, a fim de prover a modularização dos experimentos. Uma vez modularizados os experimentos, a arquitetura é responsável por interagir de modo dinâmico com os serviços contidos nos módulos, tornando-os públicos, o que propicia que sejam consumidos por aplicações clientes.

A arquitetura define a interface responsável por receber as solicitações de execuções de serviço contudo, não especifica como deve ser implementada. Assim, a definição de como receber as solicitações fica por conta da implementação da interface, o que torna a arquitetura mais genérica podendo suportar vários meios de solicitações de execuções de serviços.

A arquitetura tem como referência o NetLab WebLab apresentado por Rocha [4], que segue o modelo do Projeto GigaBOT [5], que é um modelo conceitual para a criação de WebLabs em SOA, pelo qual os experimentos são criados em forma de serviços.

### 3.1 Modelo de referência para WebLab

O NetLab WebLab utiliza como base a Arquitetura SOA. Essa arquitetura oferece uma estrutura baseada em serviços, nesse caso como pilares de sustentação de um WebLab. Para que um WebLab tenha uma boa interação, são necessárias, no mínimo, três classes de sessões [17]. Essas sessões contem serviços específicos para prover as funcionalidades necessárias tanto para acessar o laboratório quanto para a interação entre os experimentos, bem como, para a comunicação entre as partes envolvidas é realizada [2, 4, 17]. A Fig. 3.1 ilustra os serviços de cada uma das sessões por meio do diagrama de pacotes da UML (*Unified Modeling Language*).

#### 3.1.1 Sessão de Acesso

O acesso ao laboratório remoto se dá pela sessão de acesso, que fornece serviços que controlam e restringem o uso do laboratório e dos experimentos, compondo, assim, a parte gerencial do WebLab. Esse controle e restrição são feitos mediante: a) autenticação, que é a validação das credenciais fornecidas para acessar o laboratório e b) autorização, que são os conjuntos de regras e validações realizadas ao acessar um determinado experimento ou recurso no laboratório. O acesso ao laboratório pode ser concedido, entretanto só é permitido o uso dos experimentos se a determinada credencial estiver autorizada.

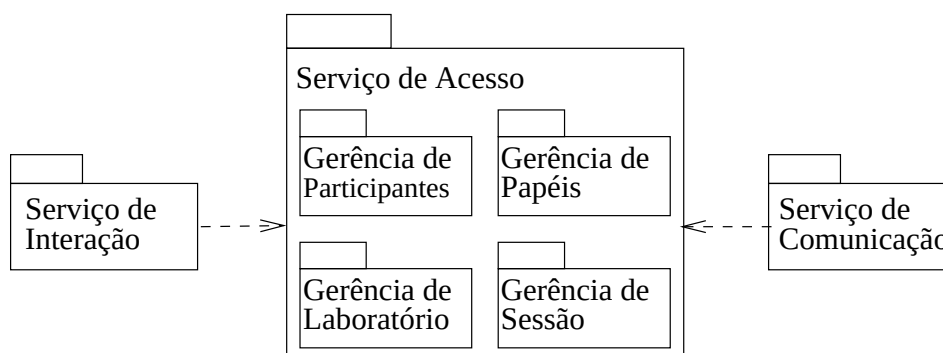


Fig. 3.1: Diagrama de Pacotes Representando os Serviços de Cada Sessão [2].

O NetLab WebLab possui recursos físicos, como *host*, e sub-recursos, como interface de rede, e lógicos, que são aplicativos distribuídos nos *hosts*. Um experimento pode ser composto por vários recursos físicos e lógicos. Alguns dos recursos não podem ser utilizados simultaneamente, por esse motivo é necessário que o experimento seja agendado, evitando, desse modo, que o mesmo experimento seja executado simultaneamente, quando o recurso não pode ser compartilhado. O agendamento de uso do experimento é realizado por serviços contidos na sessão de acesso. A criação de uma sessão de acesso, que controla a execução dos experimentos, possibilita que ele seja encerrado ao final do prazo de agendamento [4, 17].

### 3.1.2 Sessão de Interação

Serviços de interação, contidos na sessão de interação, são particulares de cada WebLab, pois, por meio desses serviços é que estão disponíveis ferramentas e funcionalidades essenciais para a execução dos experimentos. A sessão de interação também é necessária para controlar quais recursos devem ser preparados para o uso de um experimento, pois, tais recursos devem ser disponibilizados para os experimentos antes que a aplicação cliente seja carregada. O NetLab WebLab tem como domínio redes de computadores, por esse motivo, os serviços contidos nessa camada são específicos para experimentos relacionados a redes de computadores, como:

- a) serviço de retaguarda - provê os recursos necessários para a execução dos experimentos, como informações sobre recursos e sub-recursos dos *hosts* do laboratório, e também garante a conectividade do laboratório com uma rede de retaguarda;
- b) serviço fábrica RMI (*Remote Method Invocation*) - ao ser executado um experimento, antes de carregar a aplicação cliente, a fábrica RMI é responsável por instanciar todos os objetos servidores nos *hosts* que fazem parte do experimento em questão. Esses objetos servidores são as aplicações distribuídas nos *hosts* do laboratório, responsáveis por interagir com os recursos e sub-recursos da máquina;
- c) serviço de conectividade - este serviço permite testar a conectividade dos *hosts* que fazem parte do experimento;
- d) serviço configuração de interface de rede - possibilita alterar as configurações das NICs dos *hosts*;
- e) serviço de configuração de tabela de rotas - permite configurar a tabela de rotas em cada máquina que faz parte do laboratório;
- f) serviço de configuração de rotas dinâmicas - provê a configuração dinâmica de rotas nas máquinas do laboratório.

### 3.1.3 Sessão de Comunicação

Na sessão de comunicação, os serviços de comunicação fornecem mecanismos para sustentar a troca de informações entre as demais sessões. Nessa sessão, são fornecidas ferramentas úteis, como, por exemplo, ferramentas para controle de qualidade da rede, e para monitorar a troca de mensagens e se há alguma anormalidade na rede. Por se tratar de um WebLab, pode haver comunicação multi-mídia em tempo real para a transmissão da execução de experimentos, comunicação em grupo para a recepção de transmissão dos experimentos, ou seja, formas de comunicação que devem ser suportadas [17].

Alguns experimentos para laboratórios específicos necessitam de acompanhamento em tempo real, como por exemplo, laboratório de robótica. Para poder acompanhar o uso de um robô, são necessárias, no mínimo, duas câmeras, uma para acompanhar o robô e a outra do próprio robô. Nesse caso, a transmissão em tempo real é essencial.

Os serviços de comunicação em grupo também são providos por essa sessão. Esses serviços já não são específicos do domínio do laboratório. O acompanhamento da execução de um determinado experimento por um grupo propicia que uma aula seja acompanhada por vários alunos ao mesmo tempo, fazendo uso de *multicast* para a transmissão da execução do experimento.

## 3.2 Arquitetura com Suporte a Módulos Dinâmicos

A arquitetura proposta é dividida nos domínios cliente e servidor. O domínio do cliente é composto por um único componente, que é a aplicação do experimento, ou seja, é responsável por prover a interação com os serviços disponibilizados para o experimento. O domínio do servidor, por sua vez, se subdivide em dois domínios: Servidor NetLab e *Hosts*. O domínio do Servidor NetLab é responsável por acomodar os módulos de experimentos e publicar os serviços e prover a execução dos serviços contidos nos módulos. Os componentes que constituem esse domínio são: a) *Dispatcher*, b) *ServiceManager*, c) *Core* e d) *framework* OSGi (Fig. 3.2). Já o domínio dos *Hosts* é composto pelas aplicações distribuídas que são responsáveis por interagir com os recursos e sub-recursos da máquina.

O domínio Servidor NetLab pode ser composto por vários servidores Web, onde cada servidor é um nó contendo a mesma estrutura de componentes da arquitetura, dessa forma, é possível aumentar a escalabilidade horizontal do WebLab incluindo novos servidores nesse domínio. A divisão dos domínios permite que uma alteração em um domínio não tenha impacto no outro, o que permite que um novo equipamento seja incluído no domínio dos *Hosts* de forma transparente para o domínio do Servidor NetLab, da mesma forma ao incluir um novo nó no domínio do Servidor NetLab. O domínio do Laboratório foi definido para agrupar o domínio dos *Hosts* e o domínio Servidor NetLab.

A comunicação entre os domínios cliente e servidor é realizada utilizando *Web services REST*. No domínio do servidor, todos os componentes estão localizados na mesma máquina, por esse motivo, não é necessária a utilização de protocolos e, sim, chamadas de APIs (*Application Programming Interface*) seguindo o contrato estabelecido pelas interfaces dos componentes. A comunicação entre o domínio do Servidor NetLab e o domínio dos *Hosts* é realizada utilizando o Protocolo RMI (*Remote Method Invocation*).

A aplicação do protocolo RMI é de responsabilidade do serviço contido no módulo de experimento, mas não fica limitado a esse protocolo, pois nada impede que um serviço recorra a outro protocolo como *SOAP* ou *REST*. Mas, para isso, é preciso que os componentes distribuídos nos *hosts* também atendam ao protocolo em questão. A Fig. 3.3 apresenta a divisão dos domínios e os protocolos utilizados entre estes.

A contribuição dessa nova arquitetura está em boa parte no domínio do servidor e atua diretamente na sessão de interação, fornecendo mecanismos para a publicação dos módulo de experimentos e

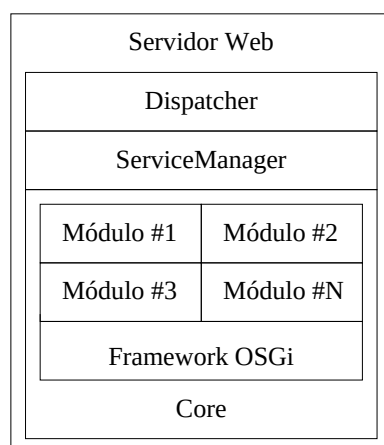


Fig. 3.2: Diagrama Bloco dos Componentes que Constituem a Arquitetura Proposta.

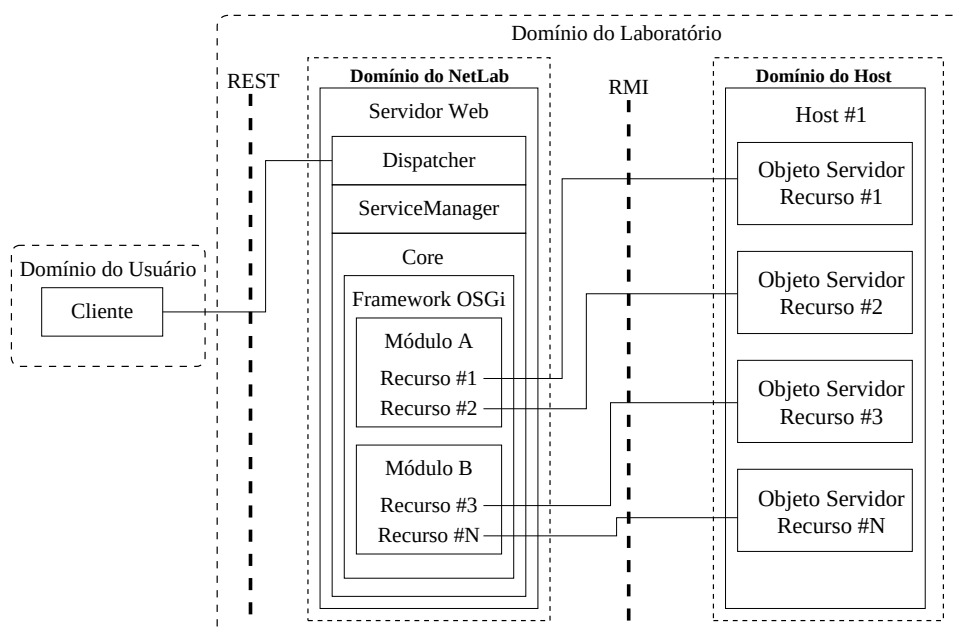


Fig. 3.3: Arquitetura Proposta com Suporte a Módulos Dinâmicos para WebLabs.

execução dos serviços contidos nesses módulos de forma dinâmica. A Fig. 3.4 ilustra o mesmo diagrama de pacotes apresentado na Fig. 3.1, incluindo, no pacote da sessão de interação, o diagrama de classes referentes aos componentes da arquitetura.

Na sessão de acesso, foi incluído um gerenciador de sessão de experimentos. A criação da sessão de interação é delegada para esse gerenciador, que cria a sessão e adiciona a sessão em um monitor, que é responsável por verificar o tempo de vida de cada sessão criada. A sessão pode ser finalizada por dois modos, um quando o usuário conclui o experimento, o outro, quando o tempo de reserva de uso do experimento expira, neste caso, o monitor de sessão notifica o gerenciador que, por sua vez, finaliza a sessão. Uma vez finalizada, esta não pode mais ser utilizada, impedindo que qualquer serviço de experimento seja executado no laboratório.

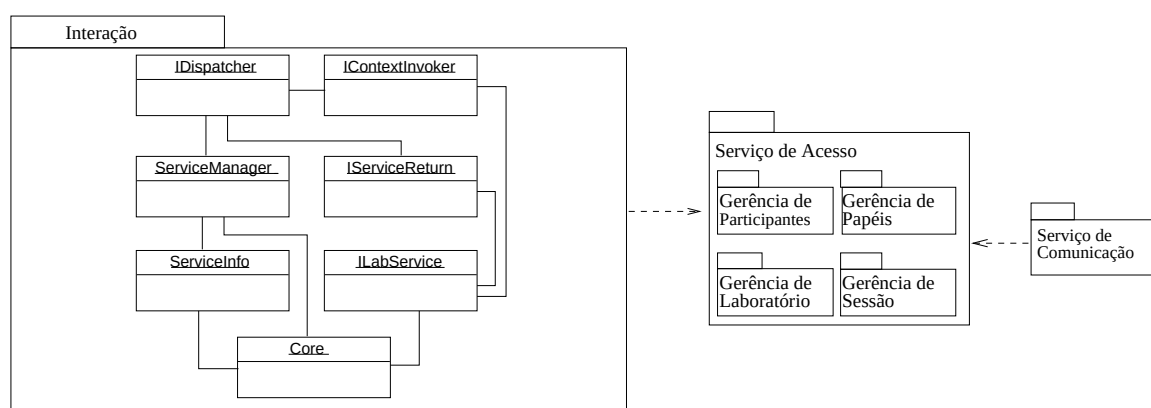


Fig. 3.4: Diagrama Pacotes com as Classes dos Componentes.

### 3.3 Descrição dos Componentes da Arquitetura Proposta

#### 3.3.1 Componente *Dispatcher*

O *Dispatcher* é o componente responsável por prover a interação entre a aplicação cliente e os serviços dos módulos de experimentos, ou seja, é o componente que recebe as requisições com solicitações de execuções de serviços. Esse componente foi criado para evitar o acoplamento do *ServiceManager* com o recebimento de requisições. Como exemplo, supondo que o *Dispatcher* não fosse incluído na arquitetura e a aplicação cliente executasse os serviços por meio do protocolo SOAP, isso causaria o acoplamento do *ServiceManager* com uma implementação da pilha SOAP. O mesmo acoplamento do exemplo também ocorreria, se a aplicação tivesse de suportar serviços REST. No caso do exemplo, seguindo a arquitetura proposta, o *ServiceManager* é desacoplado da implementação do *Dispatcher*, por isso, não tem ligação com tipo de protocolo algum.

O *Dispatcher* tem a função de extrair os parâmetros da requisição e sua URI (*Uniform Resource identifier*). De posse dessas informações, a execução do serviço pode ser solicitada para o *ServiceManager*. Dessa forma, no caso do exemplo, para atender às requisições SOAP e REST, basta criar uma implementação de *Dispatcher* para cada um desses protocolos em que cada implementação vai possuir a mesma referência do *ServiceManager*.

Na sequência, são apresentadas as definições das interfaces necessárias para o componente *IDispatcher* seguidas de uma breve descrição de cada um dos métodos.

Para todo serviço executado, é necessário criar um contexto. O contexto contém informações essenciais para a execução do serviço. Por isso, o *Dispatcher* deve criar esse contexto de forma dinâmica e fazer com que este chegue ao serviço. Nesse caso, a interface *IContextInvoker* define o método para esse contexto, conforme a interface:

```

1
2 package br.ufu.facom.netlab.service;
3
4 import java.io.OutputStream;
5 import java.util.Map;
6 import org.jdom.Element;
7 import br.ufu.facom.netlab.core.ServiceInfo;
8
9 public interface IContextInvoker {
10     String getServiceName();
11     String getContext();
12     void addParameter(String name, Object value);

```

```
13 void addParameters(Map<String, Object> params);
14 Object getParameter(String name);
15 Element getRequestElementRoot();
16 ServiceInfo getServiceInfo();
17 OutputStream getOutputStream();
18 }
```

---

Listagem 3.1: Interface IContextInvoker.java**Descrição dos métodos da interface *IContextInvoker*:**

- *getServiceName()* - retorna o nome do serviço que será executado;
- *getContext()* - cada serviço é mapeado para um contexto, o que permite que serviços com o mesmo nome possam ser publicados em contextos diferentes, esse método retorna o contexto ao qual o serviço está mapeado;
- *addParameter()* - possibilita adicionar parâmetro ao contexto;
- *addParameters()* - adiciona, de uma única vez, vários parâmetros ao contexto;
- *getParameter()* - retorna o conteúdo de um parâmetro associado a uma chave;
- *getRequestElementRoot()* - retorna um elemento *JDOM*, representando a raiz de um XML;
- *getServiceInfo()* - retorna dados do serviço, agrupa as informações do serviço em um objeto.

A interface *IDispatcher* é definida para o componente *Dispatcher* da arquitetura. O trecho de código a seguir ilustra os métodos declarados pela interface.

---

```
1
2 package br.ufu.facom.netlab.core;
3
4 import java.io.InputStream;
5 import java.io.OutputStream;
6 import java.util.Map;
7 import org.jdom.Element;
8 import br.ufu.facom.netlab.service.IContextInvoker;
9 import br.ufu.facom.netlab.service.IServiceReturn;
10
11 public interface IDispatcher {
12     public IServiceReturn execute(IContextInvoker ctx) throws Exception;
13     public IContextInvoker buildContextInvoker(String requestURI,
14         Map<String, Object> params, Element xml, OutputStream out) throws
15         Exception;
16     public Element buildXmlFromRequest(InputStream input, int size) throws
17         Exception;
18     public void writeResultToOutput(OutputStream output, Element docRet) throws
19         Exception;
20 }
```

---

Listagem 3.2: Interface IDispatcher.java

**Descrição dos métodos da interface *IDispatcher*:**

- *execute()* - método principal do componente, todo processo de execução de um serviço se inicia na execução desse método;
- *buildContextInvoker()* - extrai informações da *URI* da requisição e parâmetros e cria o contexto;
- *buildXmlFromRequest()* - método responsável em criar um XML a partir dos dados dos informados na requisição;
- *writeResultToOutput()* - após finalizada a execução do serviço, esse método grava o retorno da execução, finalizando o ciclo de execução de um serviço.

### 3.3.2 Componente *ServiceManager*

O componente *ServiceManager* é responsável por fornecer as funcionalidades relacionadas à execução dos serviços. O *ServiceManager* abstrai as rotinas de gerenciamento de serviços do componente *Core*. Essa abstração o torna dependente desse componente, contudo não evidencia dependência do componente acima, que é o *Dispatcher*. Dessa forma, qualquer outro componente pode executar os serviços publicados pela arquitetura, o que causa um fraco acoplamento entre os componentes, e ainda, torna a arquitetura mais genérica, de forma que possa se adaptar aos vários domínios diferentes de WebLabs.

Na arquitetura proposta, foi definido o componente *Dispatcher* que interage como *ServiceManager*. A arquitetura provê a execução de serviços de forma dinâmica para todo serviço que implementar a interface *ILabService*. Os métodos dessa interface são listados no trecho de código abaixo.

---

```
1
2 package br.ufu.facom.netlab.service;
3 public interface ILabService {
4     public String getName();
5     public String getDescription();
6     public ServiceReturn execute(IContextInvoker ctx) throws Exception;
7     public String getContext();
8 }
```

---

Listagem 3.3: Interface *ILabService.java*

#### Descrição dos métodos da interface *ILabService*:

- *getName()* - retorna o nome do serviço. Esse nome é utilizado para compor a chave para localizar o serviço;
- *getDescription()* - retorna uma descrição com mais informações sobre o serviço. Essa descrição pode conter uma descrição detalhada das funcionalidades que o serviço provê;
- *execute()* - método responsável por executar o serviço propriamente dito;
- *getContext()* - retorna o nome do contexto ao qual o serviço está associado. Esse contexto, assim como o nome, é empregado para compor a chave para localização do serviço no *ServiceManager*.

A interface *ILabService* fornece métodos com informações referentes aos serviços que serão publicados. Quando o *Core* recebe uma notificação da classe que implementa essa interface, o mesmo,

adiciona o serviço no *ServiceManager*, que, por sua vez, o disponibiliza para o *Dispatcher*. Uma vez adicionado o serviço, este já está disponível para execução.

Conforme a interface, todo serviço deve fornecer o contexto do qual o serviço faz parte. Dessa forma, é possível ter várias classes de serviços para um único contexto, o que permite que módulos diferentes publiquem serviços para um mesmo contexto. O *ServiceManager* usa os métodos de contexto e nome para compor a *URI* que serve para localizar o serviço. Como exemplo, uma implementação que retorne como contexto e nome, respectivamente, *vlan*, *config*, a *URI* para o registro do serviço fica da seguinte forma: *vlan/config*. Um exemplo de URL *REST* para executar esse serviço ficaria assim: *http://servidor/NetLabWL/vlan/config*.

A execução de um determinado serviço pode resultar em vários tipos de retornos, pois a arquitetura proposta não limita o retorno a apenas XML. Por esse motivo, foi definida uma interface denominada *IServiceReturn*, que é responsável por acomodar o retorno da execução de um serviço. A interface *IServiceReturn* é apresentada a seguir:

---

```
1
2 package br.ufu.facom.netlab.service;
3
4 import org.jdom.Element;
5
6 public interface IServiceReturn {
7     Element getElement();
8     boolean isWritedOutputStream();
9     String getContentType();
10    void setContentType(String type);
11 }
```

---

Listagem 3.4: Interface *IServiceReturn.java*

#### Descrição dos métodos da interface *IServiceReturn*:

- *getElement()* - no caso do serviço executado retornar um XML, esse método retorna o elemento raiz do XML;
- *isWritedOutputStream()* - um determinado serviço pode gravar diretamente no *stream* de saída passado como parâmetro no contexto de execução, nesses casos, esse método retorna verdadeiro, a fim de informar para uma implementação do *Dispatcher* que não há elemento XML retornado;
- *getContentType()* - retorna o tipo de conteúdo que foi gravado no fluxo de saída;
- *setContentType()* - informa qual o tipo de conteúdo foi gravado no fluxo de saída.

O componente *ServiceManager* é uma classe concreta, pois possui características específicas da arquitetura, por isso, não pode ser definido como uma interface. O trecho de código abaixo apresenta somente os métodos públicos da classe que são utilizados pelo *Core* e o *Dispatcher*.

---

```
1
2 package br.ufu.facom.netlab.core;
3
4 import java.util.Collection;
5 import java.util.Map;
6 import java.util.TreeMap;
```

```
7 import java.util.TreeSet;
8 import org.jdom.Document;
9 import org.jdom.Element;
10 import br.ufu.facom.netlab.service.IContextInvoker;
11 import br.ufu.facom.netlab.service.IServiceReturn;
12
13 public class ServiceManager {
14 ...
15     public static ServiceManager getInstance() {...}
16     void addService(String context, String serviceName) {...}
17     void removeService(String context, String serviceName) {...}
18     public ServiceInfo validateAndBuildServiceInfo(String uri) throws
        Exception {...}
19     IServiceReturn executeServiceDelegate(IContextInvoker ctx) throws
        Exception {...}
20     public Element buildElementError(Exception e) {...}
21
22 ...
23 }
```

---

Listagem 3.5: Trecho de código da classe *ServiceManager.java*

#### Descrição dos métodos públicos da classe *ServiceManager*:

- *getInstance()* - este componente por ser compartilhado entre diversos *Dispatchers*, foi definido como uma classe *Singleton*, esse método retorna a instância do *ServiceManager*;
- *addService()* - este método adiciona um novo serviço na lista de serviços disponíveis, uma vez adicionado, o serviço já pode ser executado pelo *Dispatcher*;
- *removeService()* - remove um serviço da lista;
- *validateAndBuildServiceInfo()* - este método cria uma instância da classe *ServiceInfo* que contém todas as informações sobre o serviço, como descrição, nome, contexto;
- *executeServiceDelegate()* - método principal da interface, é responsável em delegar a execução do serviço ao *Core*;
- *buildElementError()* - em caso de algum erro, este método cria um elemento XML.

Outra função do *ServiceManager* é validar a sessão de interação. Para todo serviço, antes de sua execução, é realizada uma validação da sessão de interação. Caso não exista uma sessão válida, a execução do serviço não é permitida. Essa validação evita que aplicações não autorizadas possam executar serviços sem ter iniciado uma sessão.

A integração com a sessão de acesso é realizada por meio do *SessionManger* que é o componente da arquitetura responsável por gerenciar as sessões de interação.

### 3.3.3 Componente *Core*

O *Core* é o componente central da arquitetura no provimento da execução de serviços para as sessões de interação e comunicação, além de iniciar e controlar todos os outros componentes. Sendo assim, a interação com os módulos dos experimentos é controlada pelo *Core*, possibilitando a centralização, mas com a execução dos serviços nos vários *hosts* do laboratório distribuída [15]. Esse componente é responsável por interagir como *framework OSGi*, para disponibilizar os módulos de

experimentos e serviços e fornecer para o componente acima (*ServiceManager*) rotinas que propiciam a execução, publicação, remoção e validação de serviços. Ele envolve o *framework* para evitar acesso direto aos módulos do contêiner *OSGi*, permitindo controlar quais módulos podem ou não ser disponibilizados no contêiner.

O *Core* recebe todas as notificações do contêiner *OSGi* e, por meio dessas notificações, seleciona quais serviços devem ser adicionados ao *ServiceManager*. Basicamente, em todo serviço que implementa a interface *ILabService* é adicionada, contudo não fica limitado a apenas essa interface. Como um módulo pode conter vários serviços, qualquer serviço que for exportado pelo módulo pode ser utilizado por outro. Nesse caso, o serviço não é publicado dinamicamente pela arquitetura, mas pode ser utilizado por outros módulos no contêiner. Assim como o *ServiceManager*, o *Core* é uma classe concreta. Abaixo, são listados os métodos públicos da classe.

---

```
1
2 package br.ufu.facom.netlab.core;
3
4 import java.io.BufferedReader;
5 import java.io.InputStreamReader;
6 import java.net.URL;
7 import java.util.ArrayList;
8 import java.util.Collection;
9 import java.util.HashMap;
10 import java.util.Iterator;
11 import java.util.Map;
12
13 import org.osgi.framework.Bundle;
14 import org.osgi.framework.BundleContext;
15 import org.osgi.framework.Constants;
16 import org.osgi.framework.ServiceEvent;
17 import org.osgi.framework.ServiceListener;
18 import org.osgi.framework.ServiceReference;
19 import org.osgi.framework.Version;
20 import org.osgi.framework.launch.Framework;
21 import org.osgi.framework.launch.FrameworkFactory;
22
23 import br.ufu.facom.netlab.service.IContextInvoker;
24 import br.ufu.facom.netlab.service.ILabService;
25 import br.ufu.facom.netlab.service.ServiceReturn;
26
27 public final class Core {
28     ...
29
30     public void init() throws Exception {...}
31     public void init(Map properties) throws Exception {...}
32     public ServiceReturn executeService(IContextInvoker ctx) throws Exception
33         {...}
34     public Core getInstance() throws Exception {...}
35     public void stop() throws Exception {...}
36
37     private class ServiceListenerImpl implements ServiceListener {
38         public void serviceChanged(ServiceEvent e) {...}
39     }
40
41     ...
42
43 }
```

Listagem 3.6: Trecho de código da classe Core.java

**Descrição dos métodos públicos da classe Core:**

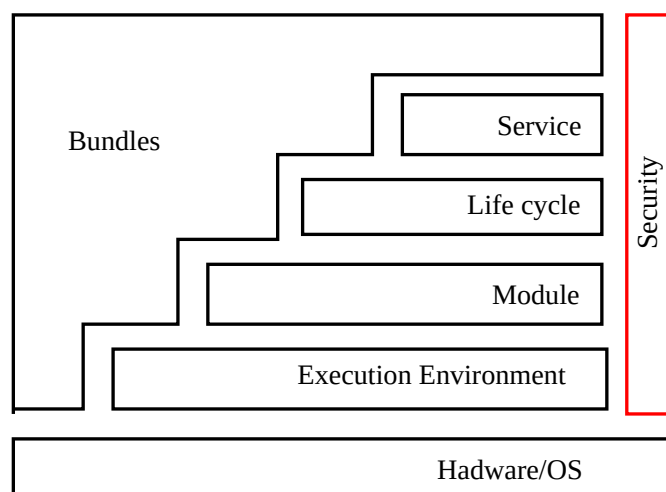
- *init()* - principal método da interface, é responsável por iniciar todos os componentes;
- *executeService()* - método utilizado pelo componente *ServiceManager*, responsável por executar o serviço;
- *getInstance()* - após ser iniciado, este método pode ser executado e retorna a instância do componente;
- *stop()* - finaliza o *framework* OSGi e o *ServiceManager*, após executado esse método, nenhum serviço pode mais ser executado;

A interface *ServiceListener*, fornecida pela especificação OSGi, é utilizada para receber notificações do *framework* OSGi sobre serviços de experimentos. Essa interface define um único método: *serviceChanged()*, que é usado quando um serviço é disponibilizado, alterado ou removido do contêiner.

**3.3.4 Framework OSGi**

O OSGi fornece as primitivas padronizadas, permitindo que aplicações sejam construídas a partir de componentes pequenos e reutilizáveis de forma dinâmica. É definido por uma especificação a *OSGi Service Platform* [18] e é responsável por gerenciar o ciclo de vida dos módulos.

A especificação do *framework* divide as funcionalidades em camadas, conforme exposto na Fig. 3.5 que ilustra como são divididas as camadas do *framework*. Em seguida, são apresentadas as funcionalidades providas pelas camadas.

Fig. 3.5: Camadas do *Framework* OSGi [18].

### Security

A camada de segurança é uma camada opcional e segue a especificação *Java 2 Security Architecture* [19], incluindo novas funcionalidades para o controle dos módulos. Essa camada provê, por meio de assinatura digital, a autenticação do módulo, que propicia o gerenciamento das permissões dos *bundles* por localização ou por assinante do arquivo *jar*. As permissões podem ser atribuídas selecionando os campos contidos no certificado digital X.509 da assinatura, como *Subject Name*, *Issuer Name* ou *Distinguished Names*.

### Execution Environment

A especificação do *framework* foi criada levando em consideração as JVMs (*Java Virtual Machine*), por isso, é possível ter uma implementação do *framework* para cada VM, incluindo versões de cada uma, a Tab. 3.1 apresenta os nomes e as descrições de exemplos de ambientes de execuções suportados pelo *framework*.

| Nome                      | Descrição                                           |
|---------------------------|-----------------------------------------------------|
| OSGi/Minimal-1.2          | OSGi EE minimal implementation of an OSGi Framework |
| j2SE-1.2                  | Java 2 SE 1.2.x                                     |
| j2SE-1.3                  | Java 2 SE 1.3.x                                     |
| j2SE-1.4                  | Java 2 SE 1.4.x                                     |
| j2SE-1.5                  | Java 2 SE 1.5.x                                     |
| j2SE-1.6                  | Java 2 SE 1.6.x                                     |
| CDC-1.1/PersonalBasis-1.1 | J2ME Personal Basis Profile                         |
| CDC-1.1/PersonalJava-1.1  | J2ME Personal Java Profile                          |

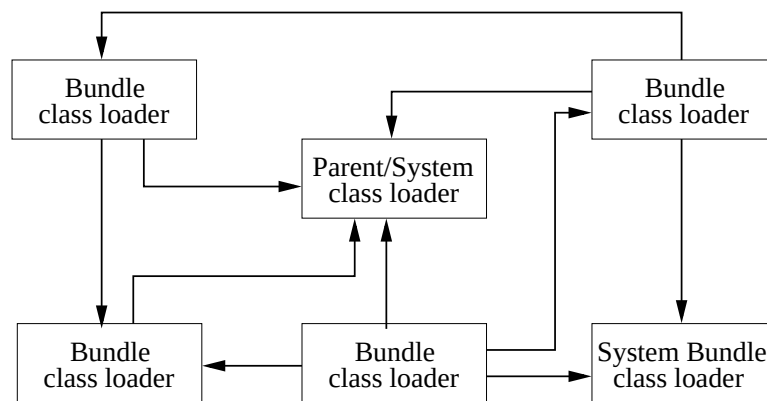
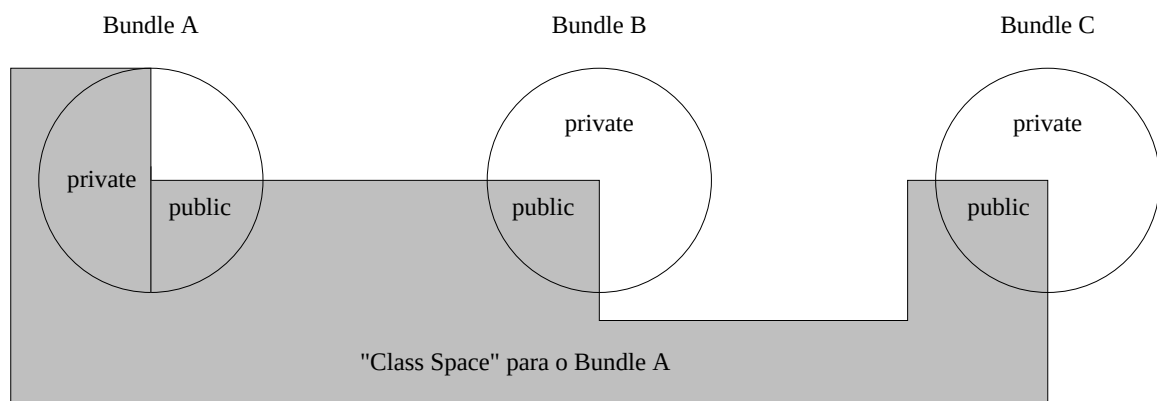
Tab. 3.1: Exemplo de ambientes de execuções [18]

O ambiente de execução pode ser especificado em uma propriedade no arquivo de *manifest* do *bundle*, a propriedade: *Bundle-RequiredExecutionEnvironment* lista em quais ambientes o *bundle* pode ser disponibilizado. Essa propriedade é verificada pelo *framework* e, no caso deste não suportar algum dos ambientes listados na propriedade, a disponibilização do *bundle* não é realizada.

Uma vez disponibilizado o *bundle*, este é compartilhado para toda VM, por isso, essa camada define um mecanismo de carga de classes que permite que classes do *bundle* possam ser compartilhadas ou não, em casos de pacotes privados. O mecanismo responsável por prover esse controle é denominado *Class Loader Delegation*. A Fig. 3.6 ilustra como é realizado o controle.

O controle por delegação possibilita a criação de um compartilhamento de classes, denominado na especificação de *Class Space*. O *Class Space* é composto por todas as classes exportadas dos *bundles*. Seguindo o modelo de delegação, quando um *bundle* possui uma dependência de determinado pacote ou classe, o *classloader* do *bundle* procura resolver a dependência, inicialmente, no próprio *classloader* do *bundle*, caso não seja resolvida, o *framework* a delega, então, para o *classloader* pai. A Fig. 3.7 ilustra o *Class Space* com três *bundles* como exemplo.

Nessa camada, é realizado o controle de versões. O *framework* permite que coexistam várias versões do mesmo *bundle* no mesmo contêiner OSGi. Essa funcionalidade é provida nessa camada. Quando um *bundle* importa um pacote, é possível, via metadados, informar qual a versão do pacote deve ser importada. Nesse caso, enquanto esse *bundle* estiver em execução, o *framework* vai continuar segurando a versão em questão. Contudo, isso não impede de outras versões do mesmo *bundle* sejam disponibilizadas no contêiner.

Fig. 3.6: *Class Loader Delegation model* [18].Fig. 3.7: *Class Space* [18].

### Module

A camada de módulo provê uma solução genérica e padronizada para a modularização na linguagem *Java* [18]. A unidade de modularização do *Framework* é denominada *Bundle*, que são arquivos *Jars* que tem como conteúdo recursos necessários para a execução da aplicação, como classes *Java*, arquivos de imagens etc. A especificação estende o uso do arquivos *jar*, incluindo propriedades (*headers*) que são utilizadas pelo *framework*. Os *headers* são incluídos no arquivo de *manifest*, seguindo o padrão de *nome = valor*. A seguir, são apresentados os principais *headers*:

- *Bundle-Activator*: nome da classe responsável por iniciar e parar o *bundle*;
- *Bundle-ClassPath*: define a lista de arquivos *jars* necessários para a publicação dos *bundles*;
- *Bundle-Description*: descrição do *bundle*;
- *Bundle-ActivationPolicy*: define como o *bundle* será ativado, por padrão, quando o *bundle* é iniciado, todas as classes são iniciadas, informando essa opção como *lazy*, as classes serão iniciadas sob demanda;
- *Bundle-Name*: define o nome do *bundle*;
- *Bundle-Version*: contém a versão do *bundle*, essa opção é utilizada pelo *framework* para gerenciar os mesmos *bundles*, de versões diferentes;
- *Export-Package*: define qual o pacote é exportado pelo *bundle*;

- *Export-Service*: estabelece quais os serviços contidos no *bundle* que são exportados;
- *Import-Package*: define os pacotes importados pelo *bundle*, nessa opção, já se inicia a relação de dependência;
- *Import-Service*: são os serviços exportados pelo *bundle*, também relacionados com dependência;
- *Require-Bundle*: *bundles* requeridos pelo *bundle* que define essa propriedade;
- *Bundle-RequiredExecutionEnvironment*: lista os ambientes requeridos para executar o *bundle*.

### Life Cycle

A camada de *Life Cycle* provê a API para o controle de segurança e ciclo de vida de operações dos *bundles* [18]. Esta tem dependência direta com a camada de *Module* e *Security* sendo esta última opcional. Outra função da camada de ciclo de vida é de iniciar o *framework*. A especificação define que o *framework* também é um *bundle* com algumas características a mais (especialização da interface *Bundle*), dessa forma, essa camada interage com o *framework* de forma quase transparente. Os estados do *framework*, durante a inicialização, são ilustrado pela Fig. 3.8 por meio de diagrama de estados.

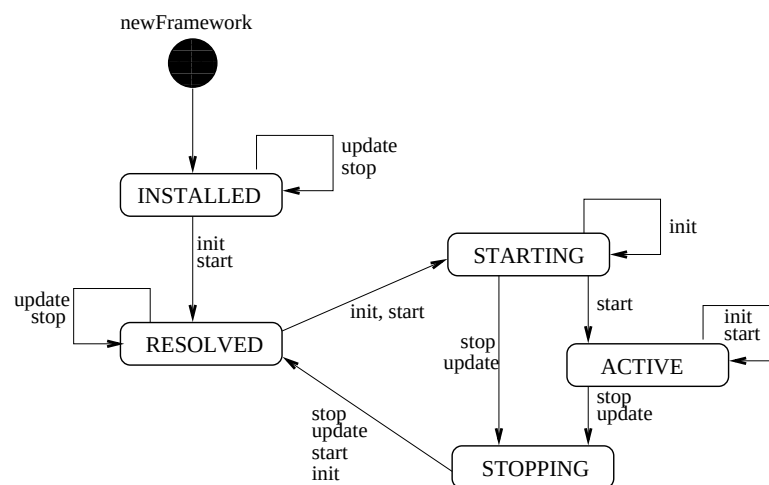


Fig. 3.8: Diagrama de Estados do *framework* [18].

A especificação define a interface de fábrica responsável por criar uma instância da implementação do *framework*. Cada contêiner deve criar uma implementação dessa interface para que o *framework* seja criado. Após criada a instância do *framework*, são seguidos os seguintes passos na inicialização:

- criar uma instância do *framework*, o estado inicial é informado como *INSTALLED*;
- executar o método *init*, o estado vai para *STARTING*;
- executar o método *start*, o estado passa para *ACTIVE*;
- após ativado, caso o método *stop* seja executado, o estado passa para *STOPPING*, após retornado do métodos *stop*, o estado passa para *RESOLVED*, podendo iniciar o ciclo novamente.

Os estados de um *bundle* seguem os mesmos estados ilustrado na Fig. 3.8, incluindo apenas mais um estado que é o *UNINSTALLED*, pois, diferente do *framework*, um *bundle* pode ser removido do contêiner.

### Service

A camada de serviço é responsável por prover o registro, localização, colaboração, notificação por eventos, filtros, permissões e também a comunicação entre *bundles*. Os serviços contidos nos *bundles* são implementações de interfaces que são registradas no contêiner, essas interfaces são fornecidas por qualquer instituição ou empresa. O registro de uma implementação de serviço é realizado pela interface do serviço, dessa forma, podem existir várias implementações do mesmo serviço no contêiner.

A publicação dos serviços na arquitetura proposta faz uso dessa funcionalidade para publicar os serviços de experimentos. Os serviços são identificados pela interface *ILabService*, e o *listener* adicionados no componente *Core* recebe notificações de serviços que implementam essa interface do contêiner.

### Bundles

O *bundle* é a unidade modular do contêiner, é formado por um conjunto de classes *Java* empacotadas em um arquivo *.jar*, que podem publicar e utilizar serviços de outros *bundles*.

Para a arquitetura proposta, os *bundles* são módulos que contém serviços dos experimentos.

## 3.4 Interação Entre os Componentes

A inicialização dos componentes da arquitetura será descrita por meio de diagrama de sequência que vai ilustrar desde a inicialização, até a execução de um serviço de um módulo de experimento, precedido de uma breve descrição do cenário ao qual o diagrama é aplicado.

### 3.4.1 Inicialização dos Componentes

Por ser um aplicação Web, a inicialização é realizada por meio da inicialização de contexto do contêiner Web. Toda vez que a aplicação é atualizada no contêiner, é gerado um evento de notificação de inicialização da aplicação. Esse evento é empregado para dar início a interação.

Os seguintes componentes fazem parte da interação:

- a) *Web Contêiner*;
- b) *Core*;
- c) *FrameworkFactory*;
- d) *FrameworkOSGi*;
- e) *ServiceListenerImpl*;
- f) *ServiceManager*.

O início da interação se dá por um agente externo, no caso, Web Contêiner, que executa a inicialização no componente *Core*, que, por sua vez, inicializa todos os outros componentes, na seguinte ordem: 1) *FrameworkOSGi*, 2) *ServiceListenerImpl* e 3) *ServiceManager*.

A troca de mensagens referente à inicialização é apresentada na Fig. 3.9.

A seguir, são expostas as descrições de cada interação entre os componentes:

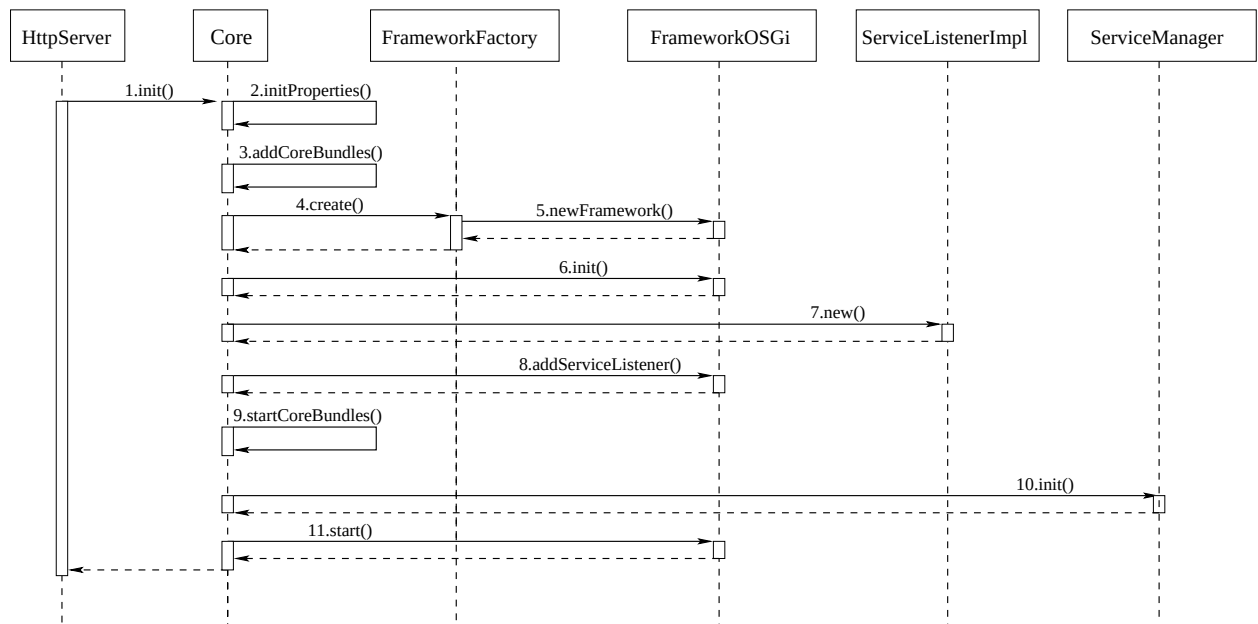


Fig. 3.9: Inicialização dos Componentes da Arquitetura Proposta.

1. *init()* - Web Contêiner executa esse método da classe principal da arquitetura que é o *Core*. Esse componente é responsável por controlar todo o ciclo de execução dos serviços dos módulos de experimento. Por esse motivo, é o primeiro a ser iniciado;
2. *initProperties()* - este método é executado no próprio *Core*. É responsável por iniciar algumas propriedades relacionadas ao *framework* OSGi;
3. *addCoreBundles()* - adiciona os *bundles* requeridos pelos componentes da arquitetura. Esses *bundles* fazem parte do *Core* da arquitetura, por isso, devem ser iniciados junto com o *framework*;
4. *create()* - cria a fábrica do *framework*. Essa fábrica é uma implementação da interface *org.osgi.framework.launch.FrameworkFactory*. Toda implementação da especificação tem uma implementação dessa interface;
5. *newFramework()* - cria uma instância do *framework* OSGi
6. *init()* - inicia o *framework* recém criado;
7. *new()* - cria uma instância do componente *ServiceListenerImpl*, que é responsável por receber as notificações de serviços de experimentos *framework* OSGi;
8. *addServiceListener()* - adiciona o componente criado no passo anterior ao contêiner OSGi;
9. *startCoreBundles()* - este método, já com o contêiner iniciado, é responsável por iniciar os *core bundles*. Esse método instala cada *bundle* que foi adicionado pelo método *addCoreBundles* no contêiner e também executa o *start*, dessa forma, todos os *bundles* ficam disponíveis para utilização;
10. *init()* - cria o componente *ServiceManager*, que é o componente que faz o interfaceamento entre o *Dispatcher* e o *Core*;

11. *start()* - após executado o *init* do *ServiceManager*, o próximo passo é executar o *start* do *framework*. Após executado o *start*, todos os *bundles* são ativados, inclusive, os *bundles* de experimentos.

O *start* do *framework* OSGi é o último a ser executado, porque, ao iniciá-lo, todos os módulos disponíveis na aplicação serão carregados. Por esse motivo, tanto o *Core* quanto o *ServiceManager* devem estar iniciados e prontos para receber as notificações de serviços.

### 3.4.2 Disponibilização e Remoção de Módulos

Para disponibilizar um novo módulo no contêiner OSGi, é necessário seguir alguns passos. O módulo deve ser instalado no contêiner por meio de uma URL, seja remota ou local (arquivo na mesma máquina). Após instalado, o módulo fica no estado de *INSTALLED*, contudo, não iniciado, ou seja, serviço algum desse módulo foi exportado para o contêiner. O módulo só vai ter seus serviços disponibilizados após ser iniciado por meio do *start*. Caso todas as dependências do módulo sejam resolvidas, o estado passa para *ACTIVE*, em caso de alguma dependência falhar, o estado muda para *RESOLVED*. Os estados de um módulo e suas transições são ilustrados na Fig. 3.8.

Esses passos não são executados automaticamente pelo *framework*, por isso, é necessário algum mecanismo externo para poder ativar o módulo automaticamente.

Como descrito anteriormente, no componente *Core*, são adicionados alguns *bundles* denominados de *coreBundles*, em que, um deles é responsável por executar o processo de disponibilização, atualização ou remoção de um *bundle*. Esse *bundle* fica monitorando um determinado diretório por arquivos *Jar*'s, quando identifica algum arquivo, ele faz todo o processo descrito anteriormente. Mais detalhes sobre o funcionamento desse *bundle* serão descritos no próximo capítulo.

Os seguintes componentes fazem parte da interação:

- a) Módulo;
- b) *FrameworkOSGi*;
- c) *ServiceListenerImpl*;
- d) *Core*;
- e) *ServiceManager*;

A Fig. 3.10 ilustra a disponibilização (*deployment*) de um módulo de experimento, considerando que todo o processo de instalação e inicialização seja executado por outro agente externo. Esse processo foi abstraído pela mensagem *deploy*.

A mensagem *load* entre o componente *FrameworkOSGi* e o Módulo, também teve todo o processo de *ClassLoader* do *framework* abstraído, para simplificar o diagrama.

Ao ser instalado um módulo, o *FrameworkOSGi* carrega todos os metadados contidos no arquivo de *manifest*. No momento da inicialização, é criada uma instância da classe declarada na propriedade *Bundle-Activator*. Esta classe é responsável por registrar todos os serviços contidos no módulo. O processo de registro do serviço faz com que o *framework* lance eventos relacionados ao serviço que está sendo registrado.

A seguir, são apresentadas as descrições de cada interação entre os componentes:

1. *deploy* - processo de disponibilização do módulo de experimento. Por depender de um agente externo, foi abstraído para simplificar o cenário. No capítulo seguinte, são descritos mais detalhes sobre esse processo;

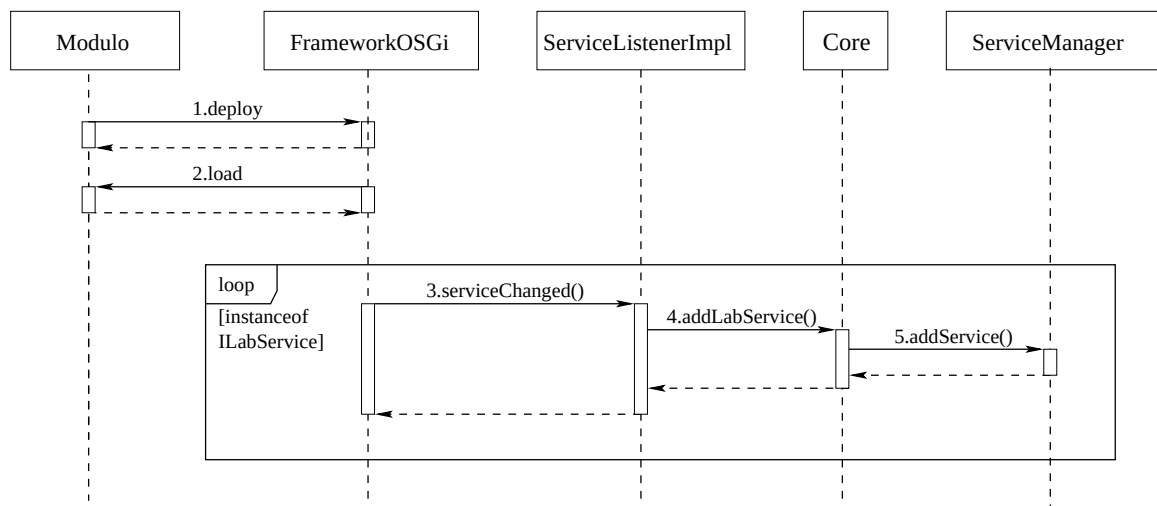


Fig. 3.10: Sequência de Disponibilização de Módulo

2. *load* - processo de carga de serviços do módulo em que o *framework* processa todos os serviços que são uma implementação da interface *ILabService*;
3. *serviceChanged()* - no processo de inicialização dos componentes, o *Core* registra a classe *ServiceListenerImpl* como ouvintes dos eventos de modificações de serviços. Dessa forma, ao ser carregado um serviço que implementa a interface *ILabService*, o *framework* executa esse método;
4. *addLabService()* - ao receber um evento de notificação de serviço, o *ServiceListenerImpl* verifica se o evento é de registro, caso seja, esse método é executado no componente *Core*;
5. *addService()* - uma vez adicionado o serviço no *Core*, esta executa esse método no componente *ServiceManager* para, então, disponibilizar o serviço para execução.

Todo esse processo descrito, é realizado para cada serviço registrado, por isso, é efetuado dentro de um laço de repetição, para todo serviço, que é uma implementação de *ILabService*.

O processo de remover um módulo segue o mesmo princípio da disponibilização, com a diferença que o módulo é finalizado (executado o *stoped* do módulo). O contêiner, então, executa o método de *stop* da classe de ativação, que, por sua vez, remove o registro do serviço do contêiner. Por isso, o laço de repetição é o mesmo da disponibilização, o que muda é um dos parâmetros informado no método *serviceChanged*, que tem como valor *ServiceEvent.UNREGISTERING*.

A Fig. 3.11 ilustra como é realizada a troca de mensagens quando um módulo de experimento é removido.

1. *remove* - este processo remove o módulo de experimento do contêiner. Esse processo, assim como o *deploy* da disponibilização, também depende de um agente externo, e foi abstraído para simplificar o cenário;
2. *load* - processo que seleciona todos os serviços que foram registrados pelos módulos em questão;
3. *serviceChanged()* - este método é executado para todo o serviço *ILabService*;
4. *removeLabService()* - ao receber um evento de notificação de serviço, o *ServiceListenerImpl* verifica se o evento é de remoção, caso seja, esse método é executado no componente *Core*;

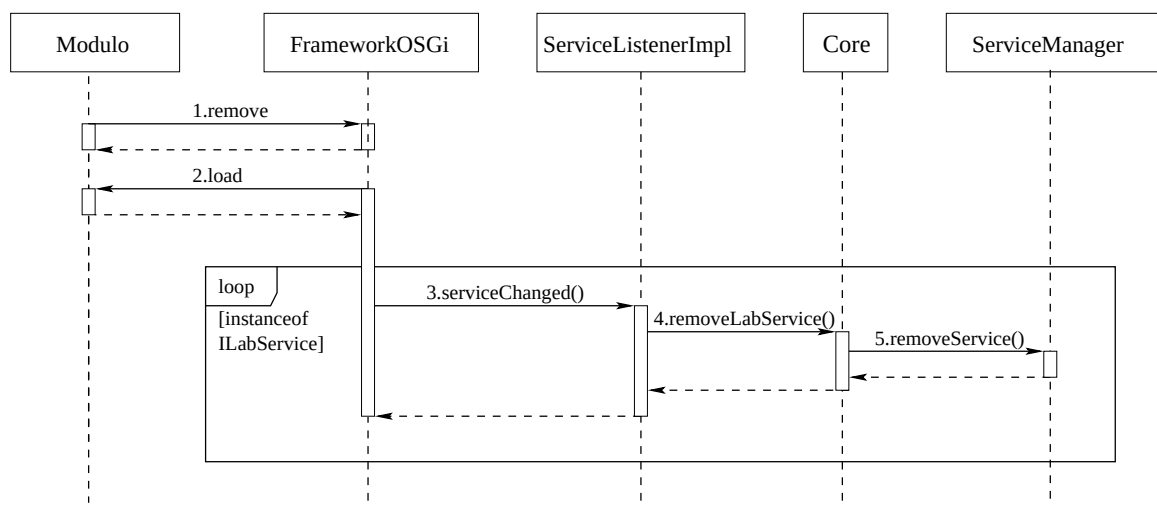


Fig. 3.11: Sequência de Remoção de Módulo

5. `removeService()` - antes de remover o serviço no *Core*, este executa esse método no componente *ServiceManager*, para, que remova o serviço. Uma vez removido desse componente, o serviço não pode mais ser executado.

Todas as chamadas de métodos no processo de ciclo de vida dos módulos são síncronas, conforme definido na especificação OSGi.

### 3.4.3 Execução de Serviço

A execução do serviço se inicia por uma requisição da aplicação cliente para o servidor do NetLab. Nesse momento, a aplicação cliente será tratada como um objeto que faz parte da troca de mensagens. Detalhes de como a requisição chega até o *Dispatcher* são específicos de cada implementação deste. Mais detalhes sobre como a aplicação executa as requisições serão descritos no capítulo seguinte.

Os seguintes componentes fazem parte da interação:

- a) Cliente;
- b) *Dispatcher*;
- c) *ServiceManager*;
- d) *Core*;
- e) *FrameworkOSGi*;
- f) *Módulo*;
- g) *Host*.

A interação se inicia com uma execução ao método `execute` do componente *Dispatcher*. A Fig. 3.12 ilustra a sequência de execução de um serviço solicitado pelo componente externo à arquitetura, que é a aplicação cliente. Logo na sequência, é apresentada a descrição de cada interação.

1. `execute()` - um agente externo executa esse método do componente *Dispatcher*;
2. `buildContextInvoker()` - para executar um serviço, é necessário haver um contexto, esse método cria o contexto, de acordo com os parâmetros passados na requisição. O que também o torna dependente da implementação do *Dispatcher*;

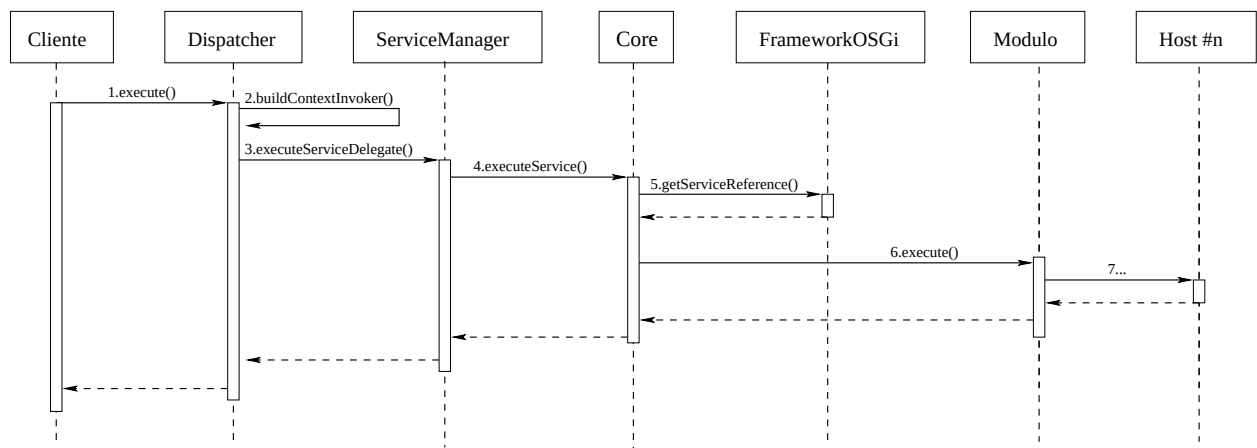


Fig. 3.12: Sequencia de Execução de Serviços

3. *executeServiceDelegate()* - este método solicita ao *ServiceManager* a execução do serviço solicitado;
4. *executeService()* - o *ServiceManager* delega a execução do serviço para o componente *Core* ao executar esse método;
5. *getServiceReference()* - este método retorna a referencia do serviço no *FrameworkOSGi*;
6. *execute()* - novamente no *Core*, com a referência do serviço, é solicitada a execução do deste;
7. a interação do módulo de experimento com o *Host* é específica da implementação, e pode ser realizada por *RMI* é, capaz de executar qualquer método, por isso, não foi descrito qual método é executado.

### 3.5 Considerações Finais

A arquitetura apresentada foi projetada para atender aos requisitos de propósitos gerais e específicos para o domínio de redes para WebLabs. A principal função da arquitetura proposta é publicar os serviços contidos nos módulos de experimentos e prover a interação entre esses serviços disponibilizados em um contêiner OSGi e aplicações clientes. A publicação de um experimento não requer alteração em nenhum componente arquitetural, dessa forma, a criação de um novo experimento se torna simples.

Para a arquitetura proposta, um experimento é somente um módulo (*bundle*) OSGi. Não é necessário mais nenhum artefato externo para que o experimento seja disponibilizado. Todas as suas dependências externas estão contidas no módulo. A aplicação cliente com todas as suas dependências externas também fazem parte desse módulo.

O uso de um *framework* OSGi aumenta a flexibilidade do WebLab, ao permitir o controle de versão de módulos de experimentos, controle de dependência, facilidade na instalação, atualização remoção.

A arquitetura não expõe diretamente o *framework* OSGi para todos os componentes, apenas um dos componentes é responsável por gerenciar o mesmo. Dessa forma, é possível abstrair as funcionalidades do contêiner OSGi sem que componentes das camadas mais altas tenham dependência com este.

O próximo capítulo descreve a implementação da arquitetura proposta.

# Capítulo 4

## Implementação e Resultados

Este capítulo apresenta os aspectos técnicos utilizados para a implementação da arquitetura proposta, bem como a criação de um novo experimento que faz uso da nova arquitetura. A criação de um novo experimento que faça uso da nova arquitetura tem por finalidade validar a arquitetura proposta.

O experimento selecionado para prova de conceito foi de configuração de VLAN (*Virtual Local Area Network*), os requisitos para criação desse experimento não só validam a arquitetura, mas também estendem o uso do WebLab por permitir a criação de várias redes virtuais no laboratório.

A criação de VLANs interage diretamente com o *hardware*, neste caso, um *Switch*. A maioria dos fabricantes desse tipo de equipamento disponibilizam formas diferentes de configurações, como: SNMP (*Simple Network Management Protocol*), SSH (*Secure Shell*), Telnet ou por uma interface Web. Essa variedade em como configurar o equipamento torna esse experimento ideal para criar uma separação da configuração do equipamento em módulos diferentes, que podem ser utilizados pelo módulo do experimento propriamente dito. Com isso, é possível ver os resultados da modularização provida pela arquitetura.

Por ter como base o Projeto NetLab WebLab [2, 4], a arquitetura proposta foi implementada de forma a permitir que experimentos já criados possam ser utilizados.

Embora a implementação da nova arquitetura não dê suporte ao protocolo SOAP, ela é extensível o suficiente para possibilitar implementações que necessitem desse protocolo.

### 4.1 Aspectos de Implementação da Arquitetura Proposta

A arquitetura proposta foi projetada para ser adaptável para WebLabs de qualquer domínio, por esse motivo, no capítulo 3, não se entra em detalhes de como determinados componentes deveriam ser implementados. Esses detalhes devem ser especificados na implementação, levando em consideração os aspectos de cada domínio de laboratório. Por isso, os detalhes da implementação de cada componente serão descritos a seguir, bem como os detalhes de infraestrutura.

A plataforma escolhida para dar suporte a nova arquitetura foi a Plataforma Java™JDK (*Java Development Kit*), versão 1.6 ou superior. A Plataforma Java, em si, não provê mecanismos para modularização dinâmica, porém provê componentes que permitem criar essa modularização. É a partir desses mecanismos que o *framework* OSGi propicia essa modularização.

Antes de entrar em mais detalhes sobre a implementação de cada componente, será apresentada a forma como esta foi estruturada.

As classes referentes a implementação da arquitetura foram separadas em dois pacotes, conforme ilustra a Fig. 4.1.

O pacote *core* é composto pelas classes de implementação dos componentes que fazem parte do núcleo principal arquitetura. Já no pacote *services*, ficam as classes relacionados aos serviços. Essa

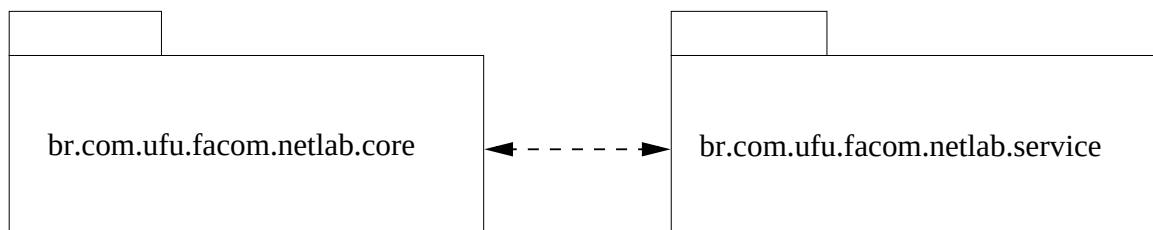


Fig. 4.1: Divisão dos Pacotes da Implementação.

divisão de pacotes entre componentes da arquitetura e componentes de serviço possibilita que sejam gerados dois artefatos do tipo *jar*, um como biblioteca, que provê a implementação referencial da arquitetura e outro como um *bundle* OSGi.

A forma de como utilizar um ou outro artefato fica por conta do desenvolvedor. No caso dessa implementação, optou-se por recorrer ao *jar* como biblioteca. O *bundle* também pode ser usado como biblioteca, pois contém as interfaces necessárias para implementação de um novo experimento.

Para criar um novo experimento, o desenvolvedor não precisa da implementação da arquitetura, basta a biblioteca (arquivo *jar* do *bundle*) que contenha as interfaces que serão utilizadas. Isso facilita tanto a publicação de novas versões quando o desenvolvimento.

Na implementação de referência, optou-se por não separar em projetos diferentes as interfaces e suas respectivas implementações. Por isso, tanto as interfaces dos componentes, quanto suas implementações foram acomodadas no mesmo projeto.

A fim de ilustrar melhor todas as classes da implementação, a Fig. 4.2 apresenta o diagrama de classes com as interfaces e suas implementações, não considerando a divisão de pacotes. Logo em seguida, a Fig. 4.3 ilustra as classes separadas por seus respectivos pacotes.

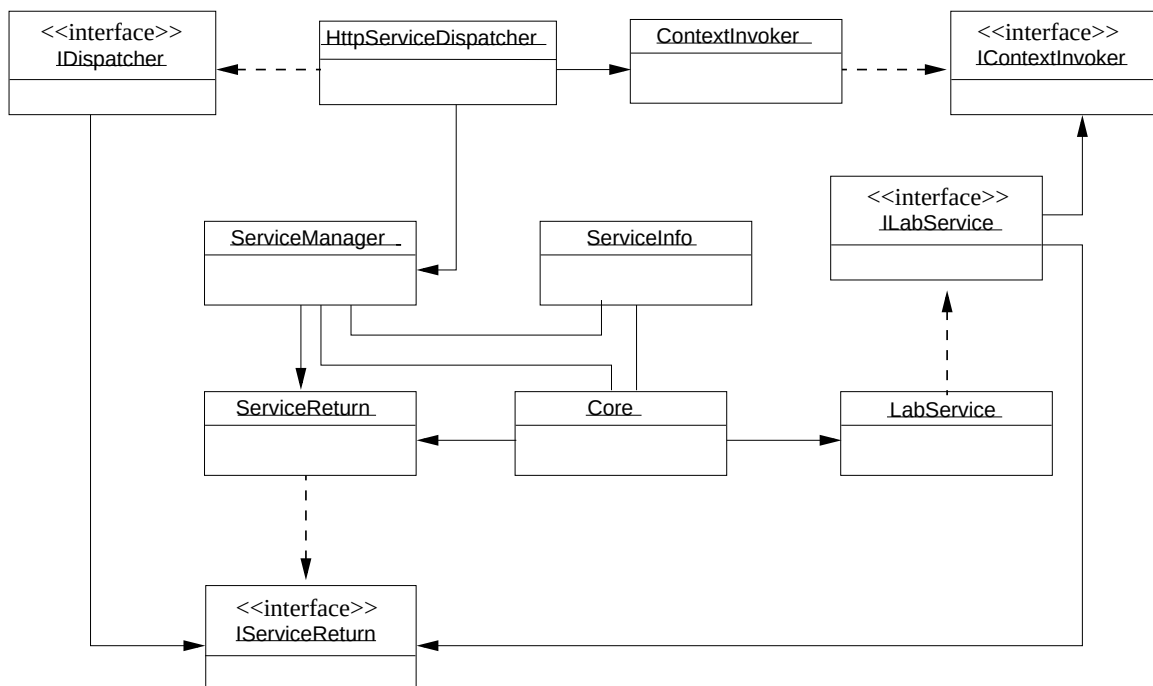


Fig. 4.2: Diagrama das Interfaces e Classes de Implementações dos Componentes.

A seguir, são descritas as características relacionadas a implementação de cada componente da arquitetura.

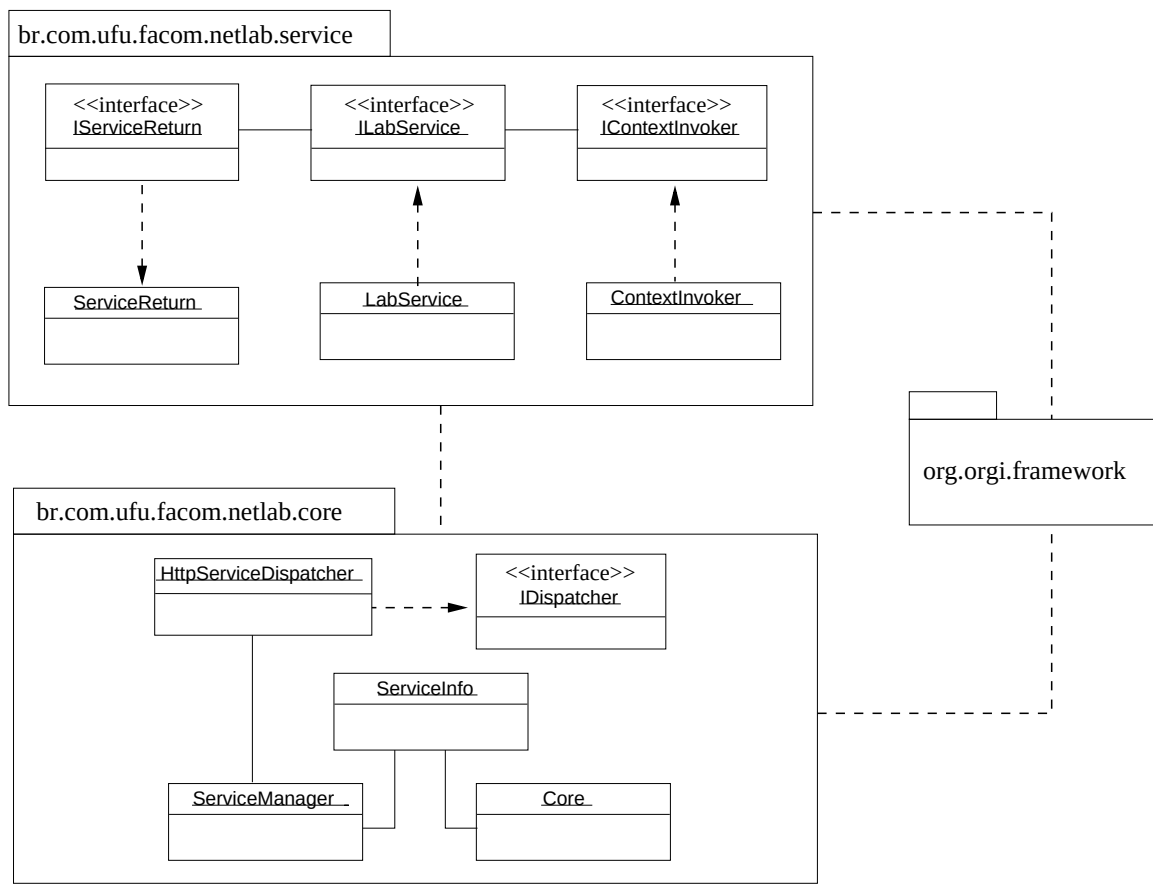


Fig. 4.3: Diagrama de Classes Separados por Pacotes.

#### 4.1.1 Componente *Dispatcher*

O componente responsável em receber as solicitações de execução de serviço é o *Dispatcher*. O fato de o NetLab WebLab ser uma aplicação Web requer que esse componente atenda a requisições HTTP. Por isso, o protocolo escolhido para atender as requisições da aplicação cliente foi REST [20]. A escolha desse protocolo levou em conta sua facilidade de uso, baixo acoplamento e melhor escalabilidade. A implementação do componente é feita pela classe *HttpServiceDispatcher*. Essa classe é um *Servlet* Java que implementa a interface *IDispatcher*. Por seguir o estilo arquitetural REST, todos os métodos do HTTP são suportados por essa classe: *GET*, *POST*, *PUT* e *DELETE*.

Todo serviço publicado dinamicamente pela arquitetura é mapeado para uma URI, que o identifica. Essa URI é composta de um contexto, nome do serviço e parâmetros no estilo REST. Cabe ao *HttpServiceDispatcher* fazer o *parser* da requisição para extrair a URI e chegar até o serviço. Como exemplo, uma URL de solicitação de execução de serviço tem o seguinte formato:

*http://netlabserver/contextoAplicacao/rest/ctxServico/nomeServico/?v=2&p1=valor1&p2=valor2*  
Onde cada parte da URL é discriminada:

- *netlabserver* - nome ou IP (*Internet Protocol*) do servidor do NetLab;
- *contextoAplicacao* - contexto da aplicação Web. Toda aplicação Web é separada por um contexto, no caso do NetLab, o contexto é NetLabWL;
- *rest* - mapeamento para o *Servlet* *HttpServiceDispatcher*. Toda requisição que tenha como URL NetLabWL/rest é atendida por esse *Servlet*;

- *ctxServico* - contexto do serviço. Todo serviço deve ser publicado em um contexto específico, o que permite que novas funcionalidades possam ser adicionadas a um experimento existente;
- *nomeServico* - nome do serviço a ser executado;
- *?v=2&p1=valor1&p2=valor2* - tudo que vem após o ? são considerados parâmetros. Que seguem o padrão nome=valor.

Uma requisição pode conter, além dos parâmetros na URL, um XML, no caso de um método *POST* ser executado. Nesse caso, é preciso ler os dados da requisição e criar o XML. Embora a implementação já espere receber o XML, este não é reconhecido pelo *HttpServiceDispatcher*, ou seja, não é necessário reconhecer o padrão do arquivo. A única coisa pertinente é criar o XML a partir dos dados enviados via *POST* pela aplicação cliente. A criação do XML enviado é feita pelo método *buildXmlFromRequest()* da interface *IDispatcher*.

Antes de solicitar a execução do serviço para o *ServiceManager*, é essencial criar o contexto de execução. A criação do contexto é realizada pelo método da interface: *buildContextInvoker()*. Esse contexto contém os parâmetros passados pela URL e também o XML (quando enviado). Assim como o XML, os parâmetros também não são reconhecidos pelo *HttpServiceDispatcher*. A Fig. 4.4 ilustra como se dá a comunicação entre a aplicação cliente e o servidor até a execução do serviço no módulo.

Após criado o contexto de execução do serviço (*ContextInvoker*), a solicitação de execução é efetuada no *ServiceManager*. O resultado dessa execução é retornado e, dependendo do tipo de contexto do serviço, o método *writeResultToOutput()* é executado para retornar o XML, retornado do serviço para a aplicação cliente.

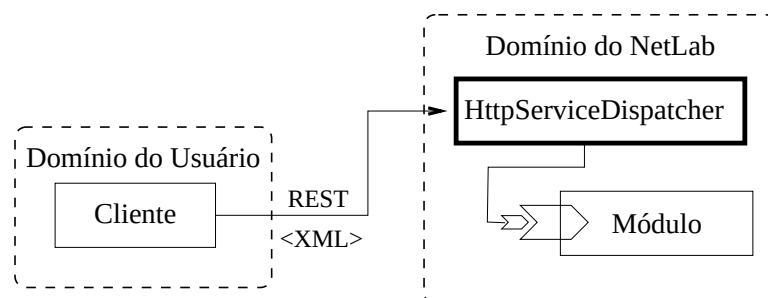


Fig. 4.4: Atendimento de Requisições REST pelo *HttpServiceDispatcher*.

O *HttpServiceDispatcher* permite o atendimento de várias requisições, simultaneamente, para uma única instância criada, utilizando linhas de execuções diferentes (*multi-threading*). Por ser um centralizador da execução dos serviços, sua implementação leva em consideração o desempenho para atender as requisições o mais rapidamente possível.

#### 4.1.2 Componente *ServiceManager*

A função desse componente é abstrair as rotinas do componente *Core* para o componente *Dispatcher*. Sua implementação não trata de nenhum detalhe sobre a requisição, o que o torna genérico para qualquer tipo de *Dispatcher* implementado. Sua interface possui métodos que são utilizados no *HttpServiceDispatcher* e no *Core*. A Fig. 4.5 ilustra essa integração.

Os métodos públicos, relacionados a execução dos serviços que são empregados pelo *HttpServiceDispatcher*, são listados a seguir:

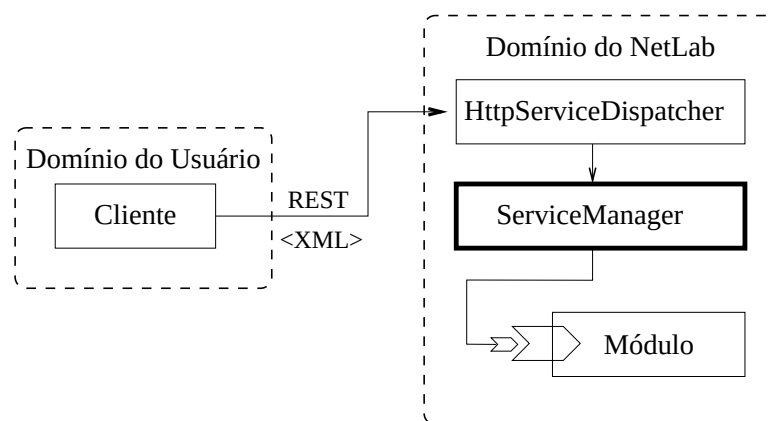


Fig. 4.5: Interação entre o *HttpServiceDispatcher* e *ServiceManager*.

- *validateAndBuildServiceInfo()* - esse método é utilizado para validar se existe um serviço associado a URI passada como parâmetro. A classe *ServiceInfo* contém informações acerca do serviço a ser executado, por isso, quando existe um serviço mapeado para a URI informada, esse método retorna uma instância de *ServiceInfo* com suas informações. Essa instância é utilizada para criar um contexto de execução: *ContextInvoker*;
- *buildElementError()* - como o *HttpServletDispatcher* atende as requisições REST, quando ocorrer erros, esse método cria um XML padronizado com informações sobre o erro;
- *executeServiceDelegate()* - após a execução bem sucedida do método de validação, o próximo passo é executar o serviço propriamente dito. Esse método não executa o serviço diretamente e, sim, delega para o *Core* executá-lo.

A validação da sessão de interação é realizada nesse componente, por isso, antes de executar o serviço, o *ServiceManager* valida se existe uma sessão criada para o usuário que solicitou a execução. Essa sessão de interação é criada ao executar o experimento pelo portal do laboratório que integra a parte administrativa do NetLab.

Para o componente *Core*, os métodos utilizados são:

- *addService()* - esse método adiciona em uma lista interna e, por contexto, os serviços disponibilizados. O *ServiceManager* não guarda referências dos serviços. A referência destes só é acessada pelo *Core*.
- *removeService()* - remove da lista interna um serviço específico.

Os métodos listados não podem ser executados por componentes fora da implementação da arquitetura, por isso, a implementação do componente faz uso de um mecanismo da Programação Orientada a Objetos que permite mudar a visibilidade dos métodos. Nesse caso, a sua visibilidade é de pacote (*default*) na linguagem Java. Isso faz com que somente classes que pertençam ao mesmo pacote possam acessar os métodos.

O *ServiceManager* não é definido por uma interface. A sua especificação exige detalhes relacionados a funcionalidades da arquitetura, como: não ter mais de uma instância, não permitir que classes fora da arquitetura executem métodos críticos que possam propiciar a quebra de integridade dos serviços. Por isso, é definido como uma classe concreta.

Para permitir que só exista uma instância desse componente, este é implementado como uma classe *Singleton*. *Singleton* é um Padrão de Projeto de *software*, que define um modelo que possibilita criar uma única instância de uma determinada classe [21]. Essa característica permite que a implementação do *HttpServiceDispatcher* e *Core* compartilhe a mesma instância do *ServiceManager*.

### 4.1.3 Componente Core

O componente *Core* é responsável por gerenciar o *framework* OSGi e restringir acesso a ele, de forma que nenhum outro componente da arquitetura possa acessá-lo diretamente. O motivo de não permitir o acesso a ele é de abstrair as funcionalidades e usar somente as que são necessárias para a publicação dos serviços de forma dinâmica.

É o principal componente da arquitetura. Entre suas funções, está a de iniciar o *HttpServiceDispatcher*, *ServiceManger* e o *framework* OSGi. A interação entre o *Core* e o *ServiceManager* é realizada por meio dos métodos *addService* e *removeService*, que são executados, quando recebe os eventos do contêiner OSGi. A Fig. 4.6 ilustra a interação ente esses componentes.

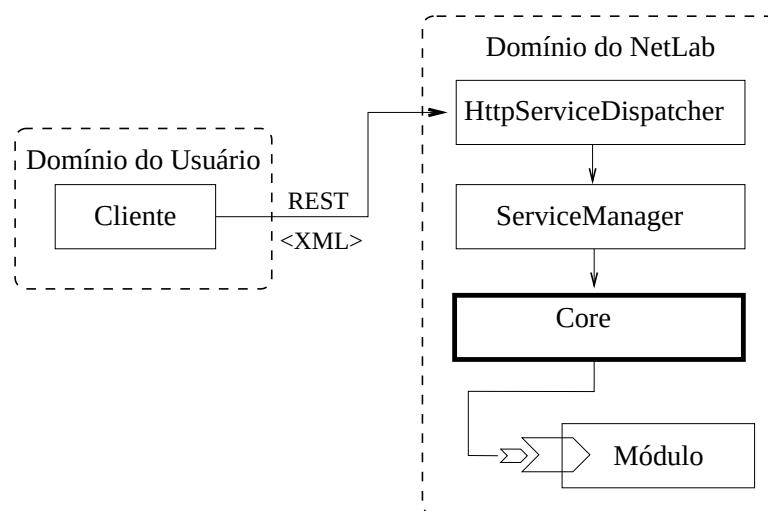


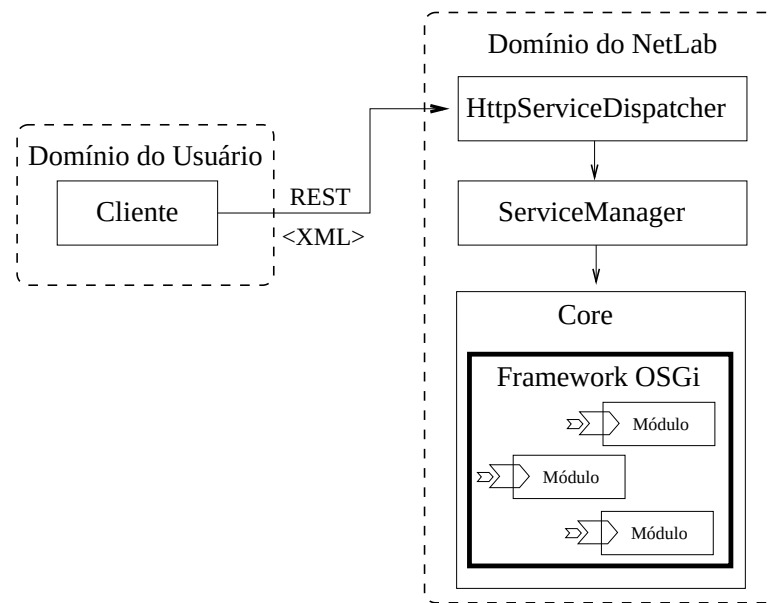
Fig. 4.6: Interação entre o *Core* e *ServiceManager*.

O *Core* envolve o *framework* OSGi (Fig. 4.7), criando uma dependência direta, contudo segue a especificação *OSGi Service Platform* [18], o que não cria dependência com um *framework* em específico, visto que todo *framework* deve prover a implementação das interfaces públicas. O *Core* só faz uso das interfaces padrão da especificação, por isso, qualquer implementação desta pode ser utilizada.

Dentre as várias implementações da especificação, os seguintes projetos foram selecionados para avaliação:

- *Felix* [22] - implementação de código aberto desenvolvida pela *Apache Software Foundation*;
- *Equinox* [23] - implementação também de código aberto disponibilizada pela *Eclipse Foundation*;
- *Knopflerfish* [24] - desenvolvido pela empresa privada *Makewave*. Pode ser utilizado na versão comercial ou código aberto.

Durante os testes realizados em alguns projetos, possibilitou a identificação de alguns requisitos que foram empregados para a seleção de qual *framework* integrar com a implementação. Os requisitos levantados foram:

Fig. 4.7: Interação entre o *Core* e *Framework OSGi*.

- a) código aberto (*open source* tem como licença a *Apache license*);
- b) tamanho incluindo dependências externas;
- c) extensa documentação;
- d) facilidade de execução em contêiner Web.

O *framework* que atendeu a todos os requisitos foi o *Felix*. É um projeto com uma documentação razoável, não tem nenhuma dependência com bibliotecas de terceiros, sendo disponibilizado em um único arquivo de, aproximadamente, 380KB. É um projeto de código aberto em constante desenvolvimento. Entre os projetos avaliados, foi o mais simples de ser executado de dentro de um contêiner Web.

Embora a restrição imposta pelo *Core* às funcionalidades do contêiner OSGi, quanto a publicação dos serviços, essas restrições não são aplicadas aos módulos de experimentos. Um módulo pode importar e exportar qualquer tipo de serviço independente da interface *ILabService*. A diferença é que qualquer serviço exportado, que não seja uma implementação desta interface, só pode ser utilizado por módulos dentro do contêiner. Por outro lado, esta torna os módulos de experimentos mais flexíveis, por possibilitar que um módulo faça uso de todas as funcionalidades de um contêiner OSGi.

### Bundles Requeridos pela Implementação da Arquitetura

Na própria implementação da arquitetura, já é possível analisar os ganhos ao recorrer a um contêiner OSGi para a gerência dos módulos de experimentos. Um *bundle* não pode ser instalado automaticamente por um contêiner, é necessária uma funcionalidade externa para fazer a instalação e iniciar o *bundle*. Nesse caso, essa funcionalidade deveria fazer parte da implementação da arquitetura, contudo existem muitos *bundles* que provêm várias funcionalidades disponibilizadas em repositórios públicos. Dessa forma, ao utilizar um *bundle* já pronto, não há necessidade de implementar, o que já demonstra os benefícios do uso de um *framework* OSGi.

O projeto utilizado para executar a tarefa de instalar e iniciar o *bundle* foi *Apache Felix File Install* [25]. O *File Install* é um agente de gerenciamento de *bundle* OSGi baseado em diretório. Ao ser

instalado o *File Install* no contêiner, é possível configurar um diretório que, então, fica sendo monitorado. Quando é copiado um *bundle* para esse diretório, o *File Install* instala-o no contêiner, e caso todas suas dependências sejam resolvidas, ele, então, executa o *start* do *bundle*. Um procedimento semelhante é realizado quando o *bundle* é atualizado no diretório. Quando o arquivo é removido, o *File Install* também remove o *bundle* do contêiner.

Na implementação da arquitetura, as configurações para o *File Install* são informadas por propriedades passadas na inicialização do *framework* pelo *Core*. No caso, a propriedade que informa qual o diretório deve ser monitorado é: *felix.fileinstall.dir*, com o valor: `%HOME%/.netlab/load`.

O *bundle File Install* depende de outro *bundle*, que é o *Config Admin* [26]. Esse *bundle* permite alterar as configurações de qualquer módulo que faça uso de suas funcionalidades em tempo real.

A definição da arquitetura proposta levou em consideração esse tipo de dependência, por isso, definiu mecanismos para possibilitar instalar e iniciar os *bundles* referentes a tais dependências. Esses *bundles* foram denominados *Core Bundles*. Na implementação, é admitido adicionar quais *bundles* fazem parte do *Core*, e uma vez adicionados, eles são instalados e iniciados, automaticamente, por mecanismos internos da arquitetura.

Outro *bundle* utilizado pelo *Core* é o de *logging* [27], que fornece serviços padronizados de *logging* para a especificação OSGi. Uma vez instalado e iniciado pelo *Core*, qualquer módulo de experimento pode fazer uso dos serviços de *logging*. Os registros de *log* são informações importantes para administradores e desenvolvedores para rastrear possíveis problemas nos experimentos. O *Pax-logging*, assim como o *File Install* também fazem uso do *Config Admin*, o que permite que suas configurações sejam alteradas em tempo de execução sem que seja preciso reiniciar o módulo ou o contêiner. O *logging* é o típico projeto OSGi, pois é necessário um módulo com a definição da interface, e outro com a implementação. O que faz essencial dois *bundles* para prover os serviços exportados pelo módulo de log.

A implementação da arquitetura faz uso dos dois projetos apresentados, sendo estes considerados partes do núcleo da arquitetura, por isso, são instalados e iniciados pelo *Core* após iniciado o contêiner. Isso possibilita que qualquer módulos faça uso dos serviços providos por esses *bundles* sem que haja intervenção do desenvolvedor instalá-los como parte da dependência dos módulos. Outras dependências que não fazem parte do núcleo da arquitetura podem ser instalados e iniciados pelo *File Install* ou de forma manual pelo usuário.

O *framework* OSGi possui uma característica que permite adicionar *bundles* com dependências não resolvidas. Estes ficam no estado *RESOLVED*, e assim que qualquer *bundle* com a dada implementação da dependência for disponibilizado no contêiner, automaticamente o *bundle* passa de *RESOLVED* para *ACTIVE*. Por esse motivo, a ordem de disponibilização de módulos não é um requisito.

## 4.2 Cenário para Validação da Arquitetura Proposta

Esse cenário descreve o que é necessário para incluir um novo experimento no NetLab, o de configuração de VLANs, bem como as alterações necessárias para esse processo. Também é descrito como proceder a reserva para o experimento criado, e a sua execução após reservado.

### 4.2.1 Infraestrutura do Laboratório

O NetLab conta com uma infraestrutura de seis computadores, sendo um servidor e os outros, *hosts* integrantes do laboratório. A Tab.4.1 apresenta as características de cada máquina.

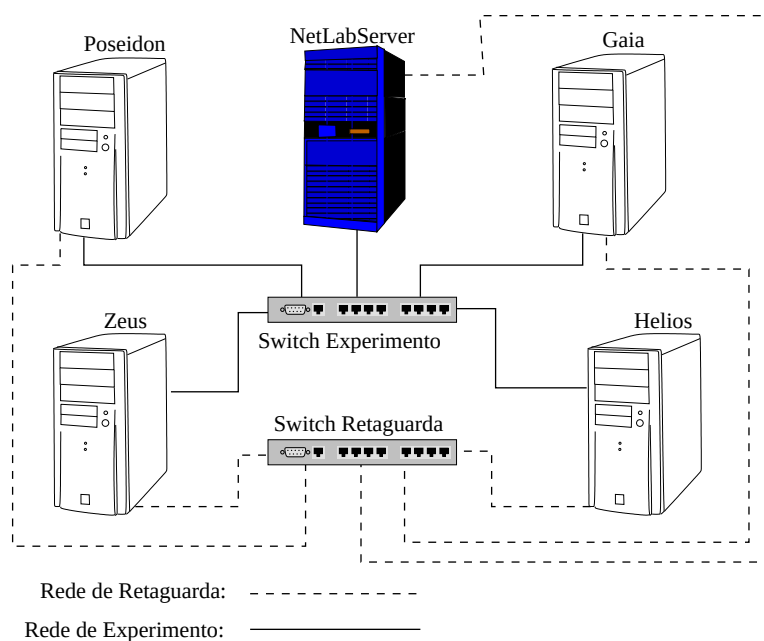
| Host         | CPU                      | RAM   | HD    | NICs |
|--------------|--------------------------|-------|-------|------|
| NetLabServer | Intel Xeon, 2x1.6Ghz     | 8GB   | 1TB   | 2    |
| Hodes        | Intel Core 2 Duo, 2.4Ghz | 3GB   | 250GB | 3    |
| Helios       | Intel Pentium 4, 3GHz    | 1GB   | 40GB  | 3    |
| Gaia         | Intel Pentium 4, 3GHz    | 1GB   | 40GB  | 3    |
| Urano        | Intel Pentium 3, 600MHz  | 256MB | 40GB  | 1    |
| Poseidon     | Intel Pentium 4, 3GHz    | 1MB   | 40GB  | 4    |
| Zeus         | Intel Pentium 4 HT, 3GHz | 512MB | 80GB  | 3    |

Tab. 4.1: Configurações dos Computadores do NetLab.

A infraestrutura conta, ainda, com dois *switches* (Tab. 4.2); um que acomoda a rede de retaguarda e outro que faz parte dos experimentos. A rede de retaguarda é uma rede que tem por finalidade garantir a comunicação entre o servidor do laboratório com os *hosts*. O NetLab possui experimentos que permitem alterar as configuração das NICs das máquinas. Esse é um dos motivos pelo qual a rede de retaguarda deve existir, pois ela garante a comunicação entre os *hosts* e o servidor, independente de qualquer configuração de rede alterada pelo experimento. A rede de retaguarda não é acessível pelos experimentos, somente pelo servidor do NetLab. A Fig. 4.8 ilustra a representação de quatro máquinas, os *switchs* e o servidor do laboratório. As outras máquinas seguem o mesmo padrão, tendo como requisito principal a conexão com a rede de retaguarda.

| Marca    | Modelo    | # Portas | Gerenciável ? |
|----------|-----------|----------|---------------|
| 3Com     | 4500      | 50       | Sim           |
| TRENDNet | TEG-160WS | 16       | Não           |

Tab. 4.2: Configurações dos Switchs.

Fig. 4.8: Infra-estrutura dos *Hosts* e *Switchs* do NetLab.

O NetLabServer é a máquina que acomoda o web contêiner com a parte administrativa, o portal

do NetLab, contêiner OSGi e também o banco de dados. A parte administrativa do NetLab tem como base o Projeto GigaBOT que possibilita a criação de laboratórios. A gerência administrativa é uma aplicação Web disponibilizada no web contêiner, com o nome *AccessService*. Este projeto é acessado pelo administrador do laboratório para a manutenção nos usuário, laboratório, experimentos ou recursos. A Fig. 4.9 apresenta a página administrativa e, em seguida, são descritas cada opção do menu:



Fig. 4.9: Portal Administrativo do NetLab parte do Projeto GigaBOT.

- *Users* - permite a manutenção dos usuários do laboratório usuários;
- *Groups* - permite a criação de grupos de usuários. Utilizados pelas rotinas de controle de acesso e autorização;
- *Roles* - permite a manutenção no cadastro de regras, empregado pelas rotinas de autorização;
- *Permissions* - manutenção no cadastro de permissões. A associação de regras e permissão a um grupo ou usuário, faz parte da autenticação;
- *Laboratories* - permite manutenção de vários laboratórios independentes de domínios. Um laboratório é composto por vários experimentos;
- *Experiments* - permite a manutenção do cadastro de experimentos. Um experimento é composto de um ou vários recursos;
- *Resources* - permite a manutenção no cadastro de recursos;

O cadastro de recursos do Projeto GigaBOT não foi projetado para suportar recursos físicos e lógicos exigidos pelos NetLab. Por esse motivo, foi preciso incluir marcações na descrição dos recursos e experimentos para fornecer informações necessárias para o NetLab. O cadastro de experimentos que fazem uso do *switch* segue essa mesma característica, por isso, as informações pertinentes para o experimento são disponibilizadas no seu cadastro.

Os *hosts*, além de conectados à rede de retaguarda, ainda possuem um objeto distribuído denominado FábricaRMI, que, dado um determinado experimento, é a responsável em criar os objetos

remotos para execução dos serviços no *host* alvo. Essa fábrica é utilizada ao iniciar o experimento. Para cada máquina que está cadastrada como um recurso físico (marcação *host* no cadastro de recurso) no experimento, é executada a criação dos recursos lógicos (marcação *serviço* no cadastro do recurso) no *host*. Após criados os objetos servidores pela fábrica, é possível executá-los remotamente a partir do servidor do laboratório via RMI.

No NetLabServer, são utilizados os seguintes *softwares* e suas respectivas versões:

- Sistema Operacional - Ubuntu-server 10.04 LTS (*Long Time Support*);
- *kernel* - 2.6.32 x86\_64 SMP;
- web contêiner - Tomcat versão 6.0.26;
- banco de dados - Mysql versão 5.1;
- framework OSGi - Felix versão 2.0.5.

Já os *hosts* que fazem parte do laboratório só apresentam dependência com o *kernel* do Sistema Operacional. Todos eles têm as mesmas especificações, conforme listado a seguir:

- sistema operacional - *Slackware* 10.2;
- *kernel* - 2.6.16 compilados com suporte a *DiffServ* [4].

#### 4.2.2 Cadastro do Experimento de Configuração de VLAN

Assim como uma máquina que faz parte do laboratório, o *switch* também deve ser cadastrado como um recurso, com suas características, como quantidade de portas, marca e modelo. A marca e modelo são informações importantes, pois, por essas informações, é que o módulo de configuração do *switch* será selecionado. No trabalho apresentado por Rocha [4] e Farias [2], os *hosts* do laboratório são cadastrados como recursos, e esses recursos, por sua vez, possuem sub-recursos, como, por exemplo, as NIC (*Network Interface Card*). No cadastro do experimento, é determinado o *enlace* entre os *hosts*. Um marcador foi utilizado para identificar a linha referente ao *enlace* na descrição do experimento, exemplo: [LINK] host1:eth2:host2:eth1.

O cadastro do experimento de configuração de VLAN segue esse mesmo modelo. Dessa forma, para os novos experimentos que fazem uso do *switch*, deve ser incluída no seu cadastro, a marcação para especificar as informações do equipamento. O *switch* usado no experimento é um 3Com modelo 4500 de 50 portas, a marcação no cadastro ficou: [SWITCH] name: 3Com, model: 4500, ports:46, conforme ilustra a Fig.4.10.

O campo *Screenplay URL* contém a URL, que é acessada quando o usuário selecioná-la para executar o experimento. Essa URL possui parâmetros que devem ser informados pelo desenvolvedor do experimento para o administrador do laboratório incluir o campo corretamente.

Para os sub-recursos (*NIC*) dos recursos (*hosts*) do experimento, também foi criada uma nova marcação, para informar em qual porta do *switch* a placa de rede está conectada. A marcação, no cadastro do recurso do tipo *host*, ficou da seguinte forma: [SW-PORT] eth1:13. O cadastro do recurso foi alterado para incluir a informação relacionada ao *enlace* do *host*, por isso, a marcação utilizada para os experimentos que utilizam a arquitetura anterior foi mantida, fazendo com que os experimentos antigos continuem funcionando. A Fig.4.11 apresenta marcação incluída para o experimento de VLAN e as marcações já existentes para o recurso.

A arquitetura proposta foi implementada levando em consideração os experimentos já existentes, por isso, houve todo um cuidado para que a disponibilização de experimentos na nova arquitetura

| Experiment Information |                                                                                                                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name                   | Vlan                                                                                                                                                                                                                    |
| Description:           | <div>Configuração de VLans</div> <div>[SWITCH] name: 3Com, model: 4500, ports:46</div>                                                                                                                                  |
| Description Url:       |                                                                                                                                                                                                                         |
| Screenplay URL:        | <a href="http://localhost:8080/NetLabWL/ServletExpV2?expName=Vlan&amp;ctx=vlan&amp;jnlp=experimento-vlan.jnlp">http://localhost:8080/NetLabWL/ServletExpV2?expName=Vlan&amp;ctx=vlan&amp;jnlp=experimento-vlan.jnlp</a> |

Fig. 4.10: Marcação no Cadastro de Experimento de VLAN.

| Resource Information |                                                                                                                                                                                                                               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name:                | Host Gaia                                                                                                                                                                                                                     |
| Description:         | <div>Micro-Computador Pentium III 600MHz; 256 KB RAM; 40 GB de Disco Rígido, Interface de Rede Ethernet 10/100/1000 Mbps.</div> <div>[NIC] <u>eth1</u></div> <div>[NIC] <u>eth2</u></div> <div>[SW-PORT] <u>eth1:13</u></div> |
| Active Link:         |                                                                                                                                                                                                                               |
| Info URL:            |                                                                                                                                                                                                                               |
| Responsable (Id):    |                                                                                                                                                                                                                               |

Fig. 4.11: Marcação no Cadastro Recurso.

não fizessem os experimentos já criados pararem de funcionar. Esse detalhe possibilita que novos experimentos sejam criados, sem que tenham que refatorar os já existentes, atendendo a um dos requisitos, que é ter um maior número de experimentos no domínio de redes de computadores.

Após ter efetuado todos os cadastros para incluir o experimento, basta copiar o arquivo *.jar* referente ao módulo do experimento para o diretório monitorado, assim, a arquitetura irá publicar todos os serviços contidos no módulo.

### 4.2.3 Reserva do Experimento

Somente experimentos reservados podem ser executados, por isso, antes de executar um determinado experimento, o usuário deve efetuar a reserva deste. Para executar a reserva, o usuário deve entrar no portal do Netlab, selecionar a opção *Supported Experiments* (essa opção, exige a autenticação do usuário) que está localizada na lista de opções do portal. Ao selecionar essa opção, serão listados todos os experimentos cadastrados no laboratório. Esses experimentos foram incluídos pela parte administrativa do GigaBOT (Projeto AccessService). A Fig. 4.13 ilustra o portal com a lista de experimentos disponíveis que são apresentados em forma de tabela com as seguintes informações:

- *name* - nome do experimento;
- *Description* - uma breve descrição do tipo de funcionalidades que o experimento provê;
- *Actions* - são as ações admitidas para o experimento;
- *Reservation* - opções relacionadas a reserva do experimento. A opção *List* permite selecionar uma data e listar a disponibilidade do experimento. A opção *Add* possibilita adicionar uma reserva para uma determinada data.



The screenshot shows the NetLab Web Lab interface. At the top, there's a logo and the text "NetLab Web Lab". Below it, a navigation bar includes "Home", "About", "Use", and "Supported Experiments". The main heading is "Internet Access to Network Laboratory". Below this, it says "Experiments for NetLab Web Lab". A table lists various experiments with columns for Name, Description, Actions, and Reservation.

| Name          | Description                                                                                                                                                               | Actions | Reservation |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|-------------|
| Rotas         | Configuracao avancada de tabela de rotas nos Hosts Helios, Urano, Gaia e Poseidon. [LINK] helios:eth3&gaia:eth1 [LINK] gaia:eth2&poseidon:eth2                            | Execute | List Add    |
| BB            | Aplicacao Java Web Start para uma implementacao do Bandwidth Broker nos Hosts Helios, Urano, Gaia e Poseidon. [LINK] helios:eth3&gaia:eth1 [LINK] gaia:eth2&poseidon:eth2 | Execute | List Add    |
| GeradorFluxos | [LINK] helios:eth3&gaia:eth1 [LINK] gaia:eth2&poseidon:eth2                                                                                                               | Execute | List Add    |
| Experimento1  | Configuracao de interface de dois hosts e teste de conectividade [LINK] helios:eth3&gaia:eth1                                                                             | Execute | List Add    |
| Experimento2  | Configuracao da interface de rede e tabela de rotas em tres hosts [LINK] helios:eth3&gaia:eth1 [LINK] gaia:eth2&poseidon:eth2                                             | Execute | List Add    |
| Rotas2        | Configuracao avancada de tabela de rotas nos Hosts Helios, Urano, Gaia e Poseidon. [LINK] helios:eth3&gaia:eth1 [LINK] gaia:eth2&poseidon:eth2                            | Execute | List Add    |
| VLAN          | Configuracao de VLans [SWITCH] name: 3Com, model: 4500, ports:46                                                                                                          | Execute | List Add    |

Fig. 4.12: Listagem de Experimentos do Portal Netlab.

Ao solicitar a opção de adicionar (*Add*) reserva, o usuário é direcionado para uma página com o calendário sobre o qual a reserva pode ser realizada. O usuário pode, então, marcar os horários para a execução. O horário de disponibilidade do experimento no laboratório também é gerenciado pelo *AccessService*. Este gerenciamento permite habilitar ou não um determinado experimento para reserva. Uma vez desabilitado o experimento, não é autorizada sua reserva. Essa opção é utilizada em casos de manutenção nos equipamentos do laboratório.

#### 4.2.4 Execução do Experimento

Com a reserva realizada, o usuário, dentro do período dela, já pode executar o experimento ao selecionar a opção *Execute*. O link referente a essa opção é o campo *Screenplay URL*, que é informado no cadastro do experimento. Essa URL redireciona para o *servlet* de experimento (*ServletExpV2*). O *ServletExpV2* recolhe as informações acerca da reserva do usuário e efetua a validação para garantir que o experimento só seja executado por usuários que tenham realizado a reserva. Caso o usuário esteja dentro do período reservado, o *ServletExpV2* redireciona a chamada para o *HttpServiceDispatcher*, passando como parâmetro as informações referentes ao experimento. Essas informações permitem que o *HttpServiceDispatcher* forneça o *download* da aplicação cliente que está contida no módulo do experimento. O *download* realizado pelo navegador do usuário é um arquivo de extensão JNLP (*Java Network Launching Protocol*), que faz com que toda aplicação e suas dependências sejam disponibilizadas na máquina do usuário. A Fig. 4.13 ilustra as interações relacionadas a execução do experimento.

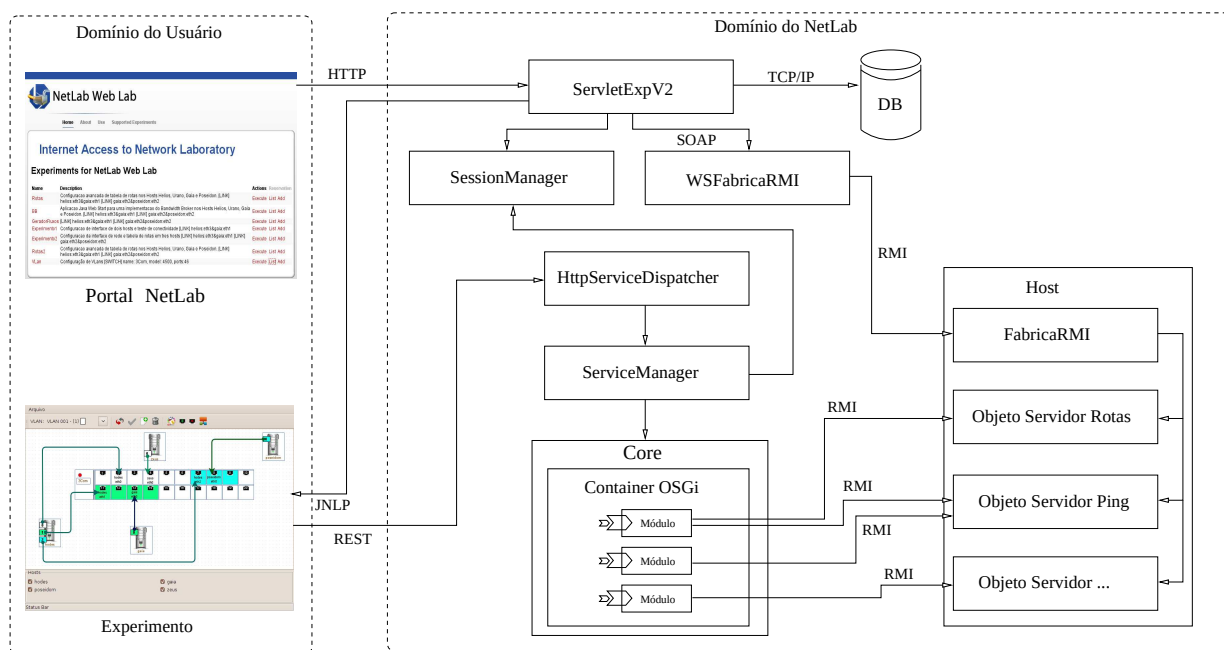


Fig. 4.13: Visão Geral das Interações Relacionadas a Execução do Experimento.

### Criação dos Objetos Remotos

O *ServletExpV2* tem outras funções, como a de criar os objetos remotos. Ao realizar a validação da reserva do experimento, é necessário instanciar todos os objetos remotos em todos os *hosts* que estão registrados como recursos, processo este realizado pela *WSFabricaRMI*. A *WSFabricaRMI* é um projeto Web e faz parte do Projeto NetLab, por isso, é hospedado no mesmo contêiner Web. Esse projeto disponibiliza, por meio de serviços Web, as funcionalidades para criar e destruir instâncias de objetos remotos nas máquinas do laboratório.

Para cada recurso do tipo *host* informado no cadastro do experimento, o *ServletExpV2* faz uma chamada remota, solicitando a criação do objeto remoto referente a cada recurso cadastrado como serviço. Em caso de alguma chamada remota falhar, o experimento não será executado.

Em cada *host* do laboratório, é registrada, via *rmiregistry*, uma instância do Projeto *FabricaRMI*. Os métodos remotos dessa fábrica são consumidos pelos *WSFabricaRMI* para registrar ou remover o registro dos objetos servidores referentes a cada serviço do experimento.

### Criação da Sessão de Interação

A sessão de interação é o elo entre a parte administrativa do NetLab com a arquitetura proposta. O ultimo passo antes de iniciar o *download* do experimento é a criação dessa sessão. Ela contém as informações acerca do usuário e da reserva do experimento. Na implementação da arquitetura, foi criado o *SessionManager*, que é responsável pela criação e gerenciamento das sessões de interações. A listagem 4.1 expõe os métodos da classe de gerência de sessão, e, em seguida, a descrição de cada método da classe é apresentada.

```

1
2 package br.ufu.facom.netlab.core;
3
4 import java.sql.Timestamp;
5 import java.util.ArrayList;
6 import java.util.Collection;

```

```
7 import java.util.HashMap;
8 import java.util.Map;
9
10 public class SessionManager {
11
12     private SessionManager() {...}
13
14     public static SessionManager getInstance() {...}
15
16     public String createSession(String userName, Timestamp endTime) {...}
17
18     public void putSessionProperty(String sessionId, String key, Object value)
19         {...}
20
21     public Object getSessionProperty(String sessionKey, String key) {...}
22
23     private SessionInfo getSessionInfo(String sessionId) {...}
24
25     public boolean isValidSession(String sessionId) {...}
26
27     private static class SessionInfo {
28         long createTime;
29         long endTime;
30         String userName;
31         String id;
32         Map<String, Object> properties;
33
34         SessionInfo(String userName, long endTime) {...}
35
36         void putProperty(String key, Object value) {...}
37
38         Object getProperty(String key) {...}
39     }
40
41     private class SessionMonitor implements Runnable {
42
43         @Override
44         public void run() {...}
45     }
46 }
```

---

Listagem 4.1: Classe de Gerenciamento de Sessões: SessionManager.java

- *getInstance()* - retorna a instância da classe. Padrão de projeto *Singleton*;
- *createSession()* - esse método cria o objeto *SessionInfo*, adiciona-o na lista de sessões e retorna a chave da sessão que foi criada;
- *putSessionProperty()* - permite adicionar um objeto a sessão dada uma chave;
- *getSessionProperty()* - retorna um objeto que foi adicionado a sessão;
- *isValidSession()* - retorna verdadeiro caso a sessão relacionada a chave, passada como parâmetro ainda estiver válida;

- *removeSession()* - remove da lista de sessões a sessão associada a chave informada no parâmetro.

A implementação da gerência da sessão de interação foi toda agrupada nessa única classe. Por isso, ela possui duas classes internas (*Inner Class*): *SessionInfo* e *SessionMonitor*. A *SessionInfo* é que encapsula as informações relacionadas a sessão, conforme apresentadas a seguir:

- propriedades
  - createTime* - data é hora de criação da sessão;
  - endTime* - data é hora de termino da sessão;
  - userName* - nome do usuário que relacionado a sessão criada;
  - id* - identificador da sessão. É composto pelo nome do usuário + *createTime* + *endTime* + um número aleatório;
  - properties* - armazena propriedades na sessão;
- métodos
  - putProperty()* - adiciona um objeto a sessão;
  - getProperty()* - retorna o objeto adicionado na sessão caso exista;

A classe interna *SessionMonitor* é uma *Thread* que fica monitorando todas as sessões contidas na lista, em que cada item da lista de sessões é uma instância da classe *SessionInfo*, que contém as informações acerca da sessão. O processo de verificação é executado a cada 30 segundos e percorre toda a lista de sessões, analisando se o tempo de término não expirou. Caso tenha expirado, a sessão é removida da lista, e ao ser removida da lista o método *isValidSession* retorna falso, o que aponta que não existe uma sessão válida relacionada a chave que é passada como parâmetro.

A implementação do *ServiceManager* deve validar se existe uma sessão válida para o usuário antes de executar qualquer serviço. Por tanto, o método *isValidSession()* é executado sempre antes que o *ServiceManager* solicite a sua execução ao *Core*. Isso garante que experimentos em execução em máquinas clientes possam executar serviços após finalizado o tempo de uso do experimento.

Ao ser criada a sessão de interação, o método *createSession()* retorna o identificador da sessão, esse identificador é passado como parâmetro para a aplicação do cliente por meio da URL, que inicia o *download* do experimento. Toda execução de serviço por parte da aplicação cliente deve enviar o identificador da sessão para que o gerenciador de serviços possa validar a sessão.

Após ter executado o ultimo passo, que é a criação da sessão, é iniciado o *download* da aplicação para a máquina do usuário.

### Características da Aplicação de Configuração de VLANs

Com a aplicação do experimento já carregada na máquina, o usuário já pode interagir com o laboratório remoto. A aplicação fornece todos os recursos necessários para a manutenção de VLANs. A Fig. 4.14 apresenta a tela da aplicação cliente do experimento.

A interface da aplicação é dividida em três partes:

1. barra de ferramentas - parte superior da tela;
2. área gráfica - centro;
3. lista de hosts - parte inferior.

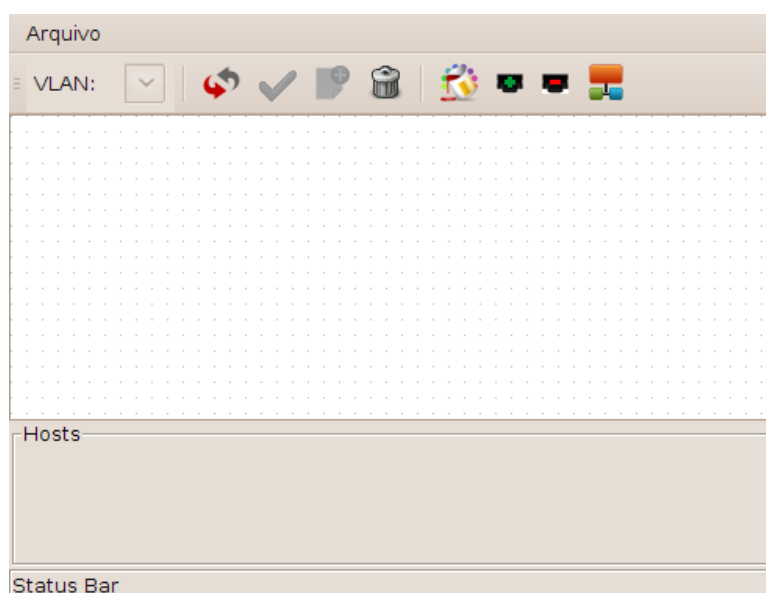


Fig. 4.14: Aplicação Cliente do Experimento.

A barra de ferramentas traz as seguintes opções da esquerda para direita:

- lista das VLANs criadas. É apresentado a descrição, id e a cor selecionada para a VLAN;
- atualizar;
- aplicar;
- incluir VLAN;
- excluir VLAN;
- altera cor da VLAN selecionada;
- adiciona portas a VLAN selecionada;
- remove portas da VLAN selecionada;
- altera visão para mapa da rede.

Apesar da aplicação estar em execução na máquina do usuário, nenhuma informação relacionada as VLANs do *switch* foram carregadas. Contudo, a mesma já possui a lista de *hosts* com suas respectivas interfaces e a quais portas essas estão conectadas e também as informações do *switch* (quantidades de portas, marca, modelo).

Quando o usuário clica no botão de Atualizar, a aplicação executa um *Web Service* REST solicitando a lista de VLANs cadastradas no *switchs*. O serviço executado faz parte do experimento de configuração de VLANs. A opção de carga das redes virtuais deve acessar diretamente o *switch* para carregar as redes criadas no equipamento. Entre as várias interfaces de administração do equipamento, a interface HTTP foi a selecionada para que o módulo do experimento interaja com o *switch*. Para carregar a lista de VLANs, o serviço do experimento faz uma requisição HTTP no *switch*. A resposta da requisição é, então, processada a fim de extrair somente as informações relacionadas as redes.

O serviço de carga de VLANs retorna um XML, contendo todas as VLANs criadas. Ao receber o arquivo XML, a aplicação cliente faz o *parsing*, carrega a lista de VLANs e cria a representação

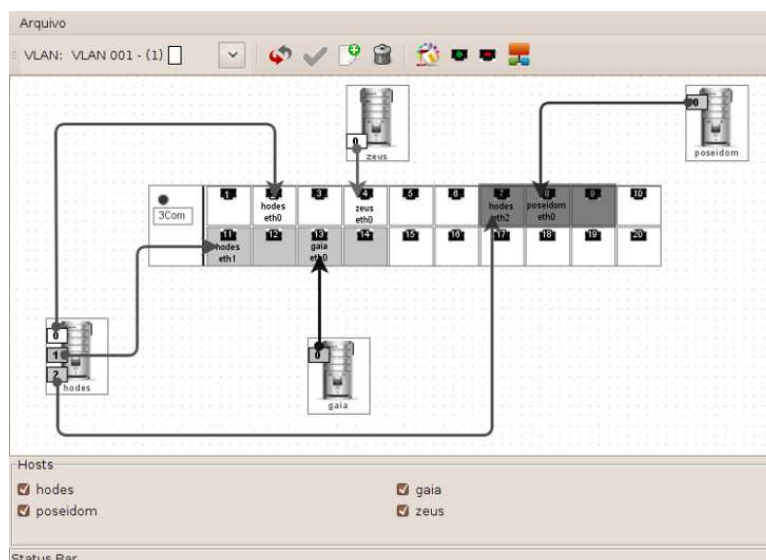


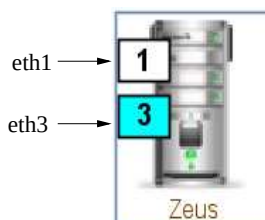
Fig. 4.15: Aplicação Cliente Após Carregadas as VLANs.

gráfica de um *switch* e os *hosts*, com suas respectivas placas de redes conectadas nas portas do *switch*. A Fig. 4.15 demonstra a aplicação após carregada a lista de VLANs.

A parte gráfica (centro da tela) permite que todos os objetos sejam arrastados, inclusive, as ligações entre os *hosts* e as portas do *switch*. Essa característica possibilita que o usuário configure sua visualização para facilitar sua interpretação.

Na parte inferior da tela, é apresentada a lista de todos os *hosts* que fazem parte do experimento com opção para marcar e desmarcar. Essa marcação propicia visualizar ou não o *host* no gráfico. Com uma quantidade pequena de máquinas integrantes ao experimento, fica simples a visualização do gráfico, contudo se o experimento tiver um grande número de máquinas, a configuração se torna mais difícil. Por isso, essa opção facilita o uso do experimento por permitir a visualização somente das máquinas que se deseja configurar.

Ao criar uma VLAN, é necessário selecionar uma cor. Essa cor facilita a identificação das VLANs no gráfico. Cada *host* do experimento é representado por uma imagem (Fig.4.16) e, dentro da imagem, um retângulo com um número que representa a placa de rede conectada a porta do *switch*. O retângulo referente a placa tem como preenchimento a mesma cor da VLAN a qual esta conectado.

Fig. 4.16: Representação gráfica de um *Host* e suas NICs.

A lista de VLAN tem uma característica específica. Quase todas as ações na barra de ferramentas são relacionadas a VLAN selecionada na lista. Por exemplo, os botões de adicionar e remover portas executam suas ações na VLAN que estiver selecionada na lista. Ao pressionar o botão de adicionar portas, este se mantém pressionado até que seja clicado novamente. Enquanto estiver pressionado, basta selecionar as portas do *switch* que a mesma será adicionada a VLAN selecionada, essa ação muda automaticamente a cor da porta para a mesma cor da VLAN. O botão de remover porta tem a

mesma característica do adicionar, com exceção que, ao remover uma porta, esta passa a fazer parte da VLAN padrão (VLAN com identificador 1 e cor branca).

Toda alteração realizada na aplicação não é aplicada automaticamente no laboratório. É necessário pressionar o botão Aplicar. Essa característica é indispensável, para evitar que toda ação execute um serviço remoto. Especificamente, serviços de interação com o *switch*. Por ser um *hardware* com processamento bem limitado, as requisições HTTP demoram a ser processadas. Por esse motivo, a aplicação permite que seja realizada toda configuração e só será executado o serviço, quando for pressionado o botão aplicar.

No desenvolvimento da aplicação de experimento, optou-se por não utilizar várias telas e sim painéis que são alternados uns com os outros. Como, por exemplo, a opção de Mapa da Rede, que possibilita uma visualização simplificada de todos os *hosts* e VLANs. No painel de configuração, todos os *hosts* são apresentados com suas respectivas interfaces de redes conectadas as portas do *switchs*. No mapa da rede em vez de apresentar o *switch*, é exibida a VLAN e, nos *hosts*, as ligações com essas VLANs. O mesmo esquema de cores utilizado para diferenciar as redes também é aplicado nesse painel. A Fig. 4.17 demonstra o painel de mapa da rede.

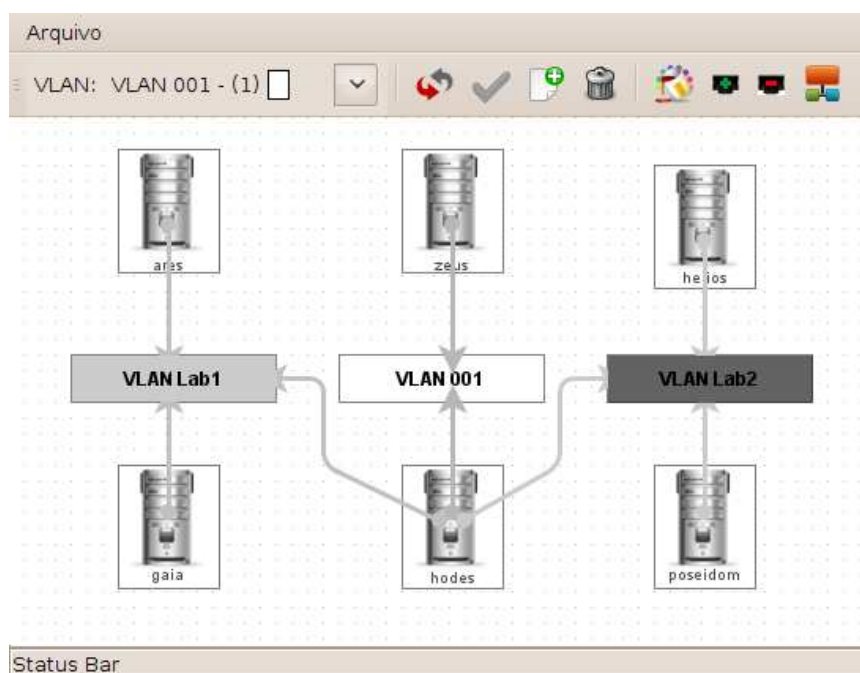


Fig. 4.17: Painel de Mapa da Rede.

O mapa da rede exposto possui seis *hosts* e três VLANs. Os *hosts* ares e gaia estão conectados a VLAN Lab 1, helios e poiseidon conectados a VLAN Lab2. O *host* zeus está conectado a VLAN 001 (VLAN padrão). O *host* hodes está conectado as três VLANs. Em resumo, o experimento criou três redes com duas máquinas em cada rede, sendo que o hodes conectado a todas as VLANs, o que permite que ele seja utilizado como *gateway* entre as redes para prover a comunicação entre as redes.

## 4.3 Cenário #1 Criação de VLAN

Esse cenário descreve a criação de uma VLAN e os procedimentos que devem ser realizados para inclusão de porta a VLAN criada.

Ao selecionar o botão de Incluir Nova VLAN, a aplicação alterna para o painel de criação de VLAN, que possui os campos de: ID, Descrição e Cor, conforme ilustra a Fig. 4.18.



Fig. 4.18: Pannel de Criação de VLANs.

Como exemplo, foi incluída a VLAN de id: 102, com a descrição: VLAN Lab C, e selecionada a cor para representar a rede virtual criada. Depois de informados os dados ao selecionar o botão de salvar, são realizadas duas validações, uma é de não permitir que seja utilizado o identificador 001, a outra é a utilização da cor branca. As duas validações são necessárias, pois a VLAN de identificador 001 é a rede padrão do *switch*, por isso, sempre haverá essa rede criada. Por esse motivo, foi atribuída a cor branca para representá-la. Caso passe pelas validações, a aplicação alterna, automaticamente, para o painel de configuração, em que a VLAN recém criada pode ser selecionada para a adição de portas.

Para incluir portas a VLAN, são pertinentes os seguintes passos:

1. selecionar a VLAN desejada na lista de VLANs;
2. selecionar no botão de inclusão de portas;
3. selecionar as portas desejadas na representação gráfica do switch.

O experimento é carregado já com as informações sobre quais portas são utilizadas pelos *hosts* que fazem parte do experimento. O que torna fácil a configuração das redes. Quando está no modo de inclusão de portas (botão Incluir portas pressionado), ao selecionar uma determinada porta no *switch*, esta tem a cor de fundo alterada para a mesma cor da VLAN selecionada.

A Fig.4.19 ilustra a inclusão de portas para a VLAN Lab C. No exemplo, foram incluídas as portas, 4, 26, 27, 28, 29 e 33.

Após configurada a VLAN com suas respectivas portas, basta selecionar o botão de Aplicar, a aplicação gera um arquivo XML, contendo todas as informações de VLAN e portas, e executa o serviço REST, que envia o XML utilizando a opção *POST* do protocolo HTTP, esse serviço compõe o módulo do experimento e é responsável em aplicar todas as configurações ao *switch*.

## 4.4 Cenário #2 Configuração de 3 VLANs Inter-conectadas

Esse cenário descreve como se utiliza a aplicação de configuração de VLANs para a criação de três redes virtuais inter-conectadas. Cada rede possui pelo menos um *host* conectado. Para validar as conexões entre as redes, a aplicação faz uso do serviço de *Ping*, que permite executar testes de conectividade entre qualquer máquina do experimento pelo painel de Mapa da Rede.



Fig. 4.19: Painel de Inclusão de Portas a uma VLANs.

O experimento possui, em seu cadastro, os seguintes recursos físicos (*hosts*):

1. Gaia;
2. Poseidon;
3. Zeus;
4. Helios.

Para permitir a execução do teste de conectividade entre as redes, os seguintes recursos lógicos foram incluídos:

1. *Ping*;
2. *Ifconfig*.

O serviço de *Ping* é responsável em executar o teste de conectividade entre as redes, contudo, para obter os dados referentes as interfaces dos *hosts*, o serviço de *Ifconfig* também é executado para cada sub-recurso do tipo NIC que estiver informado no cadastro do recurso *host*.

Na execução do experimento, criam-se as seguintes VLANs, com os respectivos *hosts*:

1. VLAN Lab A - Gaia e Helios;
2. VLAN Lab B - Helios e Zeus;
3. VLAN Lab C - Zeus e Poseidon.

A tabela Tab.4.3 apresenta as características de cada rede e *host* do cenário.

A ligação entre as redes é realizada pelos *hosts* Helios e Zeus, e ambos servem de *gateway* para repassar pacotes de uma rede para outra. Para configurar os endereços de cada interface de rede, bem como as rotas entre elas, foi utilizado o experimento já existente no NetLab, que é o experimento de configuração de Rotas apresentado por Farias [2]. A representação gráfica das redes criadas é exibida na Fig.4.20.

| Rede  | Host     | Porta | Interface | IP            |
|-------|----------|-------|-----------|---------------|
| Lab A | Gaia     | 26    | eth2      | 172.16.30.2   |
|       | Helios   | 29    | eth1      | 172.16.30.102 |
| Lab B | Zeus     | 40    | eth1      | 172.16.30.103 |
|       | Helios   | 38    | eth3      | 172.16.50.102 |
| Lab C | Poseidon | 11    | eth1      | 172.16.20.1   |
|       | Zeus     | 15    | eth3      | 172.16.20.103 |

Tab. 4.3: Características das Redes e Hosts do Cenário #2.

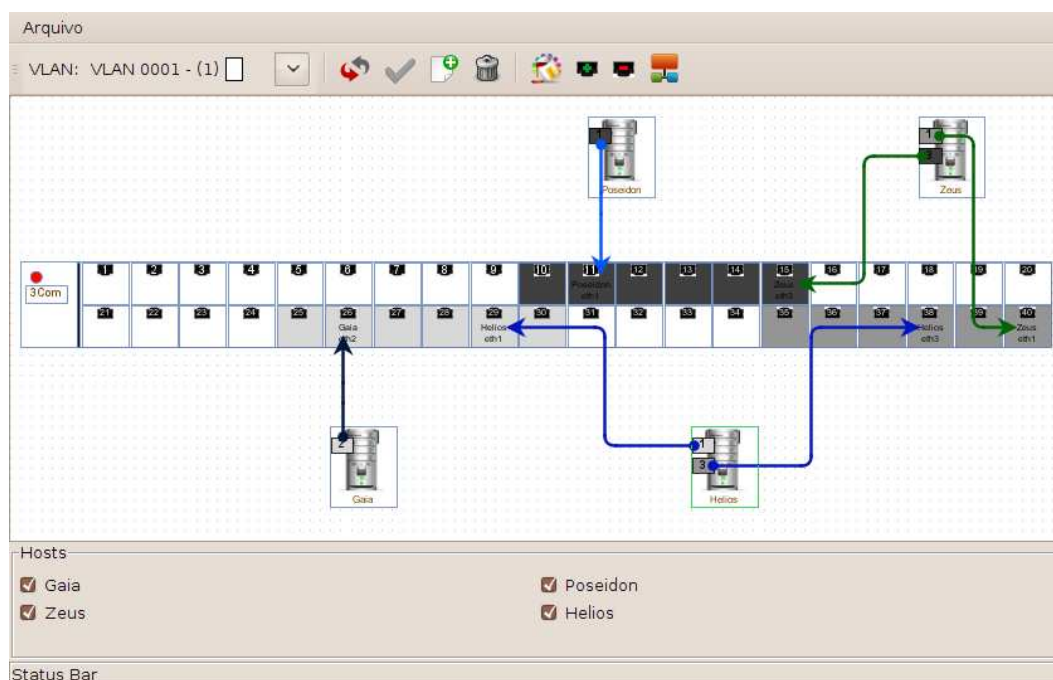


Fig. 4.20: Distribuição dos Hosts por VLANs.

A Fig.4.21 ilustra a interligação entre as redes. O *Helios* interliga a rede Lab A com a rede Lab B, o que propicia a conectividade com qualquer *host*, em qualquer uma das duas redes. Zeus interliga a rede Lab B com a rede Lab C, o que permite a conectividade entre as três redes, pois Helios liga a rede A com B, e Zeus a rede B com C, isso permite que um pacote do *Host* Gaia com destino a Poseidon seja entregue.

Para validar o cenário apresentado, a aplicação permite executar o teste de conectividade, que consiste em executar um serviço de *ping* de uma máquina para outra. Ao ser carregado o mapa da rede (a carga é realizada sob demanda, ao selecionar o botão Mapa da Rede), o serviço de *Ifconfig* é executado para obter o endereço *IP* referente a cada interface da máquina. Esse serviço faz parte do módulo do experimento e é executado por uma requisição HTTP do tipo *GET*. Para realizar o teste de conectividade, deve-se selecionar o *host* de origem. Após selecionado, utiliza-se o botão direito do *mouse* e, em seguida, é apresentado um *popup* com as interfaces e os respectivos *IPs*. Ao selecionar qualquer opção, é apresentado um sub-menu com a lista de todos os *hosts* que fazem parte do experimento, selecionado o *host* é apresentado as interfaces de rede e *IP* do *host* de destino. A Fig.4.22 demonstra a execução do teste de conectividade, tendo como origem *Gaia - eth2 - 172.16.30.2* e destinado a *Poseidon - eth1 - 172.16.20.1*.

O resultado do teste é exporto em uma tela (ilustrada na Fig.4.23) com o retorno do comando *ping* executado da máquina Gaia para Poseidon.

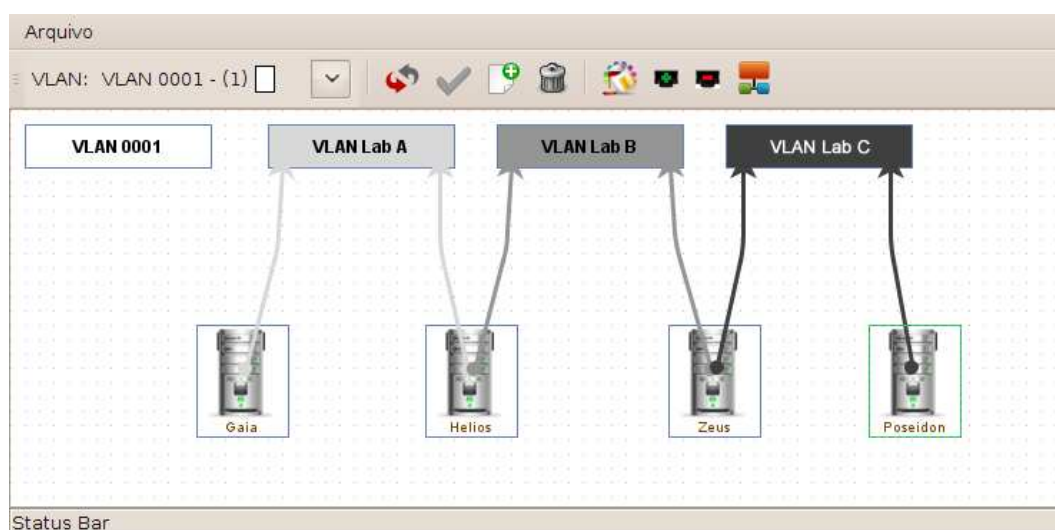


Fig. 4.21: Interligações Entre as Redes.

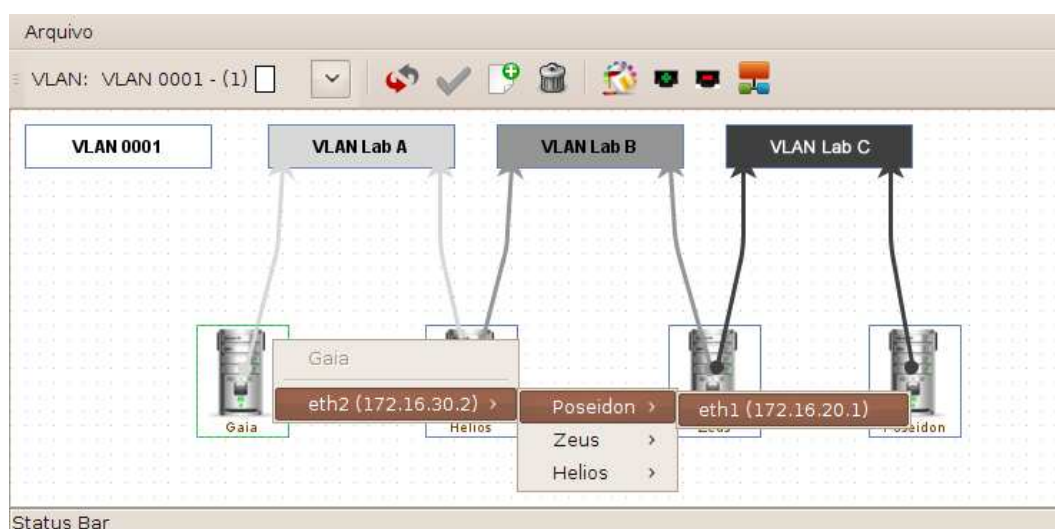


Fig. 4.22: Execução do Teste de Conectividade de Gaia para Poseidon.

O comando *ping* executado no teste de conectividade utiliza o parâmetro de *-R (Record Route)* o que permite visualizar a rota percorrida pelo pacote da origem até o destino.

## 4.5 Cenário #3 Configuração de 3 VLANs sem Inter-conexão

Esse cenário descreve a configuração de 3 redes dentre as quais apenas 2 são interconectadas. As máquinas para esse cenário são as mesmas utilizadas no cenário #2, bem como as redes e configurações das máquinas.

Para montar o cenário, é necessário excluir uma das máquinas que faz o papel de *gateway* (Helios e Zeus) de uma das redes. Essas duas máquinas estão conectadas a duas redes diferentes, por isso, para retirar uma dessas máquinas da rede, é preciso excluir a porta a qual a máquina está conectada da VLAN. A máquina selecionada, para ser removida da rede, foi Zeus, que está conectado na rede Lab B e Lab C. Para excluir Zeus da rede Lab B, os seguintes passos devem ser seguidos:

1. executar o experimento de configuração de VLAN;



Fig. 4.23: Resultado do Teste de Conectividade ente Gaia e Poseidon Cenário #2.

2. carregar as informações de VLAN (selecionar a opção de carregar na barra de ferramentas);
3. selecionar a rede LAB B na lista;
4. selecionar o botão de Remover portas da VLAN;
5. selecionar a porta 40 do *switch*;
6. selecionar o botão aplicar.

Após executados os passos apresentados, Zeus não faz mais parte da rede Lab C, conforme ilustra a Fig.4.24. A porta 15 a qual Zeus está conectado pela interface eth3, agora, faz parte da VLAN padrão (VLAN 001).

A rede Lab C ficou isolada das outras redes por não possuir mais um *gateway*, que era o papel de Zeus. A Tab.4.4 apresenta as características das redes e *hosts* para este cenário, em que Poseidon fica isolado das outras redes.

| Rede  | Host     | Porta | Interface | IP            |
|-------|----------|-------|-----------|---------------|
| Lab A | Gaia     | 26    | eth2      | 172.16.30.2   |
|       | Helios   | 29    | eth1      | 172.16.30.102 |
| Lab B | Zeus     | 40    | eth1      | 172.16.30.103 |
|       | Helios   | 38    | eth3      | 172.16.50.102 |
| Lab C | Poseidon | 11    | eth1      | 172.16.20.1   |

Tab. 4.4: Características das Redes e Hosts do Cenário #3.

Para validar a configuração realizada, o mesmo teste de conectividade descrito no cenário #2 foi executado, tendo como origem o *host* Gaia da rede Lab A e destino Poseidon na rede Lab C. O Mapa da Rede para esse cenário é ilustrado pela Fig.4.25.

A execução do teste de conectividade resultou em falha, conforme ilustra a Fig.4.26, pois Gaia não pertence a mesma rede de Poseidon e não é possível chegar da rede Lab A até a rede Lab C.

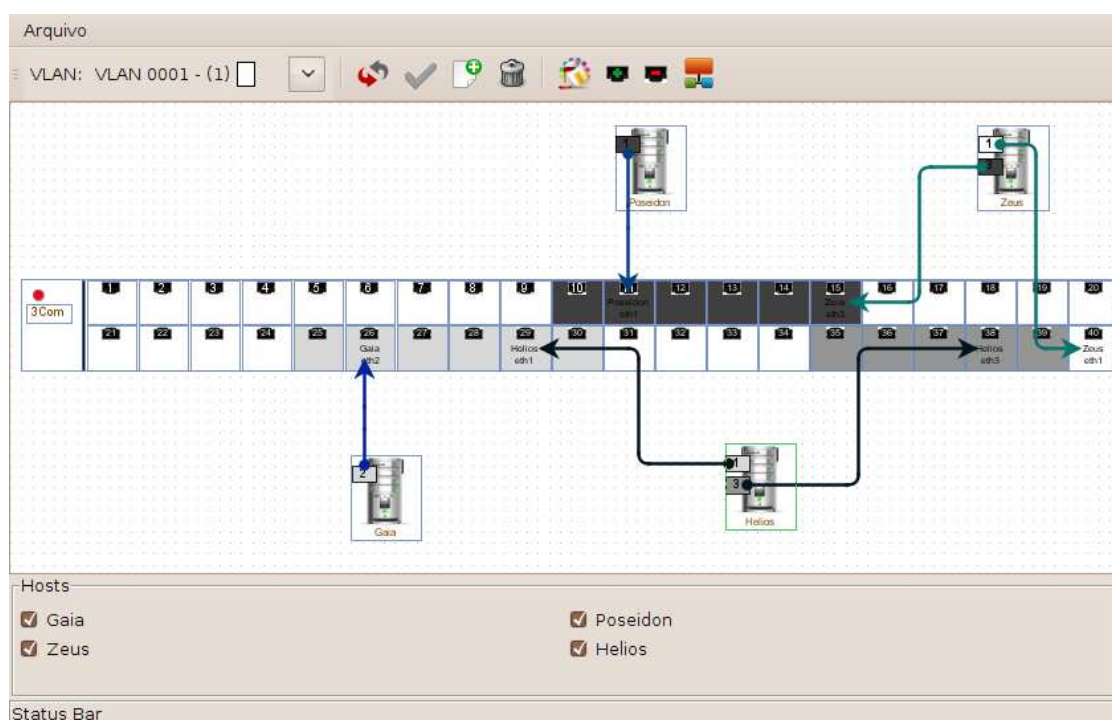


Fig. 4.24: Zeus Fora da Rede Lab B.

## 4.6 Considerações Finais

A implementação do experimento de configuração de VLANs permitiu validar a arquitetura e suas funcionalidades. Os cenários apresentados possibilitarão a validação de toda a arquitetura, pois, ao executar o experimento, todos os componentes implementados foram acionados, desde a carga da aplicação pelo portal até a execução dos serviços pela aplicação. Os cenários comprovam o funcionamento da arquitetura, e foram elaborados levando em consideração as várias formas de interação entre as máquinas e equipamentos disponibilizados no laboratório.

A nova arquitetura foi incluída no Projeto NetLab sem prejudicar os experimentos já criados, admitindo a coexistência das duas formas de criação e execução e disponibilização de experimentos. Essa coexistência permite estender o uso do laboratório, pois, possibilita aumentar o número de experimentos criados e várias formas de criar tais experimentos.

A integração com a parte administrativa do NetLab não causou nenhuma alteração, pois seguiu os mesmos princípios da criação dos experimentos do Projeto NetLab e utilizaram-se os campos de descrições dos recursos e experimentos para incluir informações adicionais requeridas pelo experimento.

A publicação dinâmica de serviços, seguindo os princípios REST, possibilitou uma melhor escalabilidade, por não exigir uma implementação da pilha SOAP, tanto no cliente quanto no servidor, além de diminuir consideravelmente os dados trafegados nas execuções de serviços. Nos serviços REST, somente o *payload* é transmitido/recebido. A simplicidade na implementação da aplicação cliente para consumir os serviços REST também traz grandes benefícios, pois faz uso apenas do protocolo HTTP e as funcionalidades providas por ele.

A disponibilização de um novo experimento na nova arquitetura se mostrou extremamente simples e eficaz. Um experimento incluindo sua aplicação cliente é disponibilizado em um único arquivo, que é um módulo (*bundle*) OSGi, o que permite uma facilidade no controle e disponibilização dos experimentos.

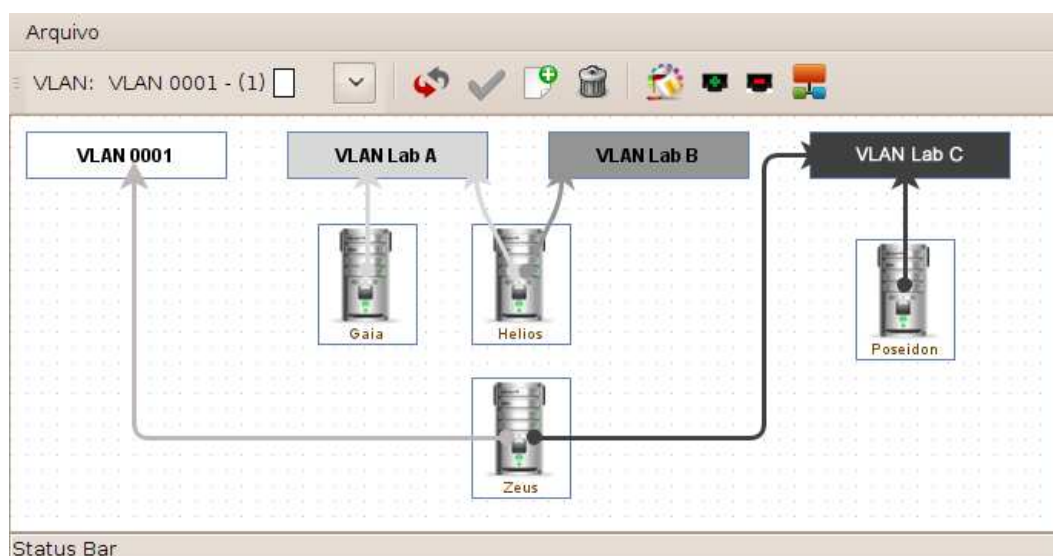


Fig. 4.25: Mapa da Rede Cenário #3.

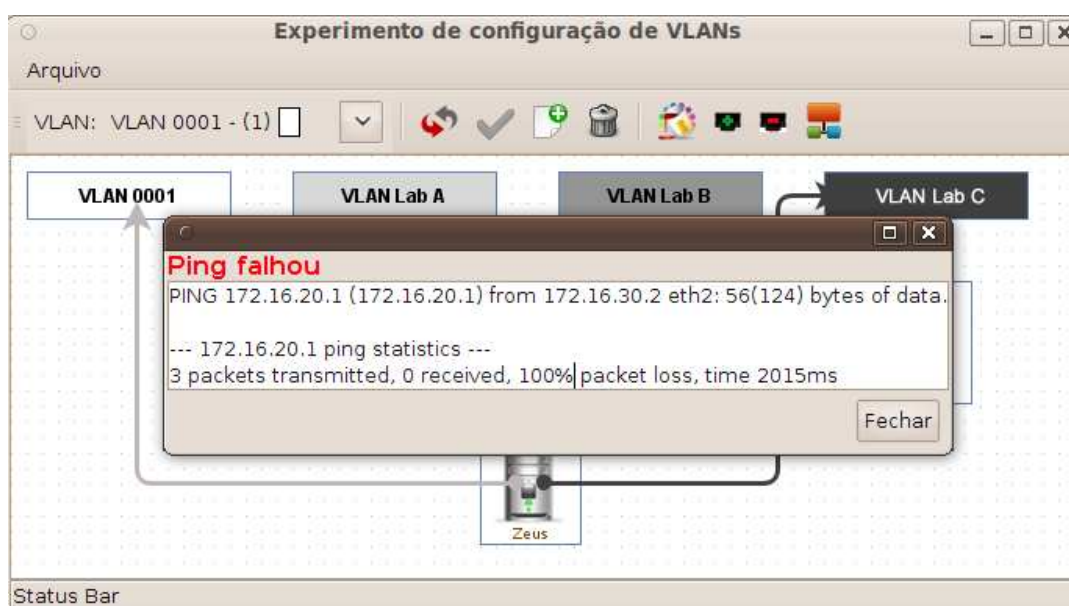


Fig. 4.26: Resultado do Teste de Conectividade ente Gaia e Poseidon. Cenário #3.

# Capítulo 5

## Conclusão e Trabalhos Futuros

Este capítulo apresenta as considerações finais sobre trabalho. Iniciando com uma retrospectiva e avaliação das contribuições, por fim, os trabalhos futuros.

### 5.1 Retrospectiva das Contribuições

A adoção e uso de WebLabs por instituições de ensino está diretamente ligada à facilidade de implantação, manutenção desses laboratórios remotos e desenvolvimento de experimentos. Uma arquitetura com suporte a carga de módulos dinâmicos, como proposta neste trabalho, contribui para a adoção e uso de WebLabs por facilitar a gerência e criação de experimentos de forma independente da arquitetura, e até de outros experimentos, levando em consideração a escalabilidade.

Na sequência, destacam-se contribuições relacionadas à utilização da carga de módulos dinâmicos, bem como da arquitetura orientada a serviços:

1. utilizar suporte a módulos dinâmicos para aumentar a disponibilidade de experimentos, bem como a disponibilidade do próprio laboratório (7 dias por semana e 24 horas por dia);
2. utilizar padrões de desenvolvimento baseados em serviços sem que se comprometa a escalabilidade do laboratório e de seus experimentos;
3. propiciar interoperabilidade por meio da utilização de protocolos não bloqueados por *firewalls*;
4. aumentar a dinamicidade e adaptabilidade dos experimentos, possibilitando ao usuário o emprego de topologias de redes dinâmicas e adaptáveis aos experimentos utilizados;
5. proporcionar independência de um experimento em relação a outros experimentos, para garantir que nenhum experimento interfira no funcionamento do laboratório, contribuindo, assim, com o aumento da disponibilidade;
6. facilitar a criação de novos experimentos bem como, a disponibilização de experimentos mediante o controle de versões sem perda da disponibilidade.

### 5.2 Avaliação das Contribuições

Um dos principais motivos para adoção de WebLabs pelas instituições de ensino é a maximização do uso de equipamentos de laboratórios fisicamente instalados. Neste contexto, a disponibilidade

de experimentos, bem como a disponibilidade do próprio laboratório constituem requisitos funcionais imprescindíveis. Outro aspecto igualmente importante é a independência que um experimento deve ter de um outro, ou seja, um novo experimento não deve interferir em nenhum componente da arquitetura. Isto denota que o desenvolvedor do experimento não necessita de acesso a componentes da arquitetura e, assim, tem-se a garantia de que experimento algum interfira no funcionamento do laboratório.

A arquitetura proposta buscou contemplar os requisitos acima descritos e referenciados no Cap. 2 (Seção 2.6). Os que não foram totalmente acomodados foram considerados na definição e na implementação dessa arquitetura, para facilitar o desenvolvimento de trabalhos futuros.

O uso de tecnologias no estado da arte, como REST, na comunicação entre a aplicação cliente do experimento e o servidor, possibilitou atender aos requisitos de interoperabilidade e escalabilidade, pois serviços Web que seguem os princípios arquiteturais REST, quando utilizados, trafegam apenas o *payload* sem nenhum *overhead*. Apenas os dados necessários para a troca de mensagens são necessários. Além disso, não é preciso que haja uma implementação da pilha SOAP no cliente nem no servidor, o que diminui o processamento, permitindo que um volume maior de requisições seja atendido.

A utilização de um *framework* OSGi para acomodar os experimentos atendeu ao requisitos de padronização de experimentos, pois um experimento passa a ser disponibilizado como um módulo (*bundle*), não sendo necessário mais nenhum outro artefato para sua utilização. Ao criar um novo experimento para disponibilizá-lo no WebLab, é necessário apenas copiar o arquivo para o diretório de módulos e incluir o experimento no portal administrativo (*AccessService*).

O requisito de disponibilidade também é atendido na arquitetura. Os módulos de experimento passam a ser versionados, ou seja, o contêiner OSGi permite que versões diferentes do mesmo módulo sejam instalados, mantendo a compatibilidade com aplicações clientes. Ao ser disponibilizada uma versão nova de um módulo já instalado no contêiner e caso exista algum serviço desse módulo em execução, o *framework* aguarda até que a execução seja finalizada para iniciar a nova versão. Isso possibilita que o usuário continue executando a aplicação cliente ao mesmo tempo em que o experimento que está sendo executado seja atualizado.

A arquitetura foi definida de modo que experimentos não necessitem de acesso aos componentes do núcleo da arquitetura, garantindo-se, assim, o isolamento dos experimentos para com a arquitetura. Caso algum experimento falhe por mau funcionamento, outros experimentos que estejam em execução não são afetados.

Os componentes da arquitetura foram projetados de forma a facilitar sua extensão, o que torna a arquitetura proposta genérica o suficiente para ser utilizada em laboratórios, independente do domínio de redes de computadores.

Os objetivos listados acima foram alcançados e validados. Os vários cenários em que se exploraram a criação e a remoção de VLANs possibilitaram a validação da implementação da arquitetura proposta. Durante a fase de desenvolvimento e testes do experimento, foram geradas várias versões que foram publicadas no laboratório, o que possibilitou comprovar os benefícios da utilização da arquitetura.

A arquitetura foi implementada no NetLab de modo a manter a arquitetura anterior para aproveitar os experimentos já disponibilizados. A integração entre as duas arquiteturas, também pode ser validada no experimento criado, pois fez-se necessária a utilização de dois serviços já existentes para compor o experimento: serviço de configuração de NICs *ifconfig* e de conectividade *ping*.

Em resumo, os requisitos contemplados pela arquitetura proposta são listados a seguir:

- suporte a múltiplos protocolos;
- padronização dos experimentos;

- maior grau de escalabilidade;
- configuração em tempo de execução;
- compartilhamento de recursos.

## 5.3 Trabalhos Futuros

Um passo para facilitar a integração entre WebLab de outras instituições, ou simplesmente executar experimentos em outras redes, é permitir que a comunicação entre os módulos de experimentos não fique limitada a utilizar o Protocolo RMI. Hoje, a comunicação entre o servidor e os vários computadores que compõem a rede interna do laboratório é realizada por meio do Protocolo RMI.

Nesse contexto, um dos trabalhos futuros é disponibilizar a mesma arquitetura utilizada no servidor do NetLab nos *hosts* do laboratório. No presente trabalho, os componentes são executados dentro de um contêiner Web. Nesse mesmo servidor, é acomodado o portal de acesso ao laboratório e o portal de gerenciamento (*AccessServices*).

A sugestão é que a aplicação do laboratório (implementação de referência da arquitetura) seja executada nos *hosts* como uma aplicação *standalone* (fora do contêiner Web). Dessa forma, é possível que os objetos remotos responsáveis em interagir com os equipamentos também sejam disponibilizados como módulos (*bundles*), fazendo uso de todos os benefícios já apresentados.

Em um segundo passo, a implementação de federação no NetLab pode se beneficiar da sugestão anterior. Outra sugestão é a inclusão de máquinas virtuais como *hosts* do laboratório, o que possibilita a utilização de um número maior de máquinas no laboratório.

## Trabalhos Publicados pelo Autor

1. Souza, D. V.; Nogueira, P. E.; da Silva Leite Coelho, P., Faina, L. F. Uma Arquitetura com Suporte a Módulos Dinâmicos para WebLab no Domínio de Redes de Computadores, XXXVI Conferência Latinoamericana de Informática, Assunção - Paraguai. CLEI2010, Qualis B4.
2. Souza, D. V.; Nogueira, P. E.; da Silva Leite Coelho, P. R., Faina, L. F. Uma Arquitetura com Suporte a Módulos Dinâmicos para WebLab, Criciúma - SC, SulComp 2010.

## Trabalhos Submetidos

1. Souza, D. V.; Nogueira, P. E.; da Silva Leite Coelho, P., Faina, L. F. Uma Arquitetura Escalável para WebLab no Domínio de Redes de Computadores, 29o. *Simpósio Brasileiro de Redes de Computadores*, 2011, Qualis B3
2. Souza, D. V.; Nogueira, P. E.; da Silva Leite Coelho, P., Faina, L. F. An Architecture with Support for Dynamic Modules to WebLab in the Field of Computer Networks, *CLEI Electronic Journal - CLEIej*, ISSN 0717- 5000, 2011, Qualis C

# Referências Bibliográficas

- [1] Javier García Zubía, Diego López de Ipiña, e Pablo Orduña. “Evolving towards better architectures for remote laboratories: a practical case”. *ijoe*, 1(2), 2005.
- [2] Adriano Fiad Farias. “Laboratório de Experimentação Remota para Ensino de Redes de Computadores”. Dissertação de Mestrado, Universidade Federal de Uberlândia - UFU Faculdade de Computação, Setembro 2008.
- [3] L. R. Agostinho, A. F. Farias, L. F. Faina, E. G. Guimarães, P. R. S. L. Coelho, e E. Cardozo. “Uma Proposta de Arquitetura para Experimentos DiffServ em Web Labs”. *XXXIV Conferência Latinoamericana de Informática*, 2008. Santa Fé - Argentina. CLEI2008.
- [4] Lucio Agostinho Rocha. “WebLab SOA no Domínio de Redes de Computadores para Experimentos DiffServ”. Dissertação de Mestrado, Universidade Federal de Uberlândia - UFU Faculdade de Computação, Fevereiro 2009.
- [5] Projeto TIDIA/KyaTera. “Laboratórios de Acesso Remoto sobre Redes Avançadas.”, Sep. 2008. <http://kyatera.incubadora.fapesp.br>.
- [6] V.J. Harward, J.A. del Alamo, S.R. Lerman, P.H. Bailey, J. Carpenter, K. DeLong, C. Felknor, J. Hardison, B. Harrison, I. Jabbour, P.D. Long, Tingting Mao, L. Naamani, J. Northridge, M. Schulz, D. Talavera, C. Varadharajan, Shaomin Wang, K. Yehia, R. Zbib, e D. Zych. “The iLab Shared Architecture: A Web Services Infrastructure to Build Communities of Internet Accessible Laboratories”. *Proceedings of the IEEE*, 96(6):931 –950, 2008.
- [7] J. García-Zubía, D. Lopez-de Ipiña, P. Orduna, e U. Hernandez-Jayo. “Experience with WebLab-Deusto”. In *Industrial Electronics, 2006 IEEE International Symposium on*, volume 4, páginas 3190–3195, Julho 2006.
- [8] El-S. S. Aziz, S. K. Esche, e C. Chassapis. “Content-Rich Interactive Online Laboratory Systems”. *Computer Applications in Engineering Education*, 17(1):61–79, 2009.
- [9] I. Calvo, M. Marcos, D. Orive, e I. Sarachaga. “Building Complex Remote Learning Laboratories”. *Computer Applications in Engineering Education*, 18(1):53–66, 2010.
- [10] J. García-Zubía, I. Angulo, U. Hernandez, M. Castro, E. Sancristobal, P. Orduña, J. Irurzun, e J.R. de Garibay. “Easily Integrable platform for the deployment of a Remote Laboratory for microcontrollers”. In *Education Engineering (EDUCON), 2010 IEEE*, páginas 327 –334, 2010.
- [11] National Instruments. “Labview - Laboratory Virtual Instrument Engineering Workbench”, Set. 2008. <http://www.ni.com/labview>.
- [12] S. Lerman e J. del Alamo. “iLab: Remote Online Laboratories”, Dez. 2009. <http://icampus.mit.edu/projects/>.

- [13] P.R.S.L. Coelho, R.F. Sassi, E. Cardozo, E.G. Guimaraes, L.F. Faina, A.Z. Lima, e R.P. Pinto. “A Web Lab for Mobile Robotics Education”. In *Proc. IEEE International Conference on Robotics and Automation*, páginas 1381–1386, 2007.
- [14] Sun Microsystems. “Lesson: Java Web Start”. Technical report, 2008. <http://java.sun.com>.
- [15] D. V. Souza, P. E. Nogueira, P. R. S. L. Coelho, e L. F. Faina. “Uma Arquitetura com Suporte a Módulos Dinâmicos para WebLab no Domínio de Redes de Computadores”. *XXXVI Conferência Latinoamericana de Informática*, 2010. Assunção - Paraguai. CLEI2010.
- [16] P. Orduña, J. García-Zubía, J. Irurzun, E. Sancristobal, S. Martín, M. Castro, D. López-de Ipiña, U. Hernáidez, I. Angulo, e J. M. Gonzáles. “Designing Experiment Agnostic Remote Laboratorie”. *Remote Engineering e Virtual Instrumentation*, 2009.
- [17] Paulo Rodolfo Silva Leite Coelho. “Uma Arquitetura Orientada a Serviços para Laboratórios de Acesso Remoto”. Dissertação de Mestrado, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, Dezembro 2006.
- [18] OSGi Alliance. “OSGi Service Platform Core Specification”. Technical report, Release 4, Version 4.2, Mar. 2010. <http://www.osgi.org>.
- [19] Sun Microsystems. “Java 2 Platform Security Architecture”. Technical report, Dez. 2009. <http://www.java.sun.com>.
- [20] Roy Thomas Fielding. “*Architectural styles and the design of network-based software architectures*”. Tese de Doutorado, 2000. Chair-Taylor, Richard N.
- [21] Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides. “*Design patterns: elements of reusable object-oriented software*”. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [22] Apache Software Foundation. “Apache Felix”, Julho 2009. <http://felix.apache.org>.
- [23] Eclipse Foundation. “Eclipse Equinox”, Junho 2009. <http://www.eclipse.org>.
- [24] Makewave. “Knopflerfish - Open Source OSGi”, Junho 2010. <http://www.knopflerfish.org/>.
- [25] Apache Software Foundation. “Apache Felix File Install”, Julho 2009. <http://felix.apache.org>.
- [26] Apache Software Foundation. “Configuration Admin Service”, Julho 2009. <http://felix.apache.org>.
- [27] Open Participation Software for Java. “Pax Logging User Guide”. Technical report, Ago. 2009. <http://wiki.ops4j.org>.