Capítulo 3

Estado da Arte de Sistemas de Controle e Monitoramento de Infraestruturas para Experimentação de Redes de Comunicação

Cesar Augusto Cavalheiro Marcondes (UFSCar), Joberto Martins (UNIFACS), José Augusto Suruagy Monteiro (UFPE), Kleber Vieira Cardoso (UFG), Antônio Jorge G. Abelém (UFPA), Vagner Nascimento (UFPA), Iara Machado (RNP), Tereza C.M.B. Carvalho (USP), Charles C. Miers (USP/UDESC), Marcos Salvador (CPqD) e Christian Esteve Rothenberg (CPqD)

Abstract

In the last years, the research in communication networks has focused on innovations, following two main approaches: clean-slate and the incremental deployment as possible proposals for the Future Internet. These new ideas need strict investigation in large scale by employing a large amount of available tools such as simulators, emulators and experimental infrastructures. In the context of infrastructures, complex architectures of control and monitoring have been developed, allowing inexperienced experimenters to test new ideas with minimal effort. The control and monitoring frameworks allow registering resources, registering the experimenter, controlling the access to infrastructure's services, creating slices and recording of resource utilization, managing experiment cycle, monitoring results, storing the results, among others actions. The aim of this short course is to survey the state of the art of these systems, making an introduction to them, providing details about their requirements, commenting implementations and deployments, and presenting some cake recipes tested in experimentation infrastructures around the world. We also discuss the efforts of the federation control systems infrastructure for experimentation in order to have a larger and heterogeneous infrastructure to conduct experiments. The course has a theoretical approach, detailing concepts and services of the CMFs, possibly with some detailing about a specific CMF or federation solution such as PlanetLab, ProtoGENI, OFELIA, OMF and SFA 2. We will show some videos and practical examples about the use of such technology in order to call the attention of Brazilian 'experimenters' in networks and distributed systems on the ease of conducting experiments with these frameworks.

Resumo

Nos últimos anos, a pesquisa em redes de comunicação tem focado em inovações que seguem duas abordagens principais: clean-slate *e as que têm implantação incremental como possíveis propostas de Internet do futuro. Essas novas ideias necessitam de rigorosa investigação em larga escala, usando uma grande quantidade de ferramentas disponíveis como simuladores, emuladores e infraestruturas para experimentação de redes. Dentro do contexto das infraestruturas, têm sido desenvolvidas complexas arquiteturas de controle e monitoramento que habilitam experimentadores sem experiência a testar novas ideias com o mínimo de esforço. Os arcabouços de controle e monitoramento, chamados CMFs (*Control and Monitoring Frameworks*), permitem registrar os recursos, registrar o experimentador, controlar o acesso aos serviços da infraestrutura, criar e registrar fatias de utilização de recursos, gerenciar o ciclo de experimentos, monitorar os resultados, armazenar os resultados, entre outras. O objetivo desse minicurso é fazer um levantamento teórico do estado da arte destes sistemas, fazendo uma introdução aos mesmos, passando por detalhar desde os seus requisitos, até as implementações, implantações e cobriremos também receitas de bolo de teste, feitas em ambientes de experimentação ao redor do mundo. Também abordaremos os esforços de federação dos sistemas de controle de infraestruturas para experimentação de modo a ter uma infraestrutura maior e heterogênea para a realização de experimentos. O curso tem um enfoque teórico, detalhando conceitos e serviços dos CMFs, eventualmente com algum detalhamento de CMF e/ou esquema de federação específico como o Planetlab, ProtoGENI, OFELIA, OMF e o SFA 2. Mostraremos alguns vídeos e exemplos práticos sobre o uso dessas tecnologia de modo a chamar a atenção de 'experimentadores' em redes e sistemas distribuídos brasileiros sobre a facilidade de conduzir experimentos com esses arcabouços.*

3.1. Introdução a Infraestruturas para Experimentação em Internet do Futuro

Este minicurso tem o objetivo de realizar uma introdução ao tema de Sistemas de Controle e Monitoramento de Infraestruturas para Experimentação de Redes de Comunicação, ou em inglês, os *Control and Monitoring Frameworks (CMFs)*. Estes sistemas, normalmente, criado integrados a suas próprias redes de experimentação, as chamadas *testbeds* permitem que pesquisadores possam fazer estudos de larga escala em redes e sistemas distribuídos. Exemplo dessas *testbeds* vão desde a GENI nos EUA (*Global Environment for Network Innovations*), FIRE na Europa (*Future Internet Research & Experimentation*), OFELIA (*OpenFlow in Europe: Linking Infrastructure and Applications*) testbed específica para redes baseadas em OpenFlow e demais iniciativas e projetos relacionados, como a futura *testbed* FIBRE (*Future Internet testbeds/experimentation between BRazil and Europe*) da qual participam os autores deste minicurso.

Atualmente, essas *testbeds* e CMFs têm despertado grande interesse na comunidade, e têm gerado esforço considerável em uma melhor especificação e em uma análise minuciosa para tornar tais sistemas mais genéricos e mais fáceis para uso e suportar a federação de seus recursos. O objetivo é ambicioso, permitindo que recursos independentes e distribuídos na rede formem um grande conjunto de recursos para testes de novas ideias em sistemas distribuídos e redes.

De modo a contextualizar os leitores sobre as Infraestruturas para Experimentação em Internet do Futuro, nós organizamos este texto para cobrir quatro áreas principais: os **Requisitos** tanto do ponto de vista de usuários (Seção 3.2), quanto de programadores que desenvolvem a parte de controle (Seção 3.3) e a parte monitoramento (Seção 3.4) dos CMFs. Em seguida, descrevemos resumidamente as **Testbeds** e respectivos CMFs com maior visibilidade internacional, mostrando um pequeno histórico, a arquitetura e em alguns casos, exemplos de uso e teste de cada uma delas (Seção 3.5). A seguir, apresentamos uma discussão sobre o esforço de padronização da **Federação** de *testbeds* (SFA) [Peterson et al. 2010] (Seção 3.6). E também apresentamos dois **Estudos de Caso** OFELIA CF (Seção 3.7) e OMF CF (Seção 3.8) detalhando como pesquisadores brasileiros podem instalar, configurar, gerenciar e utilizar *testbeds* usando recursos próprios de seus grupos de pesquisa para inovar, validar e gerar resultados significativos para a comunidade de redes e sistemas distribuídos. Ao final, encerramos o minicurso com algumas conclusões (Seção 3.9).

3.2. Requisitos Gerais das Infraestruturas do Ponto de Vista do Experimentador

Os Sistemas de Controle e Monitoramento de Infraestruturas formam um interessante ecossistema de ferramentas de suporte a experimentação de rede provendo um *workflow* completo para a pesquisa aplicada em redes de computadores e sistemas distribuídos. Estes aspectos têm que ser documentados no que se refere ao ambiente de aplicação. Os requisitos do usuário são a base para o bom desenvolvimento de um projeto.

A documentação deve contemplar todas as informações advindas do usuário, bem como sua visão de um ambiente de experimentação, como ele interagiria com o mesmo e quais funcionalidades seriam necessárias para planejar, implantar e executar seu experimento, entre outros aspectos envolvidos na infraestrutura.

Para garantir a qualidade dos requisitos é necessário um formalismo que permita que cada requisito seja identificado de forma única, desta forma uma referência única será possível a partir dos documentos gerados nesta fase com o conhecimento do experimentador.

Abordaremos o exemplo genérico que atende a maioria dos requisitos funcionais de experimentadores, o *workflow* GENI, que cobre 3 fases principais: **Planejamento de Experimentos, Implantação dos Experimentos e Execução dos Experimentos**. Em particular, descreveremos o que é feito em cada fase, como por exemplo, detecção de recursos disponíveis, obtenção de credenciais, instalação de software de apoio, controle de experimentos, instrumentação de código e medições ativas e passivas até a coleta de resultados.

3.2.1. Planejamento de Experimentos

É a fase onde o usuário planeja como o experimento será conduzido no ambiente. Ele irá determinar os recursos e ferramentas necessários para programar o experimento, os serviços de instrumentação disponíveis e suas configurações. O Planejamento de Experimentos é constituído por quatro etapas: **Apresentação de Credenciais, Descoberta de Recursos, Descoberta de Ferramentas e Desenvolvimento de Versão**.

Nesta primeira etapa de **Apresentação de Credenciais**, um experimento é criado contendo a alocação de recursos, e com esta a necessidade de certos privilégios, nomes e identificadores únicos. Por sua vez, experimentos podem ser associados a múltiplos experimentadores e estes também podem ter diferentes privilégios, tais como, a capacidade de adicionar novos experimentadores e atribuir-lhes privilégios, ou a capacidade de adicionar ou remover recursos para o experimento e a capacidade de visualização dos dados coletados. Finalmente, experimentadores com privilégios diferentes podem ser adicionados ou removidos a qualquer momento durante o ciclo de vida de um experimento.

Na fase de **Descoberta de recursos** o experimentador procura os recursos necessários para executar um experimento. Para permitir escalabilidade, a descoberta de recursos pode possuir várias fases, por exemplo, receber informações sobre os recursos e, em seguida, ter a capacidade de refinar consultas para obter informações mais detalhadas sobre os recursos específicos ou conjuntos dos mesmos. Além disto, a descoberta de recursos é um processo em constante execução sobre os novos recursos adicionados a um experimento.

Em linhas gerais, a descoberta de recursos pode acontecer das seguintes formas:

- **Automaticamente**: O projeto pode apoiar uma ou mais linguagens de especificação de experimento. Estas linguagens serão usadas para descrever os atributos de um experimento, incluindo os recursos necessários, configuração e dados a serem coletados durante a execução do mesmo.
- **Manualmente**: permite ao experimentador escrever um *script* ou programa que consulta recursos em outros ambientes.
- **Interface Gráfica**: A partir de uma interface interativa os experimentadores devem ser capazes de ver os recursos, como em um ambiente *shell* ou um ambiente gráfico intuitivo e de fácil utilização.

No caso do GENI, os experimentadores podem alocar recursos pelo navegador, se estiverem autorizados para isso. Uma vez que os recursos necessários para executar o experimento sejam descobertos, o experimentador terá que aprender sobre as ferramentas necessárias para programar e configurar esses recursos. Isso acontece na etapa de **Descoberta de Ferramentas**. Nesta etapa, experimentadores podem aprender sobre as restrições a serem contabilizadas para programar e utilizar estes recursos. Possibilita também 'apontar' para emuladores de recursos especializados que podem ser usados para desenvolver e validar software antes que ele seja implantado.

Finalmente, na etapa de **Desenvolvimento de Versão**, o experimentador desenvolve os softwares e configura os arquivos que irão eventualmente ser carregados nos recursos descobertos para o experimento. No caso do GENI, estes softwares podem ser testados em um laboratório utilizando emuladores de recursos especializados ou no próprio ambiente GENI.

O planejamento do experimento, contém uma serie de definições de serviços e ferramentas, que incluem:

- O fornecimento de uma ferramenta que possa interpretar uma especificação de experimento e consultar ambientes apropriados para a descoberta de recursos;
- Uma API que possa ser usada para consultar os recursos. Esta API torna mas fácil a chamada de bibliotecas internas;

- Uma ferramenta gráfica para acessar os recursos. Uma interface ideal deveria utilizar técnicas familiares usando um navegador para procurar recursos em outros domínios, tais como busca de arquivos em sistemas de arquivo ou mesmo livros em um catálogo on-line.
- Além das informações sobre os recursos de uma maneira formal, por exemplo, usando o padrão Rspec, que descreve os recursos solicitados. Informações que podem ser necessárias. O ambiente também deve fornecer ferramentas para extrair essas informações e torná-las disponiveis para o experimentador.

3.2.2. Implantação dos Experimentos

A implantação de um experimento é composta de quatro etapas: **Apresentação de Credenciais, Alocação de Recursos, Instalação de Software/Hardware e Verificação da Implantação**. A fase de Apresentação de Credenciais autentica o usuário e verifica se ele realmente tem permissão para alocar recursos e usá-los para a experimentação.

O experimentador então aloca recursos para o experimento. Alocações de recursos garantem que os recursos solicitados serão disponibilizados para o experimento nos horários especificados. O processo de alocação de recursos é semelhante à navegação pelos recursos e consiste em três fases: Automaticamente a partir de uma especificação do experimento, de forma pragmática ou usando uma interface interativa. Por fim é definido um tempo de início e a duração da alocação dos recursos.

Na instalação de software/hardware o experimentador pode então configurar os recursos obtidos pela: Instalação dos softwares, instalação dos hardwares, configuração dos componentes, definição da(s) topologia(s), habilitar as ligações com a Internet, conectar-se a um experimento em execução (composição de experimento).

Finalmente, o experimentador pode verificar a implantação: Verificando se os recursos necessários foram obtidos e programados com as versões corretas de software e se os fluxos de tráfego são os esperados. A verificação de uma implantação que usa recursos ligados intermitantemente pode ser um desafio.

A implantação dos experimentos consiste em uma série de definições de serviços e ferramentas que incluem o fornecimento:

- De uma lista de itens a serem considerados antes de implantar um experimento.
- Ferramenta que possa interpretar uma especificação de experimento e alocar recursos em ambientes apropriados para recursos.
- Ferramentas para coordenar os horários para a composição de experimentos de grande porte.
- Ferramenta de software em experimentos distribuídos, incentivando a utilização de software dentro de um grande número de componentes, instalação desses software, controle de versão.
- Mecanismo que permita que um experimento possa se conectar a outro experimento já em execução.
- Ferramentas e serviços para validar uma configuração de experimento: recursos alocados, versão de software instalado e a configuração de caminhos para comunicação.

- Um experimentador deve ser capaz de salvar uma lista de recursos alocados nesta fase e, em um momento futuro, solicitar que estes mesmos recursos sejam disponibilizados para o experimento. Isto irá permitir que um experimentador possa retornar a uma configuração que foi válida.
- Ferramentas e serviços para ajudar o experimentador a gerenciar os recursos alocados. Por exemplo, um serviço que notifique o experimentador quando um recurso alocado por ele está prestes a ter seu tempo expirado.

3.2.3. Execução dos Experimentos

A execução dos experimentos é dividida em sete etapas: **Controle do Experimento, Coleta e Análise dos dados, Análise de Uso dos Recursos, Injeção de Falha, Manutenção Operacional e Tratamento de Exceção** durante o ciclo de vida de um experimento. Uma vez que o experimento estiver configurado, o experimentador o controla pelas ações de iniciar, pausar, retomar, reiniciar e finalizar. Os experimentos podem ser controlados em diferentes níveis de granularidade: completo, no nível de recurso ou mesmo através do agrupamento de recursos. Adicionalmente, diferentes experimentadores podem permitir diferentes níveis de controle sobre os experimentos.

Outra atividades relacionada ao controle do experimento, inclui o crescimento ou diminuição do experimento, adicionando ou removendo recursos, ajustando o nível de instrumentação, ligando ou desligando o fluxo de/para a Internet, e conectar ou desconectar de outros experimentos em curso.

O experimentador pode ligar e desligar os diferentes níveis de instrumentação. A medição do desempenho de software distribuído e a utilização dos recursos utilizados são alguns exemplos. Dados de instrumentação podem ser visualizados em tempo real ou salvos para análise futura.

O experimentador pode especificar ações automáticas a serem tomadas quando partes do experimento falhar, se os recursos a serem utilizado tornarem-se indisponíveis ou recursos adicionais se tornarem disponíveis. O experimentador também pode injetar falhas no sistema. Estas falhas podem gerar falhas de recursos, uma redução na quantidade de recursos disponíveis, duplicação de mensagens, etc.

O experimentador também pode monitorar e manter experimentos do ponto de vista operacional. Eles podem usar ferramentas para solucionar problemas de configuração, se as coisas não saem como planejado, por exemplo, ferramentas para visualizar *logs* do sistema e diagnosticar problemas estarão disponíveis. Eles também podem dispor de um 'help-desk' para dar suporte na resolução de problemas.

A execução dos experimentos conta também com uma série de definições de serviços e ferramentas que incluem:

- Fornecer ferramentas para controlar um experimento de programação ou usar uma ferramenta interativa. Um experimentador ou um grupo de experimentadores deve ser capaz de monitorar e controlar os experimentos de vários locais. Idealmente, devem ser capazes de controlar um experimento a partir de qualquer computador usando uma interface web.
- Fornecer ferramentas e bibliotecas para que os experimentadores possam implementar mecanismos padrão para sistemas distribuídos tais como *barriers*, onde di-

ferentes partes do programa distribuído não podem ter progresso até que todas as partes tenham atingido certo ponto;

- Boas ferramentas de instrumentação são críticas para o sucesso do projeto, por causa da importância da medição e análise de experimentos científicos. As ferramentas devem permitir que o experimentador varie dinamicamente o nível de instrumentação, controlando o fluxo de medições para ferramentas de análise e arquivando esses dados. O sistema de instrumentação deve ter um impacto mínimo ou nenhum para o desempenho do experimento. São necessários mecanismos para controlar o acesso aos dados coletados, já que alguns destes experimentos podem coletar dados ou tráfego que estão sujeitos a políticas de privacidade. Nestes casos, o projeto deve:
  - Fornecer ferramentas para visualizar os dados coletados.
  - Proporcionar ferramentas para injetar uma variedade de diferentes tipos de falhas de uma maneira controlada.
- Os usuários devem ser capazes de adicionar as suas próprias ferramentas de injeção de falhas em um *framework* de controle.
- Fornecer ferramentas e serviços para depurar problemas operacionais em um experimento causados pela incapacidade de atingir ou obter certos recursos ou comportamento inesperado. Também pode haver um *help-desk* para ajudar experimentadores a solucionar problemas operacionais.
- Fornecer mecanismos para que os experimentadores especifiquem ações a serem tomadas quando determinado padrão ou exceção ocorram durante a execução do experimento.

3.3. Descrição das Principais Funcionalidades da Parte de Controle dos CMFs

A questão crucial de uma *testbed* é o controle dos recursos e como coordenar os experimentos. Para controlar, é preciso autenticar, listar quais os recursos disponíveis e deixar disponível via API, maneiras automáticas de configuração da testbed.

Nesta seção apresentaremos os requisitos e **principais funcionalidades de controle** para os arcabouços de controle e monitoramento. Para tanto, faremos uma revisão de especificações descritas nas referências de controle do GENI [GENI 2009] que serve de base e modelo para todos os CMFs.

A arquitetura de controle do GENI pode ser resumida pela Figura 3.1. No topo, está a infraestrutura chamada *Clearinghouse* que inclui todos os cadastros dos componentes e atores do sistema GENI que permitem a autenticação dos mesmos. Entre eles, pode-se notar: o registro dos pesquisadores (chamados de *principals*), registro de componentes, registro de fatias ou fatias de experimentação (chamadas de *slices*) e, opcionalmente registros relacionados à 'auditoria' de recursos e repositório de software.

Na parte inferior da Figura 3.1, existem diversas entidades independentes (chamadas *Aggregates*) que possuem recursos computacionais para serem agregados a um experimento e que serão controlados pelo plano de controle, disposto em formato de barramento (*Control Plane*). Cada agregado deve possuir um gerenciador do mesmo (*Aggregate Manager*), e opcionalmente outros gerenciadores específicos para componentes (*Component*

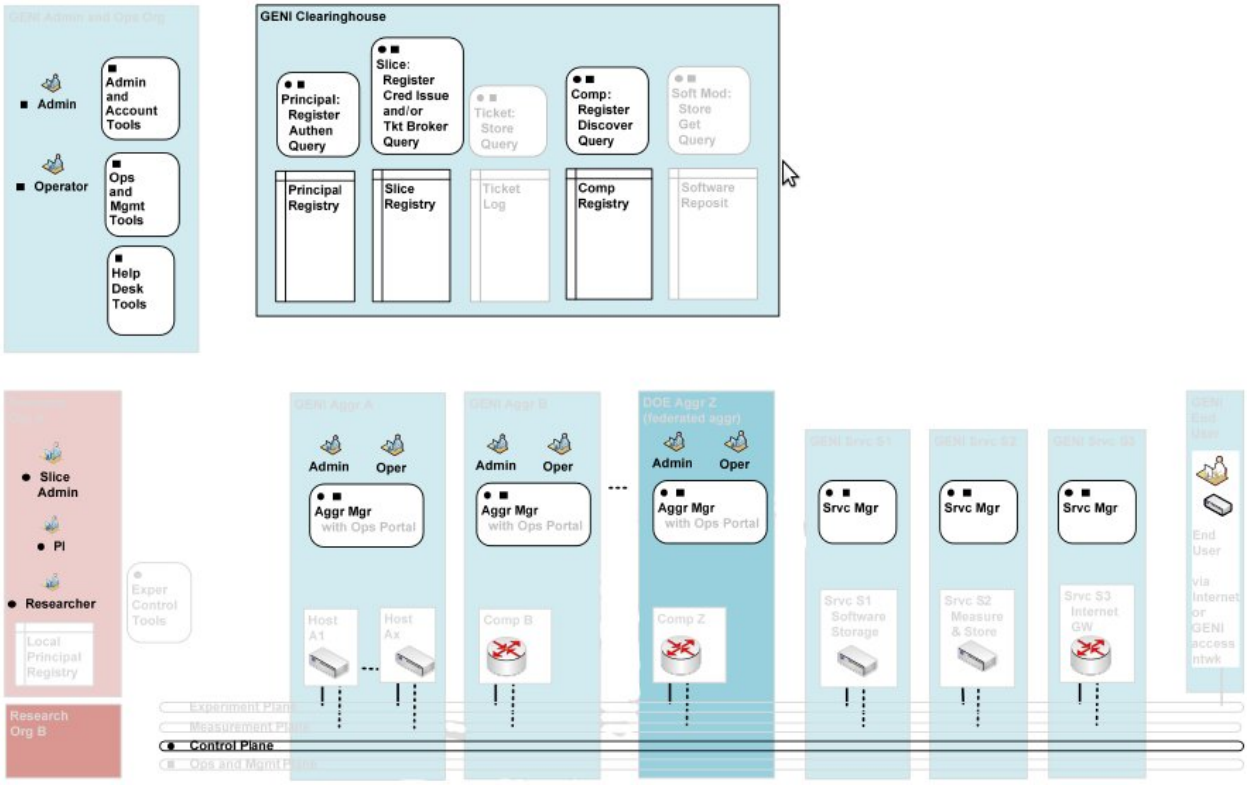


Figura 3.1. Visão Geral dos Serviços de Controle da Arquitetura GENI

Manager). Outra opção avançada é a inclusão de serviços de 'corretores' (*brokers*) de recursos, que tipicamente podem funcionar como gerenciadores de conjuntos de agregados, buscando a melhor disposição de recursos para seus clientes.

Finalmente, no canto esquerdo da Figura 3.1, existem as entidades associadas com o pesquisador (*Principal* no GENI) que estará utilizando, administrando e gerenciando recursos para seu experimento. Para isso, uma das alternativas é utilizar um navegador que permita acessar um portal para facilitar a interação com as ferramentas e comandos.

Baseado nas várias entidades descritas, é possível definir precisamente o arcabouço da parte de controle como o sendo responsável por:

- Realizar o interfaceamento entre todas as entidades (*Clearinghouse*, *Aggregates*, *Principal*);
- Definir os tipos de mensagens do arcabouço, incluindo a sintaxe e funcionalidade de protocolos básicos de controle;
- Definir o fluxo de mensagens dentro do arcabouço para realizar cenários de experimentação.

Em seguida descreveremos com maiores detalhes os diferentes requisitos de controle.

3.3.1. Requisitos de Controle de Acesso e Credenciais

Os *Principals* são pessoas que estão trabalhando a partir de um navegador, gerenciando, controlando e utilizando os recursos do GENI. Estas precisam de um identificador global, de uma etapa de registro que pode definir acesso privilegiado a recursos. E finalmente,

após o registro, a obtenção de uma chave criptográfica, comumente chamada de *credencials*, para acesso. Os tipos de usuários não abrangem somente os pesquisadores usuários mas podem também definir: operadores da testbed, gerentes de projetos, administradores de slices, PIs e pesquisadores.

3.3.2. Requisitos de Agregados e Componentes

Os agregados e seus componentes são os blocos principal do sistema GENI. No agregado são encapsulados os recursos, como máquinas virtuais, recursos lógicos (portas) e sintéticos (virtual paths) chamados de componentes. Opcionalmente, tais agregados podem revelar a estrutura interna de seus componentes. Do ponto de vista de controle, existe um gerenciador chamado Aggregate Manager (AM) que exporta uma interface bem-definida para o sistema GENI. Cada Aggregate deve ter uma identificação global e um registro permitindo sua posterior alocação.

3.3.3. Requisitos de Fatias

Para conduzir experimentos é preciso alocar recursos, dentro no controle do GENI, é usada uma abstração chamada de Fatia (*Slice*), que consiste em um conjunto interconectado de recursos reservados. Baseando-se ainda na Figura 3.1, a Fatia é um alocação horizontal nos agregados provendo a interconexão de seus componentes e provendo o plano de controle isolado ao experimentador.

Tais recursos reservados (chamados de *Slivers*) podem estar localizados em diferentes e heterogêneos agregados. O sistema de controle de referência do GENI deve prover um acesso remoto que permita aos pesquisadores descobrir, reservar, configurar, programar, debugar, operar, gerenciar e desligar recursos dentro de uma fatia para completar um experimento.

As Fatias devem ter um tempo de vida considerável, e portanto poderiam ser utilizadas em vários experimentos em sequência, por exemplo. Elas precisam ter uma identificação global e registro autenticado da alocação de Fatias. A partir das Fatias, cabe ao controle permitir funções de configuração de experimentos, a partir de serviços como descoberta da topologia e descoberta de recursos. Os recursos, por sua vez, deveriam, para melhor aproveitamento e multiplexação, suportar virtualização ou outras formar de compartilhamento justo. Para o uso de recursos deve haver a autorização e alocação dos mesmos a partir dos agregados, tudo isso pautado por políticas de uso a serem estabelecidas. Tais políticas de uso poderão ser baseadas em uma variedade de parâmetros como: relacões de confiança e de contrato com os atores do sistema, o status acadêmico do pesquisador, a presença de um sistema econômico baseado em *moeda virtual*, e é claro, na disponibilidade momentânea do recurso.

3.4. Monitoramento dos CMFs — Requisitos e Funcionalidades

Nesta seção apresentaremos os requisitos e principais funcionalidades de monitoramento para os arcabouços de controle e monitoramento. Nos basearemos na Arquitetura de Instrumentação e Monitoração do GENI (GENI I&M) [GENI 2010, Mussman 2012] por ser uma das mais abrangentes e, assim sendo, poder servir como modelo de referência apesar de ainda se encontrar em estágios iniciais de seu desenvolvimento.

3.4.1. Requisitos de Monitoração

Redes experimentais necessitam basicamente de dois tipos de monitoração: monitoração dos experimentos executados que serão úteis para a análise e reprodução dos experimentos, e a monitoração da infraestrutura por parte dos operadores da mesma de modo que possam acompanhar o estado de seus diversos componentes, identificar eventuais falhas, garantindo assim um ambiente confiável para a realização dos experimentos. Naturalmente, os experimentadores podem também ter acesso aos dados da monitoração da infraestrutura para auxiliar-lhes na execução de seus experimentos assim como para garantir a reprodutibilidade dos mesmos.

Os objetivos gerais a serem alcançados pelo sistema de instrumentação e monitoração do GENI (GIMS — *GENI I&M System*) são [GENI 2010]:

- Fornecer funcionalidades abrangentes de coleta, análise e arquivo de dados necessários para a execução com sucesso dos experimentos e do funcionamento da infraestrutura;
- Remover do pesquisador a necessidade de ser um especialista em monitoração do experimento e da infraestrutura de modo que ele possa se concentrar no seu experimento propriamente dito;
- Realizar as medições com alta precisão e acurácia de forma ubíqua, extensível, com alta disponibilidade, segura e integrada sem impactar negativamente os experimentos que estão sendo executados;
- Prover informações detalhadas sobre o desempenho do sistema e dos recursos da rede ao nível de cada etapa (*hop*), enlace (*link*), caminho (*path*) e fatia (*slice*) em termos de disponibilidade, integridade e diagnóstico de problemas percebidos e/ou que causem impedimentos reais;
- Permitir e facilitar que grupos de usuários possam acessar e controlar funções que envolvam interações entre subserviços de I&M, incluindo recursos tais como derivações físicas na rede, sensores de tempo, pontos de medição baseados em software ou em hardware, MIBs de comutadores ou roteadores assim como arquivos de dados de medição;
- Prover uma transparência do desempenho do estado dos componentes individuais do subserviço de I&M e suas interfaces com outros subserviços de modo a garantir a corretude das medições realizadas;
- Prover mecanismos que tratem segurança, privacidade e controle de acessos aos arquivos de dados de medição para permitir o acesso apenas a usuários autorizados, e também para fornecer visões diferentes dos dados baseadas nos privilégios da autorização.

3.4.2. Grupos de Usuários e Casos de Uso

Do ponto de vista de monitoramento, no GENI I&M, foram identificados os seguintes grupos de usuários: pesquisadores experimentadores, usuários de experimentos, operado-

res centrais, provedores e operadores de agregados, provedores e operadores de arquivos, pesquisadores que fazem uso dos dados armazenados de monitoração.

Os **pesquisadores experimentadores** são aqueles que executam os experimentos para a futura Internet utilizando fatias em diversos componentes da rede experimental. Estes têm interesse em realizar testes que possam comprovar que os recursos solicitados estejam efetivamente disponíveis para os seus experimentos, com as características solicitadas. Eles também têm interesse de acompanhar o andamento de seus experimentos em tempo de execução de modo a observar qualquer problema. Podem também consultar o estado de recursos de suas fatias para identificar eventuais problemas na infraestrutura que podem estar causando inconsistências nos dados de seus experimentos. Têm interesse também em dados arquivados de medições de desempenho dos recursos de suas fatias de modo que possam analisá-los mesmo após o tempo de vida das mesmas. Desejam ativar pontos de medição ativa e/ou passiva e interfaces de modo a monitorar os dados coletados utilizando ferramentas genéricas ou específicas a algum fabricante. Podem solicitar que sejam notificados em caso de alguma anomalia detectada a partir de limiares e ferramentas especificadas pelos mesmos. Podem ainda solicitar que as medições de desempenho de suas fatias sejam compartilhadas em tempo de execução ou após os mesmos com os usuários do experimento ou com outros pesquisadores com a possibilidade de estabelecer diferentes níveis de permissão.

Os **usuários de experimentos**, como o próprio nome diz, são usuários que participam de algum experimento para utilizar seus recursos, aplicações ou serviços. Estes, apesar de simples usuários do experimento, devem ter acesso a medições de desempenho dos recursos que lhes foram alocados e/ou da aplicação ou serviço que estão utilizando. Com estes dados, têm condições de reportar ao pesquisador responsável pelo experimento sobre eventuais problemas de desempenho.

Os **operadores centrais** monitoram os recursos e processos da rede experimental de uma forma consistente e confiável para toda a infraestrutura formada por diversos agregados e/ou componentes pertencentes a diferentes organizações. Com esta finalidade eles devem monitorar as diversas fatias para identificar alguma que possa comprometer os demais experimentos e removê-la. Devem também ser capazes de identificar problemas em recursos em diferentes agregados pertencentes a um dado experimento a partir de solicitação direta dos experimentadores. Devem ser capazes de configurar diversas monitorações passivas e ativas, disponibilizar estes dados em tempo real, assim como arquivá-los. Devem ainda ser capazes de analisar eventuais problemas na infraestrutura que possam ter comprometido algum experimento.

Os **provedores e operadores de agregados** são aqueles que disponibilizam componentes computacionais ou de rede para serem utilizados nos experimentos, permitindo aos usuários verificar o estado e disponibilidade destes componentes. Estes têm necessidades semelhantes às dos operadores centrais, mas limitadas aos recursos que disponibilizaram para a infraestrutura.

Os **provedores e operadores de arquivos** são responsáveis pela catalogação dos conjuntos de dados coletados em um repositório fornecendo ferramentas para o compartilhamento, anotação, busca e referência a estes conjuntos de dados de monitoração. Necessitam de informações dos demais usuários para poder catalogar mais eficientemente

os conjuntos de dados. Devem possuir mecanismos de autenticação que permitam controlar o acesso dos diversos tipos de usuários aos dados armazenados. É interessante ainda que possam receber novas ferramentas que facilitem a análise e visualização dos dados armazenados.

Finalmente, os **pesquisadores que fazem uso dos dados armazenados de monitoração** são usuários que nem realizaram nem participaram dos experimentos mas que desejam utilizar os dados fornecidos pelos provedores de arquivos para analisar os dados, testar hipóteses, etc. Estes desejam realizar buscas nos dados, assim como compartilhar, anotar e citar os dados utilizados para os seus estudos.

3.4.3. Serviços de Instrumentação e Monitoração (I&M) para o GENI

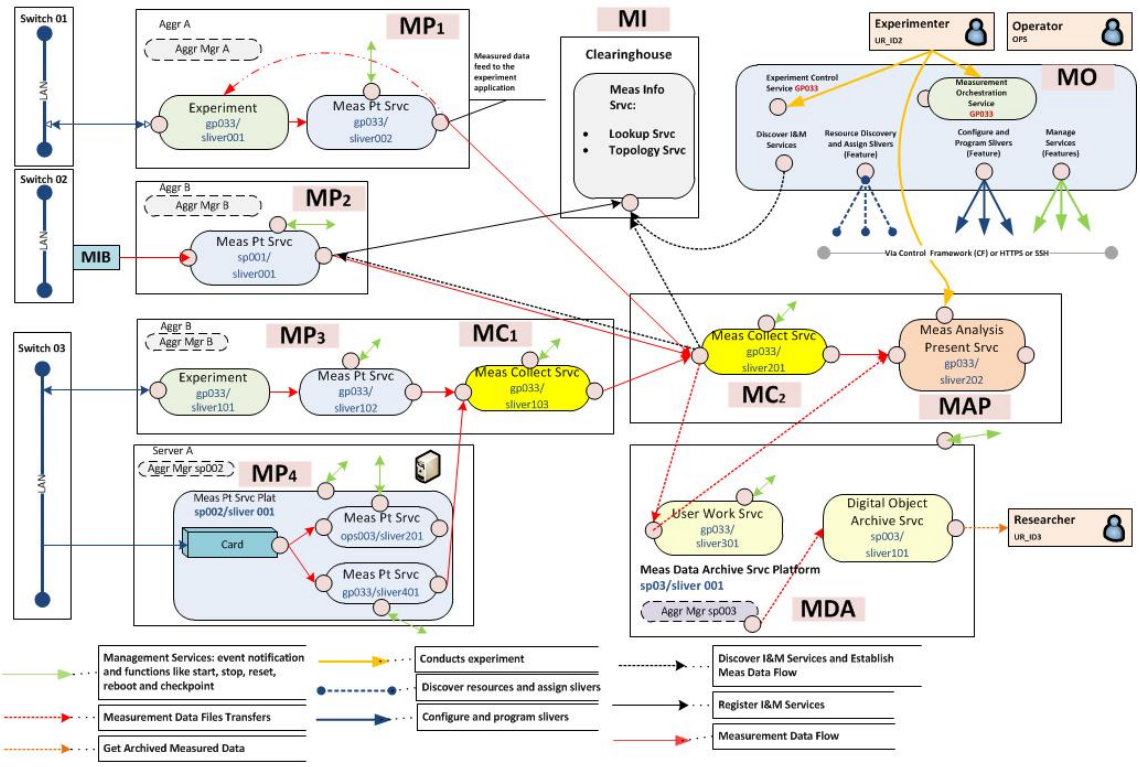


Figura 3.2. Visão Geral dos Serviços da Arquitetura GENI I&M

Na Arquitetura de Instrumentação e Monitoração do GENI [GENI 2010] foram definidos os seguintes serviços, que serão detalhados a seguir: **Serviço de Orquestração de Medições, Serviço de Ponto de Medição, Serviço de Informações de Medição, Serviço de Coleta de Medições, Serviço de Análise e Apresentação de Medições e Serviço de Arquivo de Dados de Medição**. Uma visão geral destes serviços e de seus relacionamentos é apresentada na Figura 3.2.

O **serviço de Orquestração de Medições** (MO — *Measurement Orchestration*) faz parte de um serviço de Controle do Experimento e tem como finalidade orquestrar as funções de outros serviços de I&M. Ou seja, da mesma forma que um experimentador utiliza serviços ou ferramentas de controle do experimento para reunir recursos numa determinada fatia, configurá-los e/ou programá-los e executar seu experimento, ele utiliza o MO para gerenciar os seus serviços de I&M.

O **serviço de Ponto de Medição** (MP — *Measurement Point*) provê a funcionalidade de monitoração propriamente dita interagindo diretamente com a infraestrutura da rede experimental, ou com os recursos de uma determinada fatia, exportando os dados através de esquemas padronizados.

Dentre os MPs podemos citar sensores de enlace, de nós e de tempo. Os sensores de enlace obtêm sinais básicos dos enlaces através de alguma derivação. Os sensores de nós são instalados em todos os sistemas e provêem dados básicos de utilização, estado e configuração. Finalmente, os sensores de tempo são instalados em diversas instalações para produzir marcas de tempo de alta resolução.

As aplicações de um dado experimento podem fazer uso dos dados obtidos por um determinado MP para algum tipo de adaptação de seu comportamento. Serviços persistentes, por exemplo, de dados da infraestrutura, podem ser registrados de modo que possam ser usados por operadores e, se permitido, também por experimentadores.

Os serviços de ponto de medição utilizam dados provenientes de diversas ferramentas e/ou interfaces. Por exemplo, podem obter dados de MIBs (*Management Information Bases*), de comutadores ou roteadores ou obter dados através de placas de captura como as DAGs (*Data Acquisition and Generation*)[1].

O **serviço de Informações de Medição** (MI — *Measurement Information*) provê um serviço de registro e descoberta de dados de medição, incluindo informações sobre a topologia da infraestrutura de modo que os dados disponíveis possam ser mapeados na mesma.

O **serviço de Coleta de Medições** (MC — *Measurement Collection*) é um serviço que recebe como entrada dados de medições e efetua algum processamento nos mesmos tais como combinações, transformações ou armazenamento temporário (*cache*). Espera-se que seja desenvolvida uma grande variedade de opções em servidores de propósito geral usando pacotes de software padronizados. Tanto as entradas como as saídas devem usar esquemas definidos.

O **serviço de Análise e Apresentação de Medições** (MAP — *Measurement Analysis and Presentation*) é um sistema que recebe dados de medição de MCs ou diretamente de MPs, analisa os mesmos e produz algum tipo de apresentação. Assim como os MCs, espera-se que seja desenvolvida uma grande variedade de opções em servidores de propósito geral usando pacotes de software padronizados. Tanto as entradas como as saídas devem usar esquemas definidos.

Finalmente, o **serviço de Arquivo de Dados de Medição** (MDA — *Measurement Data Archive*) provê um repositório dos dados de medição tanto para experimentadores como para operadores, assim como um Portal de acesso aos pesquisadores, se tiverem autorização para tal.

Espera-se que sejam disponibilizadas ferramentas que favoreçam o compartilhamento dos dados armazenados nestes repositórios.

[1] `http://dag.cs.waikato.ac.nz/`

3.4.4. Cenários de Uso dos Serviços

Estes serviços podem ser usados, por exemplo, para que: (i) um experimentador utilize dados de medição de sua fatia; (ii) um operador obtenha dados da infraestrutura da rede experimental; ou (iii) um experimentador utilize dados de medição de sua fatia e da infraestrutura da rede.

Nos casos em que experimentadores utilizem dados de medição de sua fatia, os serviços MP, MC e MAP fazem parte da fatia do experimentador e, portanto, são criados, configurados e gerenciados pelos mesmos. O serviço MO faz parte dos servios de Controle do Experimento e é utilizado pelo experimentador para gerenciar os seus serviços de I&M. Os dados de medição estão contidos na fatia e pertencem, portanto, ao experimentador que pode compartilhá-los ou não com outros experimentadores, pesquisadores e/ou operadores.

Um operador poderá criar, através de serviços/ferramentas de Controle de Experimento, uma fatia para monitorar os recursos do operador. Numa infraestrutura federada como a GENI tipicamente teremos MPs sendo executados por diferentes provedores de serviço que podem registrar os seus fluxos de dados de medição no serviço MI, de modo que o operador consultando-o passe a receber estes fluxos que podem ser então analisados e apresentados através de gráficos da infraestrutura da rede. Neste caso os serviços MP, MI e MAP têm diferentes proprietários e fazem parte de diferentes fatias.

Finalmente, caso um experimentador deseje também receber dados da infraestrutura da rede, ele deverá descobrir a disponibilidade destes dados através de uma consulta ao serviço MI e, sendo autorizado para usar o fluxo, configurará um MC para receber os dados tanto de seus MPs como de MPs pertencentes ao provedor de serviço.

3.4.5. Tipos de Serviços

Os serviços de I&M do GENI foram classificados em quatro tipos de acordo com o seu uso: (1) serviço contido em uma fatia; (2) plataforma de serviço comum com múltiplos pedaços de componentes (*slivers*) dedicados a múltiplos experimentos; (3) serviço comum com dados compartilhados fornecidos a múltiplos experimentos; e (4) serviço MDA com um portal para o compartilhamento de dados.

No tipo 1, o serviço está contido dentro de uma única fatia de propriedade de um experimentador, de um provedor de serviço ou de um operador. Em muitos casos podem ser usados serviços de I&M em série dentro de uma fatia. Neste caso, é necessário um processo de costura (*stitching*) para que haja um fluxo de dados de monitoração da saída de um serviço para a entrada do próximo serviço.

No tipo 2, há uma plataforma de serviço comum montada, configurada e gerenciada por um provedor de serviço e múltiplos pedaços que são adquiridos, configurados e gerenciados por vários proprietários de fatias (experimentadores ou operadores).

No tipo 3, há um serviço comum montado, configurado e gerenciado por um provedor de serviço que provê dados de medições para múltiplas fatias (de propriedade de experimentadores ou operadores). O provedor deve determinar como os dados podem ser usados ou compartilhados com experimentadores e/ou operadores.

No tipo 4, o compartilhamento dos dados de monitoração é realizado através do acesso a um portal. O proprietário dos dados decide como ele pode ser compartilhado com outros experimentadores e pesquisadores que não estejam associados à mesma fatia.

3.4.6. Ferramentas de I&M Relacionadas

Conforme mencionamos anteriormente, o GIMS ainda se encontra em fase preliminar de desenvolvimento. De qualquer forma, nesta subseção apresentaremos duas infraestruturas que estão sendo montadas baseadas em ferramentas pré-existentes buscando uma convergência para o GIMS: a GEMINI e GIMI, e uma infraestrutura independente desenvolvida no contexto do projeto OneLab, a TopHat.

O **GEMINI** (*A GENI Measurement and Instrumentation Infrastructure*) [Swany et al. 2011] é um projeto da quarta espiral do GENI que irá estender e integrar as ferramentas produzidas pelos projetos LAMP e INSTOOLS iniciados na segunda espiral aproveitando a complementariedade das mesmas.

O projeto LAMP (*Leveraging and Abstracting Measurements with perfSONAR*) [Swany and Boyd] tem como objetivo aproveitar as ferramentas e serviços desenvolvidos para o perfSONAR[2] (*PERFormance Service Oriented Network monitoring ARchitecture*) [Hanemann et al. 2005] — um ambiente de monitoração de redes multidomínios orientada a serviços. Neste ambiente, ferramentas de monitoração tais como BWCTL[3] e OWAMP[4] são usadas para efetuar medições ativas de largura de banda, atraso e perdas em um sentido entre pontos de medição (MPs) instalados em pontos estratégicos de uma infraestrutura de redes. No cenário original de uso, são realizados testes regulares ou sob demanda para que sejam identificados eventuais problemas de desempenho que possam indicar falhas de conectividade, roteamento, etc. O perfSONAR é um *middleware* que interage com ferramentas de monitoração e dados de medição coletados das mais variadas formas e disponibiliza estes mesmos dados através de uma interface padrão orientada a serviços, utilizando esquemas definidos pelo grupo de trabalho de monitoração de redes (NM-WG — *Network Monitoring Working Group*)[5] do *Open Grid Forum* (OGF)[6].

No projeto LAMP foi criado um sistema de instrumentação e monitoração baseado na versão PS do perfSONAR[7] no qual foi incluída a ferramenta Ganglia[8] para o monitoramento de grades computacionais, no contexto do ambiente de controle ProtoGENI a ser apresentado na Subseção 3.5.2. A intenção dos pesquisadores responsáveis por este projeto era a de colaborar no planejamento e desenvolvimento da arquitetura do GIMS enfatizando um formato extensível de armazenamento e compartilhamento dos dados. De fato, há uma grande semelhança entre os serviços e formatos de dados do perfSONAR e aqueles que estão sendo considerados para o GIMS.

Enquanto que o LAMP teve a sua origem no monitoramento de infraestruturas de

[2] http://www.perfsonar.net/
[3] http://www.internet2.edu/performance/bwctl/
[4] http://www.internet2.edu/performance/owamp/
[5] http://www.ogf.org/gf/group_info/view.php?group=nm-wg
[6] http://www.ogf.org/
[7] http://www.internet2.edu/performance/pS-PS/
[8] http://ganglia.sourceforge.net

redes, o INSTOOLS (*Instrumentation Tools for a GENI Prototype*) [GENI 2012, Duerig et al. 2012,Griffioen et al. 2009] teve a sua origem em instrumentar o ambiente de usuário, inicialmente no Emulab e posteriormente no ProtoGENI, através de uma interface simples e fácil de ser utilizada dentro de uma fatia do GENI.

O INSTOOLS é composto basicamente por pontos de medição (MPs) passiva que são instalados automaticamente em cada recurso da fatia e capturam dados de medição de diversas formas e por controladores de medições (MCs — *Measurement Controllers*) que coletam, armazenam, processam e apresentam os dados através de um portal. O sistema cria pelo menos um MC para cada um dos agregados que são utilizados no experimento (na fatia). Isto é feito para que o MC seja adaptado a cada agregado e possa oferecer recursos de monitoração que sejam específicos a cada um deles.

No GEMINI espera-se utilizar a interface existente do INSTOOLS com os arcabouços de controle, assim como o seu portal e se beneficiará dos padrões e definições de metadados do perfSONAR/LAMP, para facilitar o acesso de outras ferramentas de visualização aos dados coletados [Griffioen]. Espera-se ainda incluir ferramentas de medição ativas disponíveis hoje apenas no LAMP.

Além destes aproveitamentos de cada um dos projetos anteriores está prevista a realização de novos desenvolvimentos tais como [Swany et al. 2011]: o desenvolvimento de um registro global (GGR — *GEMINI Global Registry*) estendendo o UNIS (*Unified Network Information Services*) já disponível no LAMP; implementação da funcionalidade de publicação e assinatura; a ampliação das medições (origens dos dados); integração do OpenFlow [McKeown et al. 2008]; visualização com a ferramenta WorldView[9]; extensão do sistema de arquivamento (está sendo considerado o uso do iRODS[10]); e refinamento e adaptação à arquitetura do GIMS.

O projeto **GIMI** (*Large-scale GENI Instrumentation and Measurement Infrastructure*) [Zink 2011] tem como objetivo prover ferramentas fáceis de usar baseadas na OML (*Orbit Measurement Library*)[11] [Singh et al. 2005] para o ambiente GENI, colaborando com os demais projetos, em particular, o GEMINI.

A OML é um ambiente de software distribuído que permite a coleta de dados em tempo real num ambiente de larga escala. Apesar de ter sido desenvolvido para o ambiente ORBIT, ele pode ser usado em qualquer outro ambiente. A OML possui três componentes principais: o serviço OML (*OML-Service*), a biblioteca de cliente (*OML-Client library*) e o servidor de coleta de dados (*OML-Collection-Server*). O serviço OML gera o código necessário para que aplicações escritas em C/C++ possam interagir com a OML criando também arquivos de execução do experimento para o cliente e servidor baseados na definição do experimento. A biblioteca de cliente é instalada nos nós participantes do experimento e poderá agregar e filtrar os dados recebidos das aplicações participantes do experimento enviando-os ao servidor central. Este contém também um servidor proxy para armazenar os dados caso este esteja temporariamente desconectado do servidor, o que é bem comum em nós móveis. Finalmente, o servidor de coleta de dados coleta os dados provenientes de todos os nós envolvidos no experimento armazenando-os em uma

[9] `http://globalnoc.iu.edu/worldview.html`
[10] `http://www.irods.org`
[11] `http://www.orbit-lab.org/wiki/Documentation/OML`

base mySQL.

Ao ser usado no GINI o cliente OML deve ser instalado em cada nó como MPs e o servidor OML como um MC em cada fatia do experimento. Dado que as fatias são não persistentes, convém exportar os dados do servidor para um servidor persistente no espaço do usuário ou num portal, antes que a fatia expire. Para o armazenamento dos dados, assim como no GIMINI, também está sendo considerado o uso do iRODS. Podem ainda ser associados serviços de análise aos servidores OML, cujos dados também podem vir a ser exportados através de uma interface perfSONAR para compartilhamento de dados utilizando ferramentas do GEMINI.

TopHat é o componente de medições ativas do PlanetLab Europa (PLE) que provê para as aplicações do PL e, em particular, ao MySLICE[12], um serviço de monitoração de topologia durante todo o ciclo de vida de um experimento [Bourgeau et al. 2010]. A partir das medições realizadas, os usuários podem melhor alocar sua fatia, recursos que melhor atendam os requisitos do experimento, incluindo conectividade e atrasos.

O TopHat alimenta de forma transparente o MySlice com informações de diversos sistemas de monitoração tais como a TDMI (*TopHat Dedicated Measurement*), o DIMES[13] — que estuda a estrutura e topologia da Internet, ETOMIC [Csabai et al. 2010] (que provê um serviço semelhante ao obtido com o perfSONAR usando a ferramenta OWAMP), SONoMA[14], e de outros serviços de informação tais como o Team Cymru[15] que provê um serviço de mapeamento de endereço IP para AS (*Autonomous System*).

3.5. Exemplos de Testbeds e respectivos CMFs

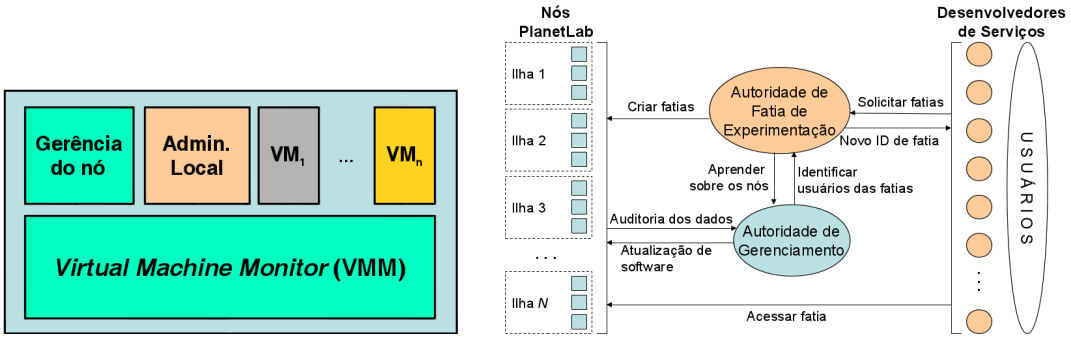
Nesta seção serão resumidas descrições de várias *testbeds* e os CMFs relacionados às mesmas. Cada resumo cobrirá desde histórico, os conceitos básicos, breve descrição sobre a arquitetura, e exemplo de testes.

3.5.1. Planetlab

O PlanetLab[16] é uma plataforma de computação distribuída que foi criada e é mantida por uma comunidade de pesquisa em mais de 405 localidades em mais de 35 países. Os participantes do PlanetLab disponibilizam uma quantidade de nós (normalmente pequena, e no mínimo dois computadores) e em contrapartida podem utilizar os recursos compartilhados disponíveis em toda a plataforma para implantar e avaliar experimentos de rede em uma escala global [Peterson and Roscoe 2006]. Os objetivos que motivaram a criação do PlanetLab [Peterson et al. 2003] foram:

- Fornecer para os pesquisadores uma plataforma de experimentação para serviços de rede em âmbito global;
- Fornecer uma plataforma para serviços de rede inovadores que possam ser implantados e usados por uma comunidade real de usuários; e

[12] `myslice.onelab.eu`
[13] `http://www.netdimes.org/`
[14] `http://complex.elte.hu/sonoma/`
[15] `http://www.team-cymru.org/`
[16] `http://www.planet-lab.org/`



(a) Componentes dos Nós PlanetLab (b) Arquitetura do PlanetLab

Figura 3.3. PlanetLab

- Proporcionar um catalizador da evolução da Internet em uma arquitetura orientada a serviços.

É importante frisar que o PlanetLab foi projetado para executar experimentos de serviço de rede em condições similares ao mundo real, não para fornecer um ambiente controlado de testes. Isso é uma consequência natural da plataforma devido à quantidade de recursos envolvidos e que são controlados por diversas organizações do mundo, sujeitos a diferentes condições. Exemplificando: algumas das máquinas que faziam parte de uma experiência executada ontem podem ter falhado, ficado sem espaço em disco, foram atualizadas ou reconfiguradas; entre muitos outros possíveis problemas, tornando-as indisponíveis para a repetição de um experimento.

3.5.1.1. Arquitetura

O PlanetLab é composto de várias ilhas que estão distribuídas em várias localidades. Os principais elementos do PlanetLab são:

- Sistema operacional dos nós. Fornece isolamento das fatias e audita o comportamento dos nós;
- Central do PlanetLab (*PlanetLab Central* – PLC). Gerencia remotamente os nós. Além disso, realiza o serviço de inicialização para instanciar e controlar as fatias;
- Serviços de infraestrutura. Monitoram as fatias e suas condições operacionais, descobrem os recursos disponíveis, criam/configuram as fatias e realizam a alocação de recursos.

Cada nó PlanetLab (figura 3.3(a)) possui software preparado para interagir com os elementos do PlanetLab, permitindo a sua conexão com as autoridades da plataforma e também concedendo alguns direitos de administração local. As máquinas virtuais devem ser configuradas usando uma imagem padrão disponível na plataforma, a fim de que possam ser gerenciadas e fornecer os dados para as análises dos experimentadores.

A figura 3.3(b) ilustra os principais elementos da arquitetura do PlanetLab:

- Ilhas. Cada ilha pertence a uma instituição que é proprietária dos nós que a compõe.
  - Hospeda um ou mais nós PlanetLab;
  - Escolhe uma Autoridade de Gerenciamento a qual irá se subordinar e aprova uma ou mais Autoridades de Fatia de Experimentação.
- Provedores de Serviços (Desenvolvedores):
  - Desenvolvem e implementam experimentos de serviços de rede;
  - Responsáveis pelo comportamento dos serviços;
- Autoridade de Gerenciamento (*Management Authority* – MA):
  - Responsável por instalar e manter atualizados os softwares nos nós;
  - Cria máquina virtuais (*Virtual Machines* – VM) e monitora o seu comportamento;
- Autoridade de Fatia de Experimentação (*Slice Authority* – SA):
  - Registra os Provedores de Serviço;
  - Cria fatias e as conecta com o provedor adequado/responsável.

3.5.2. ProtoGENI

O ProtoGENI é um arcabouço de controle que é parte integrante do Cluster C [gen 2012] do projeto GENI desenvolvido pelo Flux Research Group[17] da Universidade de Utah. Seu projeto está fortemente baseado no sistema Emulab[18], que foi desenvolvido pela mesma equipe.

O arcabouço de controle do ProtoGENI foi projetado com base na arquitetura SFA – *Slice-based Facility Architecture*, que será discutida na seção 3.6. Sendo assim, o ProtoGENI interage com os vários *testbeds* através da organização destes em federações que interagem com o arcabouço de modo a aprovisionar e fornecer os recursos necessários para realizar as tarefas envolvidas em um experimento. A estrutura do ProtoGENI é formada pelos seguintes elementos (figura 3.4):

- Central de Informações (*Clearinghouse*): inclui o registro de usuários, fatias e Agregados/Componentes. Além de alguns serviços comuns e específicos para registros;
- Autoridade de Fatias de Experimentação (*Slice Authority* – SA);
- Gerenciador de Agregados (*Aggregate Manager* – AM); e
- Usuários (incluindo os experimentadores).

Todas as autoridades do ProtoGENI possuem uma interface XMLRPC[19] com os clientes através de HTTP e SSL. Cada serviço possui uma única URL na Central de Informações e todas as requisições são realizadas através deste endereço, sendo que tanto o cliente como o servidor devem se autenticar durante o estabelecimento da conexão. Além disso, cada membro da federação possui um certificado X.509 auto assinado e todos os certificados são distribuídos pela Central de Informações. O código do ProtoGENI está disponível através de um repositório GIT[20].

[17] `http://www.cs.utah.edu/flux/index.html`
[18] `http://www.emulab.net/`
[19] `http://www.protogeni.net/trac/protogeni/wiki/XMLRPCInterface`
[20] `http://users.emulab.net/trac/emulab/wiki/GitRepository`

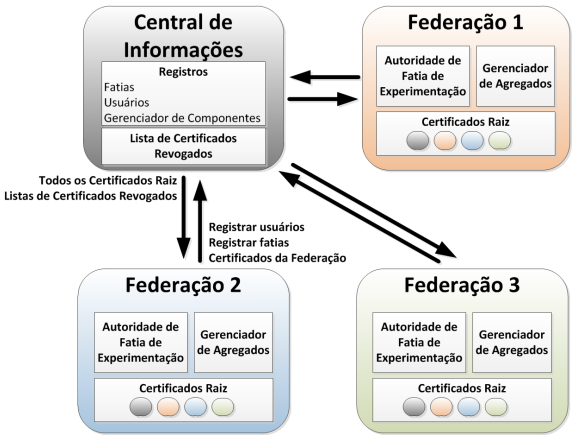


Figura 3.4. Visão Geral do arcabouço do ProtoGENI

3.5.2.1. Central de Informações

A Central de Informações é o elemento central para encontrar informações sobre os componentes gerenciados, usuários, fatias, etc. Seus objetivos são:

- Ajudar os Gerenciadores de Componentes e os utilizadores dos recursos (experimentadores) a encontrarem os elementos necessários para elaborar suas experiências através dos registros.
- Estabelecer confiança na federação, servindo como um ponto central de confiança, por onde todos os membros da federação podem trocar certificados e Listas de Certificados Revogados.

3.5.2.2. Autoridade de Fatias de Experimentação

A Autoridade de Fatias de Experimentação é responsável por uma ou mais fatias, sendo responsável por atribuir nomes, registrar e permitir que os usuários acessem e controlem suas fatias. A Autoridade de Fatias de Experimentação deve fornecer uma interface de contato para obter informações sobre as fatias ou para responder a qualquer abuso identificado nas fatias. Cada membro da federação executa um servidor XMLRPC para a Autoridade de Fatias de Experimentação. Os experimentadores utilizam certificados para se autenticarem com a Autoridade de Fatias de Experimentaçãos local, que por sua vez permite validar o usuário em toda a federação.

3.5.2.3. Gerenciador de Agregados

No ProtoGENI o elemento mais básico a ser gerenciado é o componente, que pode ser: um computador, roteador personalizável ou um ponto de acesso programável. Cada componente encapsula uma conjunto de recursos, que incluem recursos físicos (e.g., CPU, memória, disco, banda de rede), recursos lógicos (e.g., descritores de arquivo e números de portas) e recursos sintéticos (e.g., atalhos para encaminhamento de pacotes). Esses recursos podem estar contidos em um único dispositivo ou dispersos em um conjunto de dispositivos, dependendo da natureza de cada componente. É importante ressaltar que

um recurso só pode pertencer a um componente. Os componentes por sua vez podem ser agrupados em agregados, lembrando que todos os componentes de um agregado devem estar sob autoridade do mesmo Gerenciador de Agregados e que este também governa o agregado. O Gerenciador de Agregados exporta uma interface que torna o agregado remotamente acessível. Quando o Gerenciador de Agregados contém apenas um componente, ele pode ser considerado também um Gerenciador de Componentes. Tanto o Gerenciador de Agregados como o Gerenciador de Componentes definem as operações disponíveis aos serviços a nível do usuário para gerenciar as alocações dos recursos para diferentes usuários e experimentos.

O Gerenciador de Agregados do ProtoGENI suporta:

- Autenticação de usuários e verificação de credenciais;
- A *Component Manager API* que permite aos contribuidores informarem os componentes que estão disponibilizando para o PlanetLab;
- Anunciar os seus recursos disponíveis, através de RSpec;
- Aceitar a requisições de criação de fatias de qualquer membro da federação;
- Políticas locais simples para fornecer recursos a usuários de outros membros da federação;
- Criar multiplas instâncias (*slivers*) nos computadores; e
- Criar as topologias de rede entre os computadores disponíveis.

O arcabouço de controle do ProtoGENI possibilita a inclusão de vários agregados federados e seus componentes, possibilitando uma ampla gama de recursos aos experimentadores.

3.5.2.4. Exemplos

O ProtoGENI possui um formato próprio que é usado para a especificação dos recursos, chamado RSpec. O RSpec é utilizado para divulgar, requisitar e descrever os recursos disponíveis a serem usados pelos experimentadores, sendo este baseado em um formato similar ao utilizado pelo Emulab. Um exemplo simples para alocar um único nó:

```
set nodeA [$ns node]
tb-set-hardware $nodeA pcfedphys
tb-fix-node $nodeA urn:publicid:IDN+emulab.net+node+*
```

Nesse exemplo está sendo requisitado o nó *pcfedphys*, que é uma máquina física (foram apresentadas apenas as informações relevantes do arquivo NS, onde são definidas as associações dos componentes). Está especificado que o nó seja alocado a partir do *cluster* de Utah, mas não está especificado nenhum nó em específico, logo o sistema vai alocar qualquer nó livre. Caso seja desejado alocar um nó específico, pode-se mudar o URN para:

```
tb-fix-node $nodeA urn:publicid:IDN+emulab.net+node+pc333
```

Nesse comando é especificado um nó, porém este nó deve obrigatoriamente estar livre e disponível quando o experimento for instanciado.

Para instanciar o experimento, é necessário acessar a interface Web do Emulab/-ProtoGENI e utilizar a opção *Begin Experiment*. Na interface deve ser selecionado o projeto em que o experimento será configurado e preenchido os campos *Name* e *Description* com o nome e descrição do experimento, respectivamente. No campo *Your NS file*, deve ser informado o caminho para o arquivo NS criado anteriormente. Após a submissão, o experimento será processado e, se não houver erros, os recursos solicitados serão alocados. Quando o experimento estiver alocado, é possível verificar o sucesso da alocação através da página *Show Experiment*, onde o nó alocado para o experimento deverá estar listado. Este nó é um *proxy* para os recursos administrados pelo ProtoGENI, clicando sobre o nome do nó será possível acessar uma variedade de informações sobre este nó. O campo de *Hostname* é a informação que deve ser utilizada para acessar o nó via SSH.

Além de alocar máquinas físicas é também possível alocar máquinas virtuais. Caso a aplicação do usuário permita, será possível utilizar várias máquinas virtuais em uma única máquina física e, deste modo, oferecer uma outra alternativa de se obter todos os recursos desejados para o experimento. Exemplo:

```
set nodeA [$ns node]
set nodeB [$ns node]
tb-set-node-os $nodeA GENIVM
tb-set-hardware $nodeA pcfed
tb-fix-node $nodeA urn:publicid:IDN+emulab.net+node+*
tb-set-node-os $nodeB GENIVM
tb-set-hardware $nodeB pcfed
tb-fix-node $nodeA urn:publicid:IDN+emulab.net+node+*
```

Após instanciar o experimento, é possível visualizar na página *Show Experiment* os dois nós listados. Clicando sobre os nomes dos nós, é possível encontrar a informação de *Hostname*, bem como o número da porta SSH que é necessária para conectar-se a este nó. Por exemplo, se o *Hostname* for *pc76.emulab.net* e o número da porta 32333, então o comando para acessar este nó será:

```
ssh -p 32333 pc76.emulab.net
```

3.5.3. PrimoGENI

O PrimoGENI[21] é um agregado integrado ao arcabouço de controle do ProtoGENI para possibilitar a simulação de redes em tempo real. As instâncias do agregado PrimoGENI são compostas de recursos virtuais, como a rede virtual que inclui elementos simulados (roteadores, servidores, enlaces e protocolos) e elementos emulados (*hosts* e roteadores executando em máquinas virtuais); e os meta recursos associados à instanciação da rede virtual.

O PrimoGENI usa o ProtoGENI/Emulab para gerenciar, controlar e acessar os seus meta recursos. Além disso, o simulador PRIME [Pri 2012] é empregado para criar, gerenciar, controlar e acessar os recursos virtuais. Os meta recursos são instanciados a partir das instalações físicas definidas no ambiente ProtoGENI/Emulab. Por sua vez, os recursos virtuais são instanciados sobre os meta recursos previamente instanciados.

O PrimoGENI possui uma interface de interação com outros elementos do arcabouço de controle, que utiliza a API de Controle de Componente do ProtoGENI para se comunicar com as centrais de informações utilizando os protocolos XMLRPC, HTTPS e

[21] `http://groups.geni.net/geni/wiki/PrimoGENIDesignDocument`

SSH.

3.5.4. ORCABEN

O ORCABEN[22] é um arcabouço de controle que é parte integrante do Cluster D [gen 2012] do projeto GENI desenvolvido pelo Flux Research Group. O ORCABEN é um projeto conjunto entre o grupo NRIG (Rede de Pesquisa e Grupo de Infraestrutura) do Instituto RENCI[23] e o grupo NiCl da Universidade Duke. O projeto tem como objetivo adaptar o arcabouço de controle ORCA da NiCl à infraestrutura BEN[24], uma rede óptica experimental de escala metropolitana, para permitir ao ORCA criar e gerenciar fatias de experimentação na infraestrutura BEN em várias camadas: desde a camada física, através de DWDM (*Dense Wavelenght Division Multiplexing*) até as camadas 2 e 3.

3.5.4.1. Visão Geral do ORCA

O ORCA é um arcabouço de controle de código aberto para gerenciar recursos compartilhados controláveis através de programação, que podem conter qualquer combinação de servidores, equipamentos de armazenamento, redes, ou outros componentes de computação em nuvem.

Existem três agentes principais na arquitetura do ORCA, representando os provedores de recursos, experimentadores e intermediários, respectivamente. Podem haver várias instâncias de cada um desses agentes, por exemplo, que representem diferentes provedores de recursos ou diferentes experimentadores. São eles (figura 3.5):

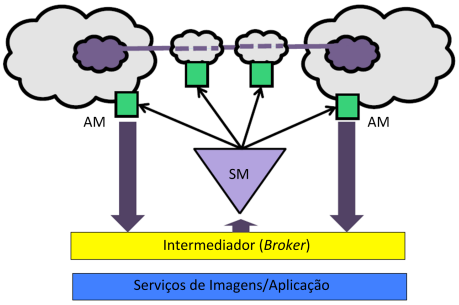


Figura 3.5. Visão Geral do ORCA

- Gerenciador de Agregados (*Aggregate Manager* – AM). Controla o acesso a um subconjunto de componentes;
- Gerenciador de Fatias de Experimentação (*Slice Manager* – SM). É responsável por criar, configurar e adaptar uma ou mais fatias;
- Intermediador (*Broker*). Possibilita a descoberta e alocação de recursos, controlando o agendamento dos recursos a um ou vários prestadores de recursos ao longo do tempo. Pode ser visto como um dos serviços que é executado dentro de um Centro de Informação.

[22] `http://groups.geni.net/geni/wiki/ORCABEN`
[23] `http://www.renci.org/`
[24] `https://ben.renci.org/`

O Gerenciador de Agregados disponibiliza os recursos para serem alocados através do Intermediador (um Gerenciador de Agregados pode disponibilizar os recursos para um ou mais Intermediadores). O Intermediador reserva os recursos requisitados até o SM solicitá-los. Os AMs criam as instâncias de recursos e passam o controle das instâncias ao SM.

3.5.4.2. Visão Geral do BEN

A infraestrutura BEN (Rede Experimental Particionável) é uma plataforma única para a experimentação de redes localizada no Research Triangle Park, na Carolina do Norte e é composto por vários segmentos de fibra escura, formando um recurso de tempo compartilhado que está disponível para a comunidade de pesquisa. O BEN é uma instalação criada e gerida por pesquisadores e cientistas, a fim de promover a descoberta científica, fornecendo uma infraestrutura e recursos para a experimentação com tecnologias de rede disruptivas.

Cada PoP RENCI[25] compartilha a arquitetura de rede, o que permite os seguintes recursos de apoio à investigação:

- Comutador de fibra reconfigurável (camada 0) fornece acesso dos pesquisadores às portas de fibra para atribuir dinamicamente novas conexões, o que possibilita configurar diferentes topologias físicas;
- Solução de gerenciamento *Out-of-band*, permitindo o acesso aos equipamentos instalados no PoP; e
- Gerenciamento remoto de energia para permitir a reinicialização automática e desligamento do equipamento experimental.

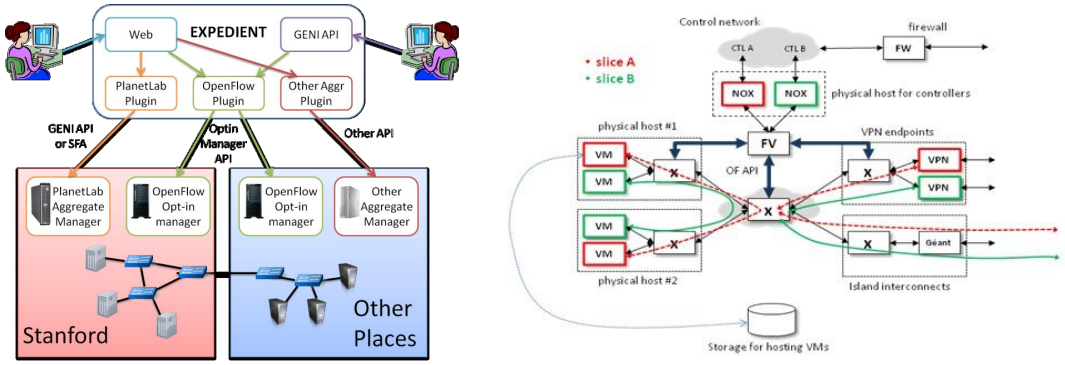
3.5.5. OFELIA-CF

O OFELIA é uma iniciativa da European Union 7th Framework Programme - FP7 que provê uma infraestrutura experimental (*testbed*) única baseada em Openflow, além do software OFELIA CF (OCF) estar disponível publicamente. Este projeto permite que pesquisadores não fiquem somente restritos a testar a rede, mas também, permite o completo controle da mesma de maneira precisa e dinâmica, a partir do substrato OpenFlow. Iniciou a operação em 2010 e possui 8 ilhas OpenFlow interconectadas na Europa. Tais ilhas permitem experimentação em multi-camadas e multi-tecnologias de redes, com programabilidade suficiente para o experimentador montar sua própria rede em nuvem.

3.5.5.1. Arquitetura

O OFELIA faz uso de vários serviços web e softwares como Expedient e Optin (Figura 3.6(a)) para gerenciar os recursos das ilhas OpenFlow. Sua interface web é inspirada em sites de viagens, nos quais o usuário pode reservar um vôo, hotel, carro, no mesmo sistema. A Figura 3.7(a) apresenta os principais componentes do arcabouço [Kopsel and Woesner 2011].

[25] `http://www.renci.org/`



(a) Arquitetura OFELIA (b) Parte Interna do OFELIA

Figura 3.6. OFELIA

Internamente o OFELIA CF (Figura 3.6(b)), utiliza-se do software *proxy* de controle do openflow, o chamado FlowVisor (FV na figura), para criar fatias de experimentação. O FlowVisor trabalha com o conceito de FlowSpaces, particionando a rede, do ponto de vista de controle, para suportar vários controladores OpenFlow ao mesmo tempo. Esse particionamento é baseado em conjuntos de campos de cabeçalhos como rótulos de VLAN onde, baseado nos cabeçalhos, mensagens de controle serão enviadas aos respectivos controladores dos usuários. A partir disso, um Controlador do experimentador pode controlar switches Openflow (representado por X na figura) da infraestrutura e instanciar máquinas virtuais (VMs) associadas a esses rótulos de VLAN para seu experimento. Maiores detalhes do OFELIA CF são apresentadas na seção 3.7.

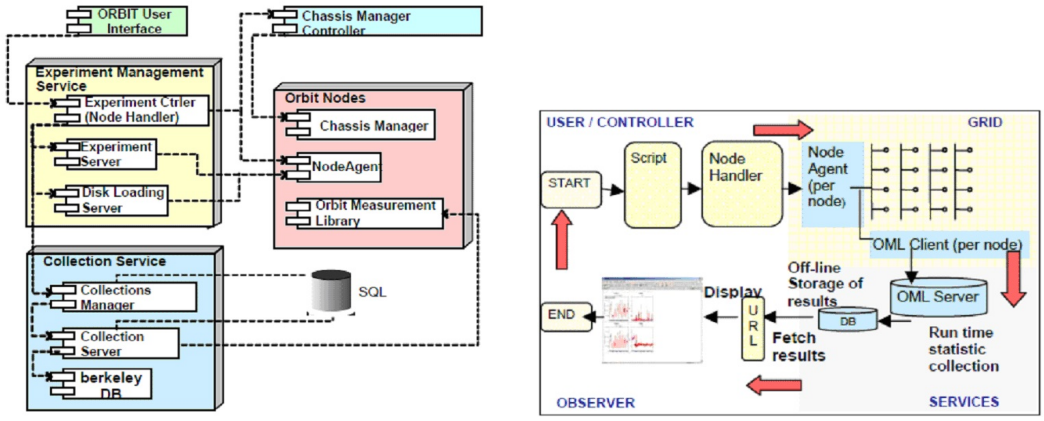
3.5.6. OMF

O ORBIT Management Framework (OMF) foi desenvolvido na Universidade Rutgers em conjunto com o NICTA (National ICT Australia) para controlar experimentos de redes sem-fio através da criação de *testbeds*. Seu desenvolvimento começou em 2003 e desde então passou a ser adotado em projetos de *testbeds* da Internet como módulo sem-fio.

3.5.6.1. Arquitetura

A Figura 3.7(a) apresenta os principais componentes do arcabouço [Raychaudhuri et al. 2005]:

- *Node Handler*: Dissemina *scripts* experimentais usando multicast para o nó Agente que residem nos nós individuais, a fim de orquestrar o experimento.
- *Collection Server*: Coleta as medidas notificadas durante o experimento.
- *Disk Loading Server*: Permitir uma rápida re-imagem de discos rígidos em nós conforme as exigências do experimentador.
- *Node Agent*: Trata-se do equivalente ao componente NodeHandler que reside em nós ORBIT e escuta comandos do Node Handler ORBIT.



(a) Arquitetura OMF (b) Workflow OMF

Figura 3.7. OMF

- *ORBIT Measurement Library (OML)*: Define as estruturas de dados e funções para envio / recepção e (de)codificação de dados de medição que são trocados em formato XDR.

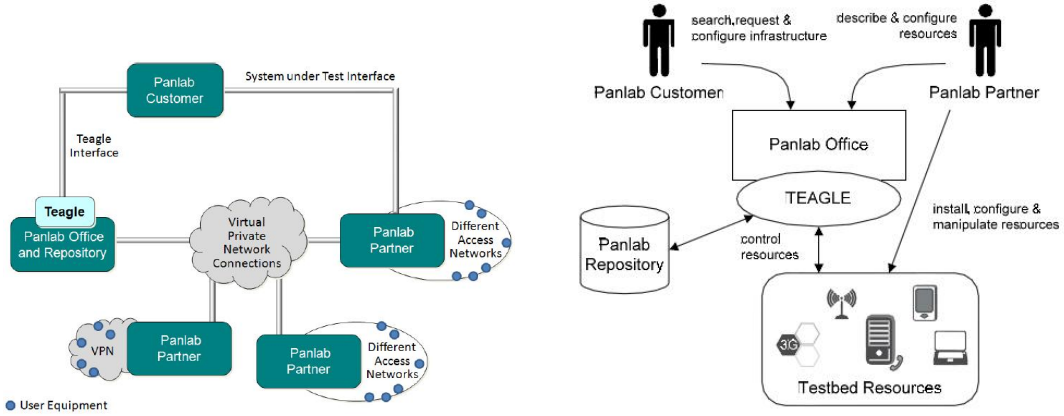
Para criar e executar experimentos em OMF, conforme a Figura 3.7(b), existe um *script* com as configurações detalhando e identificando as características dos nós do experimento a ser realizado. Ele é passado para o Node Handler que assume a responsabilidade de escalar os Node Agents. Cada Node Agent tem n antenas e um OML-Client que se conecta ao OML-Server. O OML-Server pega os dados do OML-Client gerado pelas antenas e então processa gerando as estatísticas que são armazenadas em Banco de Dados. Assim quando o observador acessar as páginas, essa acessará a base de dados gerando os gráficos.

3.5.6.2. Exemplos

Exemplos de pesquisas experimentais incluem redes para veiculares e redes P2P sem fio [Dwertmann et al. 2009]. A intenção dos pesquisadores é incluir interfaces com o PlanetLab, ORCA e GENI de tal forma que essas *testbeds* incluam o OMF nas suas respectivas *testbeds* [Stasi et al. 2009].

O *Network Implementation Testbed* ou NITOS é um *testbed* com foco em redes sem fio de código aberto, administrado pelo CERTH, um parceiro do projeto OneLab. O NITOS consiste de 50 nós localizados na Universidade de Thessaly, e já foram feitas pesquisas de colisão de pacotes, erros, frequências, algoritmos de controle de carga e otimização de tráfego, eficiência energética, etc. Também há um suporte a operações *off-line* importante em experimentos de mobilidade.

Existem outros *testbeds* sem fio menores que fazem parte do OneLab e usam também o OMF para prova de conceitos: ETH Zurich e INRIA. No total, mais de 20 *testbeds* do mundo todo usam o OMF, incluindo seis de larga escala (*meso-scale*) com suporte a WiMAX.



(a) Arquitetura Panlab (b) Sistema Teagle na arquitetura Panlab

Figura 3.8. Panlab

3.5.7. Panlab

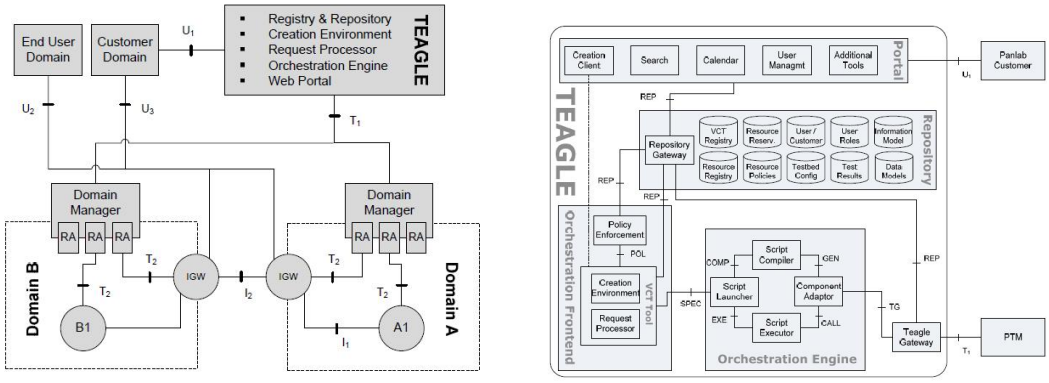
O *Pan-European Laboratory (Panlab)* é uma iniciativa do *European Union 6th Framework Programme - FP6*, com o objetivo de conceber uma plataforma de rede com o conceito de oferecer facilidades de testes sob a ótica de federação de *testbeds*. O conceito do Panlab é interconectar diversos laboratórios no conceito de federação e oferecer um serviço fim-a-fim coordenados a partir de uma entidade centralizada.

3.5.7.1. Arquitetura

O Panlab pode ser classificado como uma rede de domínios federados (Network Domain Federation — NDF) [Wahle et al. 2008] e é organizado conforme mostra a figura 3.8(a). A arquitetura é constituída por três entidades descritas a seguir:

- *Panlab Partners*: Entidades participantes do Panlab que oferecem elementos de infra-estrutura de comunicação e serviços (recursos). Cada parceiro possui sua própria infra-estrutura que é disponibilizada a partir de um acordo com a Federação Panlab.
- *Panlab Customer*: Entidade usuária dos serviços oferecidos pelo administrador (*Panlab Office*) utilizando os recursos dos parceiros (*Panlab Partners*).
- *Panlab Office*: Entidade administrativa que realiza a coordenação do sistema global e negocia os recursos de teste solicitados pelos usuários (*Panlab Customers*). Ela é responsável por provisionar a infraestrutura através dos participantes da federação (Panlab Partners) através de uma ferramenta apropriada.

A ligação física entre os parceiros para criar a rede de teste pode ser estabelecida conforme a tecnologia disponível. No entanto, consideramos que deve ser uma tecnologia que propicie uma rede sobreposta (*overlay*) definindo caminhos como *Virtual Private Network* (VPN) [Andersson and Madsen 2005], *Virtual Private LAN Service*



(a) Interfaces do sistema Teagle (b) Arquitetura do sistema Teagle

Figura 3.9. Teagle

(VPLS) [Kompella and Rekhter 2007], *Generic Routing Encapsulation* (GRE) ou *Virtual LAN* (VLAN). O elemento gerenciador central é chamado Teagle [Wahle 2008], cuja localização na arquitetura e funcionalidade é mostrada na figura 3.8(b).

3.5.7.2. Teagle

O sistema **Teagle** [Wahle 2008] é o elemento central de gerenciamento do Panlab implementando a configuração e orquestração do *testbed*. O Teagle provê uma interface para os usuários (*Panlab Customer*) que recebe suas necessidades e determina a melhor maneira de atendê-los. A interface baseada em Web apresenta aos usuários todas as informações sobre a plataforma de testes, assim como as funcionalidades disponíveis. Essa plataforma também pode combinar diferentes componentes de teste em uma entidade funcional única, tornando possível testes complexos onde um simples *testbed* não seria capaz de oferecer [Magedanz et al. 2008].

O Teagle é capaz de gerenciar vários domínios, cada qual com sua própria administração, que concordam em colaborar formando uma federação. O sistema dispõe das informações de cada domínio, e a partir da requisição do usuário, irá determinar o caminho mais adequado e estabelecer a melhor configuração. Cada domínio possui um gerenciador de domínio, *Domain Manager* — DM), responsável por realizar a configuração em seus próprios dispositivos.

Como uma federação não pressupõe a uniformidade de equipamentos, as interfaces entre seus componentes precisam estar bem definidas. A figura 3.9(a) mostra as interfaces do sistema Teagle e o DM. As interfaces U definem a relação com os usuários e as interfaces T entre o Teagle, os DMs e os equipamentos.

A figura 3.9(b) mostra a arquitetura detalhada do sistema Teagle. Essa ferramenta provê uma interface para o usuário baseada em Web possibilitando que ele possa visualizar os recursos disponíveis e realizar o provisionamento do *Virtual Customer Testbed* (VCT).

3.6. Federação de Testbeds: Criando Valor para Experimentadores (SFA 1.0 e 2.0)

Nesta seção, descreveremos o esforço que tem sido feito para definir, de uma maneira padronizada, um conjunto mínimo de interfaces e tipos de dados que permitem a interoperação de várias *testbeds* através do uso de abstrações como a federação do substrato das redes participantes e a alocação de recursos baseada em 'fatias'. Apresentaremos também os principais conceitos relacionados a federação, tais como: entidades, componentes, agregados, dentre outros. Por fim, descreveremos a principal API para federação conhecida como SFA (*Slice-based Federation Archicteture*).

3.6.1. Introdução

No contexto de infraestruturas para experimentação de redes de comunicação, a federação pode ser definida como a alocação unificada de recursos disponíveis em diferentes *testbeds* para realização de testes [OneLab2_Executive_Committee_and_guests 2009]. Os recursos disponíveis em uma *testbed* podem ser os mais variados, tais como máquinas (físicas ou virtuais), roteadores, *switches*, pontos de acesso sem fio, etc. Cada *testbed* pode ter um controle administrativo independente, o que leva a uma conceito chave em federação: existência de objetivos comuns. Em uma federação de *testbeds*, é esperado que o principal objetivo seja realizar experimentos, em geral, com o intuito de estudar, verificar, avaliar e/ou mensurar novas propostas (de algoritmos, mecanismos, arquiteturas, sistemas, etc) para redes de comunicação ou sistemas distribuídos. Alguns exemplos de outros objetivos potenciais da federação de *testbeds* são: A) aumentar a escala dos testes; B) aumentar o nível de realismo ao qual protótipos são expostos; C) ter acesso a recursos variados; D) economizar através de compartilhamento, acessando recursos de outras *testbeds*; E) melhorar a qualidade da pesquisa, trocando resultados e criando uma comunidade.

Por outro lado, a federação de *testbeds* precisa lidar com questões que afetam de diferentes maneiras seus objetivos. Em geral, essas questões surgem porque cada *testbed* em uma federação pode ter um conjunto de regras, políticas, hardware e software que influenciam sua relação com as demais, conforme ilustrado nos exemplos a seguir. Uma *testbed* pode suportar repetibilidade de experimentos através da captura de *traces* e reexecução (*replay*) dos mesmos, enquanto outra *testbed* semelhante em termos de hardware não oferece suporte em seu software de controle. Em uma federação podem haver *testbeds* que foram projetadas para garantir isolamento dos experimentos enquanto outras foram desenvolvidas para permitir variabilidade semelhante à do mundo real, ou seja, sem controle. A heterogeneidade de hardware pode ser muito alta em uma federação, dificultando a reserva de recursos e orquestração de experimentos.

Neste documento, vamos abordar a federação de infraestruturas para experimentação de redes de comunicação de maneira ampla, ou seja, considerando que *testbeds* com características notadamente diferentes e em diferentes partes do mundo podem ter interesse de se juntar a uma mesma federação. Por essa razão, apresentaremos a SFA em detalhes nas seções que seguem, pois ela é a proposta mais madura e abrangente para federação de *testbeds* em escala global. No entanto, vamos citar algumas propostas alternativas de escopo mais limitado. Em [Faber and Wroclawski 2008], os autores propõem

uma arquitetura de federação chamada DFA (DETER *Federation Architecture*) com enfoque em segurança de computadores. DETER (*cyber-DEfense Technology Experimental Research*) é baseado no software Netbed do Emulab [White et al. 2002], o qual foi uma das primeiras implementações de um arcabouço para controle e monitoramento de *testbeds*. Os autores de [Sanchez et al. 2011] propõem uma solução para federação de *testbeds* de rádios cognitivos. O objetivo é oferecer recursos para lidar com as necessidades específicas de redes cognitivas e de acesso dinâmico ao espectro.

3.6.2. SFA 1.0

Originalmente, a SFA [Peterson et al. 2009c] (que expandia originalmente para *Slice-based Facility Architecture*) foi proposta como uma especificação para suportar federação entre infraestruturas como PlanetLab, Emulab, VINI e GENI, mas deveria estar preparada para lidar com outras *testbeds* com projetos diferentes das anteriores. A especificação SFA define um conjunto mínimo de interfaces e tipos de dados que permitem a interoperabilidade de uma federação de *testbeds* que sejam baseadas em fatias de experimentação (ou *slices*). Conforme descrito em [Tronco 2010], o núcleo da SFA é a autorização baseada em autenticação de usuários de *testbed*. A autenticação é realizada através de certificados X.509 e a autorização é feita de maneira distribuída com base em políticas que são definidas localmente pelo proprietário do recurso. Assim, os principais aspectos arquiteturais da SFA podem ser resumidos como um mecanismo de autenticação e uma linguagem para descrever recursos, requisitá-los e conceder acesso à fatia de experimentação.

A SFA define quatro tipos de perfis de usuários na federação, chamados de atores:

- **Proprietários** – são os donos de partes do substrato de rede, sendo responsáveis por definir o comportamento visível externamente de seus equipamentos e estabelecer as políticas de alto nível para uso da sua porção do substrato.

- **Operadores** – administram partes do substrato de rede, em geral trabalhando para os proprietários. As principais tarefas dos operadores são manter a plataforma funcionando, fornecer um serviço aos pesquisadores e prevenir atividades maliciosas ou que possam provocar danos à plataforma.

- **Pesquisadores e desenvolvedores** – empregam o substrato de rede para realizar experimentos, implantar serviços experimentais, mensurar aspectos da plataforma, dentre outras ações.

- **Investigadores principais** (**PIs** - *Principal Investigators*) – representam organizações de pesquisa que assumem a responsabilidade por pesquisadores individuais.

A SFA se propõe a mediar as seguintes atividades:

- Permitir que proprietários declarem políticas para alocação e uso de recursos para infraestruturas sobre seu controle e fornecer mecanismos para fazer cumprir essas políticas. A suposição é que existirão múltiplos proprietários e uma federação das infraestruturas que formarão a totalidade da rede de testes.

- Permitir que operadores gerenciem o substrato de rede, o que inclui instalação de novo hardware e remoção de antigo ou com defeito, instalação e atualização de software e monitoração da rede em termos de desempenho, funcionalidade e segurança. O gerenciamento é provavelmente descentralizado, havendo mais de uma organização administrando coleções disjuntas de infraestruturas.
- Permitir que pesquisadores criem e populem fatias de experimentação, reservem recursos para elas e executem experimentos nelas.
- Permitir que PIs identifiquem o conjunto de pesquisadores da sua organização que possuem permissão para utilizar uma infraestrutura.

A SFA define três entidades principais no contexto da federação:

- **Autoridade de gerenciamento** (**MA** – *Management Authority*) – é responsável por um subconjunto de componentes do substrato, tendo as seguintes atribuições: fornecer estabilidade operacional para os componentes, garantir que os componentes se comportem de acordo com as políticas de uso e executar a alocação de recursos desejada pelo proprietário do componente.
- **Autoridade de fatia de experimentação** (**SA** – *Slice Authority*) – é responsável pelo comportamento de um conjunto de fatias de experimentação, autenticando os pesquisadores que realizam experimentos em cada fatia e tomando as devidas ações se houver falha ou uso indevido da fatia de experimentação.
- **Usuário** – é uma pessoa executando um ou mais papéis em uma infraestrutura. Exemplos: um pesquisador que deseja executar um experimento, um operador que gerencia alguma parte do substrato, um PI de uma instituição que conduz uma pesquisa na infraestrutura ou um proprietário que oferece recursos para uma infraestrutura.

Nas subseções a seguir, apresentamos uma síntese das abstrações, do esquema de nomes e identificadores usados na SFA. Apresentaremos também os tipos de dados e interfaces definidas pela SFA. Por fim, vamos mostrar as origens e o fluxo de confiança através de um sistema baseado em SFA. Detalhes sobre todos esses tópicos podem ser encontrados na especificação da SFA 1.04 [Peterson et al. 2009c].

3.6.2.1. Abstrações

Duas abstrações são fundamentais: componentes e fatias de experimentação (*slices*).

Componentes são blocos de construção básicos da arquitetura. São exemplos de componentes: computador, roteador customizável e ponto de acesso programável. Um componente encapsula uma coleção de recursos que podem ser físicos (como CPU, memória e disco), lógicos (como descritores de arquivos e números de portas) e sintéticos (como caminhos para encaminhamento de pacotes). Os recursos podem estar contidos em um único dispositivo físico ou distribuído por um conjunto deles, dependendo da natureza do componente. No entanto, um recurso pertence a apenas um componente.

Cada componente é controlado através de um gerenciador de componentes (CM – *Component Manager*) que exporta uma interface bem definida acessível remotamente. O gerenciador de componentes define as operações disponíveis para gerenciar a alocação de recursos para diferentes usuários e seus experimentos. Uma autoridade de gerenciamento (MA) estabelece as políticas sobre como os recursos são atribuídos aos usuários.

Deve ser possível multiplexar (ou fatiar) recursos de componentes entre múltiplos usuários. O fatiamento pode ser realizado através de virtualização do componente ou do particionamento do componente em conjuntos de recursos distintos. Em ambas as abordagens, o recurso oferecido ao usuário é chamado de porção (*sliver*) do componente. Cada componente deve incluir mecanismos de hardware ou software que isolem as porções umas das outras, de maneira que uma porção possa ser tratada como um container de recursos.

Algumas vezes é conveniente representar uma coleção de componentes como um único agregado. Por exemplo, nós e enlaces de uma rede podem ser tratados como um agregado. Um agregado pode ser acessado através de um gerenciador de agregado (AM – *Aggregate Manager*) que exporta uma interface semelhante a de um componente individual.

Fatia de experimentação (*slice*) é definida pelo conjunto de porções (*slivers*) espalhadas por um conjunto de componentes e o conjunto associado de usuários que possuem permissão para acesso às referidas porções. Uma fatia de experimentação recebe um nome e um conjunto (que pode ser vazio) de porções. A partir da perspectiva de um pesquisador, uma fatia de experimentação é uma rede de recursos de computação e comunicação capaz de executar um experimento ou serviço de rede experimental federada. A partir da perspectiva de um operador, fatia de experimentação constitui a abstração primária para fins de contabilidade.

Uma fatia de experimentação tem três estágios durante a sua existência, cada um correspondendo a uma ação (operação) que pode ser realizada na fatia, a saber: 1) Registrar (*Register*) – a fatia recebe um nome e é vinculada a um conjunto de usuários; 2) Instanciar (*Instantiate*) – a fatia é instanciada em um conjunto de componentes e os recursos são atribuídos a ela; 3) Ativar (*Activate*) – a fatia é ativada (inicializada), realizando o experimento do usuário.

As fatias são registradas em uma SA apenas uma vez, mas o conjunto de usuários associados pode mudar ao longo do tempo. O registro de uma fatia tem um tempo de vida limitado e é responsabilidade da SA atualizar o registro periodicamente. A instanciação de um fatia pode ser repetida múltiplas vezes e frequentemente envolve dois passos. Uma fatia é inicialmente instanciada em um conjunto de componentes com recursos atribuídos em um regime de melhor-esforço, ou seja, sem garantias. No segundo passo, a fatia pode receber recursos adicionais, eventualmente, de maneira garantida. Durante a fase de ativação, múltiplos experimentos podem ser executados em uma única fatia. Para cada execução de experimento, os parâmetros, o conjunto de componentes e o conjunto de recursos podem ser alterados. A velocidade de reconfiguração de uma fatia de experimentação depende de como as operações de provisionamento e instanciação são implementadas.

3.6.2.2. Nomes e Identificadores

A SFA define um identificador global (GID – *Global IDentifier*) para cada objeto que compõe uma federação. Exemplos de objetos: componentes, fatias, serviços e as três entidades principais (MA, SA e usuário). Os GIDs formam a base para um sistema seguro, no qual uma entidade que possui um GID é capaz de confirmar sua legitimidade assim como verificar a autenticidade de objetos que também possuam GIDs.

Um GID é um certificado que associa três informações:

```
GID = (PublicKey, UUID, Lifetime)
```

O objeto identificado pelo GID mantém a chave privada correspondente à chave pública (`PublicKey`), formando a base para a autenticação. Cada objeto possui um identificador universalmente único (UUID – *Universally Unique IDentifier*) [x667 2004] que é imutável e absoluto. Ou seja, o UUID permanece o mesmo se a chave pública mudar e identifica o objeto univocamente dentro de uma federação. O tempo de vida (`Lifetime`) descreve a validade do GID, o qual precisa ser periodicamente atualizado. Uma autoridade também é identificada por seu próprio GID. Logo, é possível verificar um GID através de chaves criptográficas que levam de volta, possivelmente em uma cadeia, até uma ou múltiplas raízes bem conhecidas.

A SFA define um registro como o mapeamento de um nome (humanamente) legível (HRN – *Human-Readable Name*) para um GID, podendo incluir informações adicionais sobre o objeto, tais como: o URI (*Uniform Resource Identifier*) do gerenciador de objetos, o endereço IP (ou de hardware) da máquina na qual o objeto é implementado, o nome e endereço postal da organização que hospeda o objeto, dentre outras informações.

De maneira semelhante a um nome no serviço DNS (*Domain Name System*), um HRN para um objeto identifica a sequência de autoridades que são responsáveis pelo objeto. A SFA permite que os registros sejam organizados de maneira arbitrária. No entanto, em uma federação efetivamente operacional é recomendado que o espaço de nomes siga uma estrutura hierárquica das autoridades que delegam o direito de criar e nomear componentes e fatias de experimentação. Essa hierarquia deve assumir uma autoridade de nível mais alto que seja confiável para todas as entidades, resultando em nomes com o seguinte formato:

```
top-level_authority.sub_authority.sub_authority.name
```

Exemplos de autoridades de nível mais alto (`top-level_authority`) seriam `geni`, `planetlab` e `fibre`. O registro mantém informação sobre uma hierarquia de MAs junto com o conjunto de componentes pelos quais as MAs são responsáveis. Por exemplo: `fibre.br.rnp.go.gyn`. Esse registro identificaria um componente no PoP (*Point of Presence*) de Goiás (situado em Goiânia) que faz parte do *backbone* da RNP.

O registro mantém também informação sobre a hierarquia de SAs junto com o conjunto de fatias de experimentação pelas quais as SAs são responsáveis. Por exemplo: `fibre.br.ufg.inf`. Esse registro daria nome a uma fatia de experimentação criada pela SA FIBRE, a qual delegou para o BR que repassou para a UFG o direito de aprovar fatias para projetos ou grupos de pesquisa nessa instituição (e.g. INF).

Por fim, é importante observar que um registro pode ser distribuído, tendo um servidor que implementa uma parte de uma hierarquia e que aponta (através de um URI) para um servidor que implementa uma sub-árvore da hierarquia.

3.6.2.3. Tipos de Dados

A SFA define quatro tipos principais de dados além do GID: especificação de recurso (RSpec – *Resource Specification*, também usado no ProtoGENI), registro (*Registry Record*), bilhete (*Ticket*) e credencial (*Credential*). A seguir, é apresentada a definição abstrata de cada um desses tipos de dados, sendo que a definição concreta está fora do escopo da SFA [Peterson et al. 2009c].

Uma **RSpec** descreve um componente em termos de recursos que ele possui e suas restrições e dependências na alocação de recursos. A forma exata de uma *RSpec* não é definida pela especificação da SFA, mas os dois campos seguintes devem existir além das informações sobre os recursos dos componentes: `(StartTime, Duration)`. Esses campos indicam durante quanto tempo se deseja reservar os recursos ou durante quanto tempo foi autorizado o uso dos recursos.

Um **Registro** (*registry record*) guarda informações sobre objetos em um sistema (por exemplo, componentes e fatias de experimentação) e sobre entidades (por exemplo, MAs, SAs e usuários). Os *registry records* são definidos de acordo com o seguinte formato:

```
Record = (HRN, GID, Type, Info).
```

Onde `Type` pode ser `SA`, `MA`, `Component`, `Slice` ou `User`. De acordo com o valor de `Type`, `Info` pode receber diferentes conteúdos, a saber:
se `Type = SA`, então `Info = (PI[], Organization)`
se `Type = MA`, então `Info = (Owner[], Operator[], Organization)`
se `Type = Component`, então `Info = (URI, LatLong, IP, DNS)`
se `Type = Slice`, então `Info = (URI, Researcher[], InitScript)`
se `Type = User`, então `Info = (PostalAddr, Phone, Email, AuthTokens[])`

Em geral, as informações disponíveis em um registro são relativamente estáticas. Para obter informações mais detalhadas e dinâmicas sobre um objeto é necessário contactar quem o controla ou responde por ele. Por exemplo, para obter mais informações sobre um componente, o seu CM deve ser contactado através do URI fornecido pelo seu *registry record*.

Um componente assina uma `RSpec` para produzir um **Ticket**, indicando um compromisso pelo componente de vincular os recursos ao portador do bilhete em algum momento. Esses bilhetes são emitidos por um componente e, posteriormente, resgatados para a aquisição de recursos no componente. De acordo com a SFA, os bilhetes devem receber as seguintes informações: `Ticket = (RSpec, GID, SeqNum)`. O número de sequência (`SeqNum`) garante que o bilhete é seja único. Essa informação é assinada pelo componente que emite o bilhete.

Uma **Credencial** carrega os direitos emitidos para uma entidade em particular. Por

exemplo, um usuário pode receber uma credencial que o permite instanciar uma fatia de experimentação em um conjunto de componentes por um determinado período de tempo. Uma credencial é descrita pela seguinte tupla:

`Credential = (CallerGID, ObjectGID, ObjectHRN, Expires, Privileges, Delegate)`

Onde `CallerGID` identifica a entidade para a qual a credencial foi emitida; `ObjectGID` e `ObjectHRN` identificam o objeto para o qual a credencial se aplica; `Expires` informa por quanto tempo a credencial é válida, `Privileges` identifica a classe de operações que o detentor da credencial tem permissão para invocar; e `Delegate` informa se o detentor da credencial pode delegar a credencial para outra entidade.

Cada privilégio (`Privilege`) implica no direito de invocar uma certa quantidade de operações em uma ou mais interfaces da SFA. A Tabela 3.1 resume a relação entre privilégios, interfaces e operações.

Tabela 3.1. Operações em cada interface de acordo com o privilégio.

Privilégio	Interface	Operações
`authority`	`Registry`	todas
`refresh`	`Registry`	`Remove, Update`
`resolve`	`Registry`	`Resolve, List, GetCredential`
`pi`	`Slice`	todas
`instantiate`	`Slice`	`GetTicket, CreateSlice, DeleteSlice, UpdateSlice`
`bind`	`Slice`	`GetTicket, LoanResources`
`control`	`Slice`	`UpdateSlice, StopSlice, StartSlice, DeleteSlice`
`info`	`Slice`	`ListSlices, ListComponents, GetSliceResources, GetSliceBySignature`
`operator`	`Management`	todas

3.6.2.4. Interfaces

Semelhante aos tipos de dados, as interfaces são definidas apenas em alto nível pela especificação da SFA. Devido à limitação de espaço, apenas listaremos as operações suportadas em cada interface e seus parâmetros. Detalhes sobre as operações podem ser encontradas em [Peterson et al. 2009c].

Interface *Registry* – Suporta seis operações: 1) `Register(Credential, Record)`, 2) `Remove(Credential, Record)`, 3) `Update(Credential, Record)`, 4) `Record = Resolve(Credential, HRN, Type)`, 5) `Record[] = List(Credential, HRN, Type)` e 6) `Credential = GetCredential(Credential, HRN, Type)`.

Interface *Slice* – Suporta 14 operações que são separadas em 4 grupos de acordo com sua função, a saber: instanciação, provisionamento, controle e informação.

Instanciação de Fatia: 1) `Ticket = GetTicket(Credential, RSpec)`, 2) `RedeemTicket(Ticket)`, 3) `ReleaseTicket(Ticket)` e 4) `CreateSlice(Credential, RSpec)`.
Provisionamento de Fatia: 1) `NewTicket = SplitTicket(Ticket, GID, RSpec)`, 2) `LoanResources(Credential, GID, RSpec)` e 3) `UpdateSlice(Credential, RSpec)`.
Controle de Fatia: 1) `StartSlice(Credential)`, 2) `StopSlice(Credential)`, 3) `ResetSlice(Credential)` e 4) `DeleteSlice(Credential)`.
Informação sobre Fatia: 1) `SlicesNames[] = ListSlices(Credential)`, 2) `RSpec = GetResources(Credential)` e 3) `SliceName = GetSliceBySignature(Credential, Signature)`.

É importante destacar que a interface *slice* é usada para criar e controlar fatias de experimentação, ou seja, ela define um plano de controle para as fatias. A partir da interface *slice*, é possível obter uma fatia de experimentação devidamente instanciada e é onde termina as atribuições do plano de controle proposto pela SFA. O comportamento individual das porções, isto é, como são acessadas, programadas e usadas, é específico de cada componente.

Interface *Management* – Suporta três operações: 1) `SetBootState(Credential, State)`, 2) `State = GetBootState(Credential)` e 3) `Reboot(Credential)`.

3.6.2.5. Controle de Acesso e Autorização

Todos os direitos relacionados a fatias de experimentação originam com SAs que aprovam ou se responsabilizam por fatias e usuários associados com elas. Cada SA tem implicitamente o privilégio `authority` para os registros correspondentes ao conjunto de usuários e fatias pelas quais é responsável.

Todos os direitos relacionados a recursos de componentes originam com MAs que definem as políticas de alocação de recursos (para os componentes que eles gerenciam) e aprovam todos os usuários que operam tais componentes. Cada MA tem implicitamente o privilégio `authority` para os registros correspondentes ao conjunto de usuários e componentes pelos quais é responsável.

Usuários, componentes e autoridades recebem o privilégio `refresh` para seus próprios registros. Usuários também possuem o privilégio `refresh` para as fatias de experimentação a que estejam afiliados. Todos os usuários e autoridades recebem o privilégio `resolve` para todos os registros e o privilégio `info` para todas as fatias.

Usuários associados com uma SA (i.e. PIs) têm o privilégio `pi` para todas as fatias registradas naquela SA, assim como em todas as fatias registradas em qualquer subautoridade que tem aquela SA como raiz. Usuários associados com uma fatia (i.e. pesquisadores) possuem os privilégios de `instantiate`, `bind` e `control` para aquela fatia. Todos os usuários (pesquisadores) recebem o privilégio `info` para todas as fatias e para

todos os componentes que hospedam fatias.

Usuários associados com uma MA (i.e. operadores) possuem o privilégio `operator` para todos componentes gerenciados por aquela MA. Operadores também recebem o privilégio `pi` em todos os componentes que eles gerenciam, incluindo todas as porções das fatias hospedadas nesses componentes.

Cada componente implementa uma política de alocação de recursos que determina quantos recursos são atribuídos a cada fatia de experimentação. Um usuário que recebe os privilégios `instantiate` ou `bind` para uma determinada fatia é visto como tendo o direito de solicitar recursos a um componente. No entanto, depende de cada componente a decisão de hospedar ou não uma fatia e de quanto recurso alocar a ela.

3.6.3. SFA 2.0

A versão 2.0 da SFA [Peterson et al. 2010] (que passou a expandir para *Slice-based Federation Architecture*) é uma tentativa de unificar implementações diferentes, por exemplo, as apresentadas em [Peterson et al. 2009a, Peterson et al. 2009b] que foram baseadas em sua versão anterior [Peterson et al. 2009c]. Basicamente, a versão 2.0 consolida as adequações que foram identificadas a partir da experiência adquirida com as implementações.

3.6.3.1. Principais Diferenças entre as Versões 1.0 e 2.0

Inicialmente, a versão 2.0, substitui o usuário PI por um novo tipo de usuário mais geral, chamado âncora de identidade (*identity anchor*) ou fornecedor de identidade (IdP – *Identity Provider*). O papel do IdP é conduzir a autorização através da declaração de atributos ou papéis para outros usuários. Por exemplo, um IdP poderia declarar que um determinado usuário é um PI de uma instituição de pesquisa.

Como consequência da substituição do usuário PI, o privilégio `pi` não é mais definido como padrão nas credenciais da versão 2.0.

As definições de AM e de CM foram simplificadas na versão 2.0, deixando claro que CM é apenas um caso especial de um AM que contém um único componente.

Na versão 2.0, houve uma reestruturação significativa na parte de registro que passou a ser tratada de maneira unificada, incluindo o esquema de nomes, credenciais, controle de acesso, autorização, tipos de dados e interface. Embora ainda existam informações sobre registro em outras partes do texto da versão 2.0 da SFA, a maior parte passou a se concentrar em uma única seção mais coesa.

A operação `ListSlices(Credential)` foi modificada para fornecer GIDs e, opcionalmente, nomes simbólicos ou HRNs. Como consequência, na versão 2.0, houve alteração na outra operação relacionada a informações sobre a fatia de experimentação: `RSpec = GetResources(Credential, GID)`

Na parte de controle de acesso e autorização, a versão 2.0 propõe que seja possível utilizar um arcabouço ABAC (*Attribute-Based Access Control*) ou um subconjunto adaptado às necessidades da federação.

Por fim, a versão 2.0 apresenta algumas considerações sobre o uso de um sistema de capacidade (*capability system*) que é um caso especial de um arcabouço ABAC. Em um sistema de capacidade, todos os atributos representam diretamente privilégios específicos para objetos específicos. Essa característica oferece uma simplificação significativa, uma vez que uma credencial pode representar diretamente os privilégios que ela habilita. Assim, qualquer entidade pode determinar os privilégios inspecionando apenas a credencial, sem a necessidade de um procedimento para inferência.

3.7. Estudo de Caso de Implantação do CMF OFELIA em uma Testbed OpenFlow

Nesta seção, nosso objetivo é mostrar um exemplo prático que possa orientar outros pesquisadores que gostariam de instalar *testbeds* experimentais locais em seus departamentos. Para tanto, demonstraremos o Sistema de Controle OFELIA (OCF), que oferece serviços de experimentação em redes baseadas em OpenFlow.

3.7.1. Requisitos e infraestrutura

Conforme apresentado na Seção 3.5, o Sistema de Controle (CF) do OFELIA é subdividido em três componentes: Expedient, VT Manager e Optin Manager. Esses componentes são implantados preferencialmente em um mesmo equipamento, o qual será denominado servidor ao longo deste documento. Os equipamentos responsáveis pela provisão de recursos de virtualização para os experimentos são chamados de agentes.

Os componentes do OFELIA são escritos na linguagem Python, utilizando o *framework* Django para as interfaces web. Nota-se que o OFELIA possui como dependência a versão 2 da linguagem. Adicionalmente, ele utiliza um conjunto de serviços que devem ser instalados previamente.

Tanto para o servidor quanto para os agentes, recomenda-se a utilização de um computador de arquitetura x86 ou x86_64 com a distribuição Linux Debian Squeeze. Utiliza-se o software de virtualização Xen nos agentes, sendo que não é necessário equipamento com suporte de virtualização por hardware (HVM).

Neste documento, assume-se que o servidor possui ao menos duas interfaces de rede, utilizadas respectivamente para tráfego *out-of-band* e de controle. Os agentes devem possuir pelo menos três interfaces de rede; as primeiras interfaces serão utilizadas para tráfego *out-of-band* e de controle, enquanto as demais serão utilizadas para a rede de experimentação. É sugerido que o software de autoconfiguração da rede (*NetworkManager*) seja desativado ou desinstalado.

3.7.2. Instalando e Configurando o OFELIA CF

3.7.2.1. Configuração da Rede — Máquina Servidor

Na configuração utilizada, uma interface de rede (eth0) é utilizada para tráfego *out-of-band*, enquanto outra (eth1) é utilizada para a rede de controle do OFELIA. Neste documento, será utilizada a VLAN 999 para o tráfego de controle. A seguir é apresentado o conteúdo do arquivo `/etc/network/interfaces` para a configuração descrita:

```
auto eth0
iface eth0 inet static
address 192.168.254.193
netmask 255.255.255.0

auto eth1.999
iface eth1 inet static
address 172.16.1.101
netmask 255.255.255.0
vlan_raw_device eth1
```

Nota-se que a configuração de VLANs possui como dependência o pacote vlan (`apt-get install vlan`). Após a configuração, é necessário reiniciar as interfaces (`/etc/init.d/networking restart`).

3.7.2.2. Instalação do Flowvisor - Máquina Servidor

O Flowvisor é um controlador utilizado como proxy transparente para outros controladores, multiplexando a infraestrutura de rede OF para vários experimentos e isolando-os entre si. Para seu funcionamento, é necessário instalar um ambiente Java. Assumindo que o repositório `nonfree` esteja habilitado no Debian, a instalação pode ser realizada da seguinte forma:

```
vim /etc/apt/sources.list (repositório non-free habilitado)
apt-get update
apt-get install sun-java6-jre sun-java6-jre
apt-get install sun-java6-plugin ant
update-java-alternatives -s java-6-sun
```

Para compilar o Flowvisor a partir do código-fonte disponibilizado no repositório GIT, é interessante instalar ferramentas de desenvolvimento adicionais. Abaixo está indicado o procedimento para compilação do Flowvisor:

```
apt-get install build-essential autoconf automake libtool git
git clone git://gitosis.stanford.edu/flowvisor.git
cd flowvisor
make
adduser flowvisor
make install -prefix=/usr
update-rc.d flowvisor defaults
/etc/init.d/flowvisor start
```

3.7.2.3. MySQL — Máquina Servidor

O OFELIA CF utiliza o banco de dados MySQL para armazenar as informações dos usuários, projetos, slices e demais recursos. Após a instalação do software é necessário criar um banco de dados para cada componente, conforme descrito a seguir.

```
apt-get install mysql-server
mysql -p

create user 'fibre' identified by 'senha';
create database expedient;
create database optin;
create database vt_am;

grant all on expedient.* to 'fibre' identified by 'senha' ;
grant all on optin.* to 'fibre' identified by 'senha' ;
grant all on vt_am.* to 'fibre' identified by 'senha' ;
\q
```

Por motivos de segurança, recomenda-se concatenar uma *string* aleatória ao final do nome de cada *database* em um ambiente de produção.

3.7.2.4. OFELIA CF — Máquina Servidor

A instalação do OFELIA CF é realizada a partir do `tarball` existente na página do projeto no Codebasin [26]. O conteúdo do `tarball` obtido deve ser extraído no diretório `/opt/ofelia`.

Para cada componente do OFELIA CF há um diretório correspondente. A instalação é realizada com a execução do *script* `ofver`, localizado no subdiretório `bin` de cada componente. Ressalta-se que o *script* é sensível ao diretório corrente, sendo necessário estar no diretório `bin` para executá-lo corretamente. O procedimento de execução dos *scripts* é demonstrado abaixo:

```
cd /opt/ofelia/expedient/bin
ofver install -f

cd /opt/ofelia/optin_manager/bin
ofver install -f

cd /opt/ofelia/vt_manager/bin
ofver install -f
```

Dependências adicionais serão encontradas e instaladas automaticamente durante o processo de instalação. Para cada componente é gerado um certificado SSL com as informações fornecidas na instalação. Os certificados serão utilizados pelo software Apache para disponibilizar as interfaces web dos componentes.

Durante a instalação, requisita-se a edição de arquivos de configuração do OFELIA CF. Os parâmetros a serem editados definem o nome de usuário, senha e e-mail de um usuário administrador inicial, dados para conexão com o banco de dados, assim como o IP e porta para a interface do componente. Abaixo encontram-se trechos da configuração dos componentes Expedient, Optin Manager e VT Manager, considerando a configuração descrita neste documento.

Por padrão, o Expedient utiliza a porta 443, o Optin Manager utiliza a porta 8443, e o VT Manager utiliza a porta 8445. Todas as interfaces são disponibilizadas somente por HTTPS.

[26] `http://codebasin.net/redmine/projects/ocf`

```
Expedient
ROOT_USERNAME = "fibre"
ROOT_PASSWORD = "minha_senha"
ROOT_EMAIL = "meu@email.com"
ISLAND_NAME = "minha_ilha"

ADMINS = [
("Nome do Admin", ROOT_EMAIL),
]
MANAGERS = ADMINS

"' SITE_DOMAIN deve ser alterado em um ambiente de produção
SITE_DOMAIN = "192.168.254.193"
SITE_IP_ADDR = "192.168.254.193"

DATABASE_USER = "fibre"
DATABASE_PASSWORD = "senha_bd"
DATABASE_HOST = "127.0.0.1"
DATABASE_NAME = "expedient"

VT Manager
"'General parameters"'
ROOT_USERNAME = "fibre"
ROOT_PASSWORD = "minha_senha"
ROOT_EMAIL = "meu@email.com"
VTAM_IP = "192.168.254.193"
VTAM_PORT = "8445"
ISLAND_NAME = "minha_ilha"

"'Database parameters"'
DATABASE_NAME = "vt_am"
DATABASE_USER = "fibre"
DATABASE_PASSWORD = "senha_bd"
DATABASE_HOST = "127.0.0.1"

Optin Manager
ROOT_USERNAME = "fibre"
ROOT_PASSWORD = "minha_senha"
ROOT_EMAIL = "meu@email.com"
ISLAND_NAME = "minha_ilha"

ADMINS = [
("Nome do Admin", ROOT_EMAIL),
]
MANAGERS = ADMINS

"' SITE_DOMAIN deve ser alterado em um ambiente de produção
SITE_DOMAIN = "192.168.254.193"
SITE_IP_ADDR = "192.168.254.193"

DATABASE_NAME = "optin"
DATABASE_HOST = "127.0.0.1"
DATABASE_USER = "fibre"
DATABASE_PASSWORD = "senha_bd"
```

3.7.2.5. Configuração da Rede — Máquinas Agentes

Para os agentes, uma interface de rede (eth0) é utilizada para tráfego *out-of-band*. Uma segunda interface (eth1) é utilizada para a rede de controle do OFELIA. As demais interfaces serão utilizadas para a rede de experimentação. A configuração abaixo representa o arquivo de configuração `/etc/network/interfaces` para um agente:

```
auto eth0
iface eth0 inet static
address 192.168.254.148
netmask 255.255.255.0

auto eth1.999
iface eth1 inet static
address 172.16.1.103
netmask 255.255.255.0
vlan_raw_device eth1

auto eth2
auto eth3
## [...]
```

3.7.2.6. Instalação e Configuração do Xen — Máquinas Agentes

O OFELIA utiliza o *hypervisor* Xen como plataforma de virtualização. No Debian Squeeze, a instalação básica pode ser realizada da seguinte maneira:

```
# Distribuição x86
apt-get install xen-linux-system-2.6-xen-686 vlan python-libvirt libvirt-dev xen-tools
# Distribuição x86_64
apt-get install xen-linux-system-2.6-xen-amd64 vlan python-libvirt libvirt-dev xen-tools
```

É necessário reiniciar o computador e selecionar o *kernel* com Xen habilitado no *boot*. Uma vez reiniciado, deve-se configurar o Xen. No arquivo`/etc/xen/xend-config`, remover os comentários das seguintes linhas:

```
(xend-http-server yes)
(xend-port 8000)

#Adicionar o 'network-multi-bridge-vlan' manualmente
(network-script 'network-multi-bridge-vlan')
```

A seguir, as linhas abaixo devem ser inseridas no arquivo `/etc/modules`, de forma a carregar explicitamente os respectivos módulos:

```
8021q
loop max_loop=64
```

No arquivo `/etc/default/xendomains`, deve-se desabilitar o parâmetro XENDOMAINS_RESTORE e comentar o parâmetro XENDOMAINS_SAVE:

```
#XENDOMAINS_SAVE=/var/lib/xen/save
XENDOMAINS_RESTORE=false
```

Finalmente, deve-se obter os arquivos no repositório OFELIA da UFSCar ($SITE) [27] e armazená-los conforme descrito a seguir.

```
wget $SITE/network-bridge-vlan -o \
/etc/xen/scripts/network-bridge-vlan
wget $SITE/network-multi-bridge-vlan -o \
/etc/xen/scripts/network-multi-bridge-vlan
chmod +x /etc/xen/scripts/*
wget http://www2.comp.ufscar.br/ ricardofg/OFELIA/xend -o /etc/init.d/xend
```

Como os *scripts* do Xen não configuram VLANs devidamente, é preciso configurar nomes e endereçamento de interfaces na função `configure_vlans()` do arquivo `/etc/init.d/xend`. As configurações definidas nesse arquivo sobrescrevem as presentes no arquivo `/etc/network/interfaces`.

3.7.2.7. Obtendo o OFELIA — Máquinas Agentes

A instalação do OFELIA nos agentes é realizada a partir do mesmo *tarball* existente na página do projeto no Codebasin. Descompacta-se o *tarball* da mesma forma mencionada anteriormente. Realiza-se, então, o seguinte procedimento:

```
mkdir -p /opt/OFELIA/oxa
ln -s /opt/OFELIA /opt/OFELIA/oxa/repository
```

3.7.2.8. Criação de uma VM Template — Máquinas Agentes

Como o *template default* do OFELIA não está disponível publicamente no momento, é necessário criar um template para posterior instanciação. Sugere-se seguir o roteiro abaixo para gerar um template com ferramentas básicas de desenvolvimento e de rede.

[27] `http://www2.comp.ufscar.br/~ricardofg/OFELIA`

```
cd /etc/xen-tools/role.d
wget $SITE
wget $SITE/xen/OFELIA-helper.sh
chmod +x OFELIA*

xen-create-image -passwd -role=OFELIA -install-method=debootstrap -dist=squeeze -ip=172.16.1.99
-netmask=255.255.255.0 -hostname=fibre-default

mkdir /home/vm-images
mkdir /mnt/xen
mount -o loop /home/xen/domains/algum_hostname/disk.img /mnt/xen
cd /mnt/xen
tar pcfz /home/vm-images/default.tar.gz *
umount /mnt/xen
cd /home/vm-images/
md5sum default.tar.gz » default.hash
```

Terminada a geração do *template*, deve-se editar o arquivo `/opt/OFELIA/vt_manager /src/ python/ agent/ tools/versions/default/install/lib/template` , de modo que o *script* de instalação obtenha com sucesso os arquivos necessários. Uma opção consiste em disponibilizar os arquivos da pasta `/home/vm-images` em um servidor web e editar as URLs correspondentes.

3.7.2.9. Instalação do OFELIA - Máquinas Agentes

Novamente, a instalação é realizada com a execução do *script* `ofver`. Segue abaixo os passos para a instalação do agente de virtualização:

```
cd /opt/OFELIA/oxa/repository/vt_manager/src/python/agent/tools
./ofver install
```

O agente é configurado automaticamente para iniciar durante o processo de inicialização do sistema. A interface XMLRPC do agente utiliza a porta 9229.

3.7.3. Configurando e administrando o OFELIA CF

Nesta subseção demonstramos como administrar o OFELIA CF, desde a criação de recursos até a autorização e alocação dos flowspaces OF. Todas essas configurações se fazem necessárias no Expedient, VT Manager e OptIn para que seja possível realizar um experimento.

3.7.3.1. Acessando as interfaces do OFELIA

Durante a instalação do OFELIA são pedidas informações para a criação de um usuário administrador e senha para as três interfaces WEB do OFELIA: Expedient, OptIn e VTManager.

A tela inicial de cada uma das interfaces quando não se tem usuário logado, é a tela de *login*. Para acesso como administrador, basta preencher os campos *username* e *password* com os definidos na ocasião da instalação do OFELIA. A interface inicial após o *login* é chamada DashBoard.

3.7.3.2. Expedient — Visão de administrador

O Expedient faz a administração do FlowVisor, um *proxy* de controladores para o OpenFlow. O FlowVisor é gerenciado a partir de *Flowspaces*, que são basicamente tipos de

fluxo de rede, que podem ser diferenciados por diversos campos de cabeçalho de protocolo, desde o nível 2 até o nível 4. Portanto, chama-se de *flowspaces* quaisquer filtragens que possa se desejar fazer. Esses *flowspaces* determinarão o tráfego que será redirecionado para o controlador do experimento. Resumindo, *FlowSpace* é o nome que se dá ao fluxo definido por uma regra no *flowvisor*.

Com um usuário logado no Expedient estarão disponíveis as seguintes opções para o administrador na aba Dashboard.

- *Message & Logs*: São exibidas as últimas 3 mensagens de log do sistema, relativas a ações como uma máquina virtual iniciada como sucesso. A opção *Go to Message Center* leva a uma interface que exibe todas as mensagens de logs.

- *Projects*: Aqui são exibidos todos os projetos alocados em uma ilha (termo usado para designar uma *testbed* OFELIA). As informações exibidas são o nome do projeto, quantos usuários são gerentes do projeto, quantos membros, quais fatias foram alocadas, e ações que podem ser tomadas no projeto, como excluir o projeto.

- *Island Aggregate Managers*: Mostra os agregados (recursos) atuais do sistema. A *combo box Aggregate Type* permite escolher o tipo de agregado a se criar, que pode ser um de dois tipos, *OpenFlow Aggregate* ou *Virtualization Aggregate*. Escolhido o mesmo, clicar no botão *Add Aggregate*. Mostraremos mais detalhes sobre como criar e editar agregados em seguida.

- *Permission Management*: Serve para aceitar ou negar pedidos de permissão dos usuários em geral.

Em outra área do sistema (*Account*) são exibidas informações de um usuário em específico do sistema. O administrador pode alterar informações relativas à sua conta como primeiro nome, último nome, e-mail, senha, afiliação, entre outras. Por outro lado, a aba *User administration* permite ao administrador monitorar e controlar todos os usuários presentes no sistema, podendo adicionar ou remover usuários do sistema. Para essa aba temos duas seções importantes (*Current users* e *Create new user*), uma tabela que lista os usuários cadastrados no Expedient e um formulário no qual digitamos as informações do novo usuário a ser cadastrado. O botão *Create user* valida os campos presentes no formulário.

Em outra área do sistema (*Aggregate*) é possível adicionar agregados ao Expedient para posterior uso. Existem dois tipos, agregados OpenFlow e agregados de Virtualização. Para o agregado OpenFlow é preciso marcar dados como nome, localização, disponibilidade, e os parâmetros para a gerência desse agregado via OptIn (login, senha do Optin, url do servidor em formato xmlrpc). Os campos para a criação de agregados de virtualização usam campos semelhantes.

Após a criação de agregados e contas de usuário, o gerenciamento acontece pela interface web, sendo que cada sistema web (expedient, optin, vtmanager) cuida de seus respectivos tipos de recursos. Assim sendo, no VT Manager será possível adicionar uma máquina virtual através do comando *Add Server*. Para esses recursos é preciso especificar tipo de distribuição Linux, hypervisor, URL xmlrpc da máquina Agente, parâmetros de rede, como a interface eth1.999 que é a subrede de controle configurada com o Xen. Na

parte de *Data Bridges* faz-se o cadastro das demais interfaces virtuais a serem usadas na rede de experimentação. Por exemplo, colocar a porta virtual eth2 conectada com o *switch* (DATAPATH) permitindo a conexão daquela máquina virtual com um dos *switches* OpenFlow da rede. Depois, ainda na *dashboard*, será preciso cadastrar uma faixa de IPs e faixa de MACs a serem usados pelas VMs.

O próximo passo é a configuração do Optin. Neste caso será preciso informar as configurações do FlowVisor, como login, senha, a URL xmlrpc de onde foi definida a instalação do FlowVisor. E depois informar quais usuários poderão gerenciar esse recurso do FlowVisor, o que é chamado de *Set ClearingHouse*.

A parte final consiste em configurar o Expedient para adicionar os agregados conforme comentado acima. E posteriormente solicitar de maneira *off-line* uma requisição (*flowspace request*) de isolamento do controle através de *flowspaces*, por exemplo, definindo rótulos de VLAN a ser liberada pelo administrador. Uma vez atendido o pedido de *flowspace*. De volta ao *dashboard* do Expedient, na parte *Projects* é possível criar um projeto, criar uma fatia, e definir o tempo de expiração desta fatia. Uma vez criada a fatia (*Slice* na interface), basta o usuário entrar em *Details*, e clicar na região *Aggregates*, de modo a adicionar agregados à sua fatia. Sendo pelo menos 1 agregado OpenFlow (onde será configurado o pedido de FlowSpaces) e 1 ou mais agregados de virtualização.

3.7.4. Executando um experimento no OFELIA CF

Para que um pesquisador interessado possa utilizar a infraestrutura supervisionada pelo OFELIA, são necessários alguns passos preparatórios antes de efetivamente realizar experimentos e testes. Além da criação de uma conta de usuário (via web), é preciso criar uma fatia em que o experimento será montado. Nesta fatia são instanciadas as máquinas que serão necessárias para o experimento. Na fatia será feita uma requisição de um ou mais *flowspaces* (filtragens para isolamento dos experimentos). Relembrando que esses *flowspaces* determinarão que o tráfego de controle será redirecionado para o controlador do experimento. Por fim, dá-se início ao controlador do experimento, configura-se na interface web do OFELIA o endereço do mesmo (tipicamente em uma VM da própria fatia) e realiza-se o experimento, utilizando as máquinas virtuais que foram criadas previamente.

3.7.4.1. Exemplo

Um detalhamento deste procedimento será realizado de maneira mais ilustrativa com um exemplo. Neste será abordada a instanciação de 2 hosts e um teste de `ping` entre eles, passando por um *switch OpenFlow*. Esse *switch* estará sendo controlado por uma instância do `pyswitch.py` (executado com o NOX) em uma terceira VM da fatia. Para isso, seguiremos os seguintes passos:

- Acessar o endereço da interface web, fornecendo os dados: nome, email, senha, organização, ilha (do OFELIA) e uma chave pública (que pode ser gerada com o comando `ssh-keygen -t dsa`. Uma vez criada, um e-mail será enviado confirmando a autorização da criação da conta no email fornecido durante o cadastro.
- Uma vez autorizada a conta, loga-se na ilha em que se registrou. Em seguida, ao

navegar pela interface web, a primeira coisa a se fazer é instanciar uma fatia para o experimento. Ao clicar no botão *create slice*, serão requisitados: nome da fatia, resumo do experimento e data de expiração do experimento.

- Em seguida, criam-se as máquinas virtuais. Escolhe-se o servidor em que se deseja criar a VM, e, em seguida, fornece-se nome da VM, quantidade de memória, imagem base e tipo de virtualização.

- Para preparar as VMs para o experimento, faz-se acesso individual através do `ssh`. Na interface web é possível visualizar o IP de cada máquina e também a senha de *root* para quaisquer necessidades de instalação e configuração do sistema. Neste caso particular, criam-se 3 VMs, todas com 256M de memória, SO debian, de nomes Host1, Host2 e *Controller*. Host1 e Host2 são as máquinas usadas no teste de `ping`, e o *Controller* é a VM em que se instanciará o controlador.

- Dando sequência, acessando a opção *Book OpenFlow and Planetlab Resources*, configura-se o *Flowspace* do experimento. Nessa tela, visualiza-se a infra-estrutura disponível, e seleciona-se os caminhos que deseja-se ter controle. Neste caso, um *switch OpenFlow* e as portas que o conectam a cada um dos servidores envolvidos. Ao avançar para a segunda tela, montam-se as regras para definir o *Flowspace*, preenchendo-se campos como MAC origem, MAC destino, porta, etc. Também se seleciona, para cada regra, quais os caminhos que essa regra cobrirá, dentre os selecionados previamente para o experimento. Neste caso, não se criou nenhuma regra específica, considerando que deseja-se todo o tráfego no caminho especificado.

- Ao ser autorizado pelo administrador, o *Flowspace* aparecerá como *Granted Flowspace* na interface, e terá uma identificação de VLAN. Para que o OFELIA faça corretamente a separação do tráfego do experimento, é necessário configurar todas as interfaces das VMs envolvidas no experimento com essa VLAN. Para tanto, faz-se `ssh` nas VMs, e, supondo que a VLAN seja a 15, configura-se da sequinte maneira.

Configuração de cada uma das VMs:

```
ifconfig eth1 up
ifconfig eth2 up
modprobe 8021q
vconfig add eth1 15
ifconfig eth1 0.0.0.0
ifconfig eth1.15 <endereço IP do experimento> up
```

Dessa maneira, o tráfego do experimento será todo na VLAN 15, e o OFELIA o filtrará de acordo. Para este experimento, configura-se as interfaces na mesma subrede, para que o `ping` seja bem sucedido entre os hosts. Utilizou-se 192.168.1.1/24 no Host1 e 192.168.1.2/24 no Host2. Faz-se o `ping` do Host1 ao Host2 e vice-versa e verifica-se que ainda não há conectividade entre ambos. Para que isso aconteça, acessa-se via `ssh` o *Controller*, e instancia-se o controlador do experimento, que, neste exemplo, é feito utilizando o NOX para rodar a aplicação `pyswitch.py` (ambos existentes na instalação do OpenFlow 1.0), dando o seguinte comando no *Controller*:

```
nox_core pyswitch.py ptcp:6633
```

Para que o *Controller* seja o controlador do experimento, define-se na interface web, no botão *Set Slice Controller* o endereço e a porta do controlador, como sendo o

endereço do *Controller* (a que se dá o `ssh`) e a porta 6633 (que é a porta em que este foi instanciado). Ao aplicar esta configuração, tenta-se novamente o `ping` entre o Host1 e o Host2, que desta vez será bem-sucedido.

3.8. Estudo de Caso de Implantação de OMF em uma Testbed Sem Fio

Nesta seção, apresentaremos a instalação, configuração e uso do OMF em uma pequena *testbed* sem fio. Forneceremos informações suficientes para que uma implantação similar possa ser replicada, assim como referências que auxiliam na resolução de eventuais problemas. Além disso, ilustraremos o uso do OMF através de dois exemplos básicos que empregam recursos desse arcabouço para configuração do ambiente, execução do experimento e coleta de resultados.

3.8.1. Requisitos e infraestrutura

O OMF possui três componentes críticos: gerenciador de agregado (AM – *aggregate manager*), controlador de experimentos (EC – *experiment controller*) e controlador de recurso (RC – *resource controller*). Enquanto o RC deve ser implantado em cada dispositivo que estará disponível para o experimentador, o AM e o EC fazem parte da infraestrutura de suporte aos experimentos. De fato, o EC pode ser implantado fora da infraestrutura de experimentação, por exemplo, no sistema de um experimentador remoto. No entanto, há questões relacionadas a acesso e segurança nesse tipo de configuração que não serão abordados nesse curso. Utilizaremos uma configuração mais simples, na qual AM e EC estão em um único equipamento.

O OMF possui também um recurso para medições, filtragem e coleta de resultados conhecido como OML, o qual está dividido em três partes: servidor, cliente e *proxy* (opcional). A parte cliente está disponível através de uma biblioteca que fornece uma API para que aplicações distribuídas sejam desenvolvidas ou modificadas para serem capazes de armazenar as medições em um servidor OML. Posteriormente, apresentaremos um exemplo que utiliza uma aplicação instrumentada com OML. Embora o OML seja considerado um recurso opcional do OMF, ter um serviço de medições é muito importante em *testbeds* e, portanto, vamos descrever também a instalação e configuração do servidor OML. Novamente, optaremos por uma configuração mais simples, na qual o servidor OML é instalado no mesmo equipamento que o AM. Nesta seção, chamaremos de *servidor* o equipamento no qual serão instalados AM, EC e servidor OML.

Apesar de o OMF ser escrito na linguagem Ruby, não é suficiente apenas ter um interpretador dessa linguagem para executá-lo. O OMF faz uso de um conjunto de serviços que precisam estar disponíveis e devidamente configurados para sua correta execução. Apresentaremos a instalação e configuração mínima dos serviços necessários ao OMF.

Para a instalação do AM, do EC e do servidor OML, recomendamos que seja usado um computador de arquitetura x86 com sistema operacional Linux instalado. Assumindo que todos os serviços relacionados ao OMF também serão instalados nesse equipamento, é recomendado um computador com processador equivalente a Intel i3 ($\sim$3 GHz), com 2 GB de RAM e 50 GB livres de disco. É interessante ter duas interfaces de rede no equipamento, sendo uma para acesso à Internet e outra para controle e monitoramento da infraestrutura de testes. Qualquer distribuição pode ser usada, mas nesse material des-

creveremos os procedimentos específicos para instalação no Ubuntu 11.10 “oneiric”, pois esse sistema possui pacotes para todos os serviços necessários, inclusive o próprio OMF na versão 5.4 (a mais recente durante a elaboração desse material). Utilizamos a documentação dos desenvolvedores do OMF [OMF 2012d] como base para a implantação que descrevemos nesta seção.

O controlador de recurso (RC) pode ser instalado em qualquer equipamento que possua um interpretador Ruby, permitindo que vários tipos diferentes de equipamentos possam ser utilizados como nós de experimentação em uma *testbed*. Nesta seção, vamos descrever a instalação em um pequeno computador com a seguinte configuração: placa-mãe mini-ITX, processador VIA C7 de 1,8 GHz, 1 GB de RAM, HD de 320 GB com 2,5 polegadas (usado em *laptops*), duas placas de rede Ethernet e duas interfaces sem fio IEEE 802.11 b/g (conectadas através de USB). A disponibilidade de duas interfaces de rede Ethernet é útil para separar o tráfego de dados e de controle. É possível utilizar nós de experimentação mais simples e de custo mais baixo, como os baseados em SBCs (*Single Board Computers*) [OMF 2012b] ou em pontos de acesso [Stasi et al. 2009]. No entanto, o uso de equipamentos com mais recursos permite a realização de experimentos mais sofisticados e a exploração de mais recursos disponíveis no OMF, como a possibilidade de escolher diferentes sistemas operacionais para serem carregados nos nós de experimentação. Um recurso opcional, porém muito útil em uma *testbed* para uso remoto, é o controle remoto de energia. Há diferentes maneiras de implementar esse recurso, conforme descrito em [OMF 2012e, NITLab 2012].

3.8.2. Configuração da Rede

No servidor, sugerimos a desativação ou remoção de software para autoconfiguração da rede como *NetworkManager* e *Avahi*. Na distribuição Linux sugerida (Ubuntu 11.10), o *NetworkManager* é instalado por padrão e pode ser removido com o seguinte comando:

```
apt-get remove network-manager
```

Na configuração que utilizamos, uma das interfaces de rede (`eth0`) foi conectada à Internet, enquanto a outra foi usada para controlar e monitorar os nós de teste. A seguir, apresentamos o conteúdo do arquivo `/etc/network/interfaces` para configuração da interface `eth0` de maneira semelhante a de uma estação convencional (que obtém seu endereço IP através de DHCP) e da interface `eth1` para gerência do ambiente de testes (e, portanto, com endereço IP fixo):

```
auto eth0
iface eth0 inet dhcp
auto eth1
iface eth1 inet static
address 10.0.0.200
netmask 255.255.255.0
```

Após a configuração, é importante reiniciar as interfaces (`ifdown ethX; ifup ethX`) e verificar se elas estão com os endereços IP corretos (`ifconfig ethX`) e se há conectividade (`ping` ou outra ferramenta).

Nesta seção, usaremos a rede `10.0.0.0/8` para o ambiente de testes, no entanto, é importante verificar se essa faixa de endereços não é a utilizada na rede de produção (onde a interface `eth0` foi conectada).

Por fim, é importante verificar se os serviços de *shell* remoto seguro (SSH) e

sincronização de tempo (NTP) estão instalados. No Ubuntu 11.10, não é necessária configuração adicional após a instalação básica que pode ser realizada da seguinte maneira:

```
apt-get install ssh ntp
```

3.8.3. Serviços

Apresentamos a seguir, os principais serviços utilizados pelo OMF e uma breve descrição do uso desses serviços no contexto da infraestrutura de experimentação.

- Configuração automática de endereços IP – é utilizada para configurar a interface de controle dos nós de teste. Usaremos o software `dnsmasq` para implementar esse serviço.
- Tradução de nomes – permite que nomes, ao invés de endereços IP, sejam usados para identificar os servidores (AM e OML) e os próprios nós de experimentação. Usaremos o software `dnsmasq` para implementar esse serviço.
- Inicialização remota – auxilia na customização completa de um nó, permitindo que até o sistema operacional seja substituído na inicialização do equipamento. Usaremos o software `dnsmasq` em conjunto com `Syslinux` para implementar esse serviço.
- *Middleware* orientado a mensagens – constitui o núcleo do OMF, provendo a comunicação entre AM, EC, RC e OML para fornecer as funcionalidade de controle e monitoramento dos experimentos. Usaremos o software `Openfire` para implementar esse serviço.
- Base de dados – armazena o inventário dos nós, usuários, resultados coletados e demais informações para operação e uso do *testbed*. Usaremos o software `MySQL` para implementar esse serviço. O servidor OML utiliza o `SQLite`, porém não é necessária nenhuma configuração adicional após a instalação básica.

As subseções a seguir, apresentam detalhes sobre a instalação e configuraçõ do software utilizado pelo OMF, assim como de seus componentes: AM, EC e OML. Conforme comentamos anteriormente, o RC pode ser instalado em um sistema operacional com um interpretador Ruby para transformar o equipamento em um nó de experimentção. Neste material, apresentamos o uso de uma imagem do sistema operacional Linux com o RC previamente instalado, o qual pode ser implantado em um nó usando a própria infraestrutura do OMF. Detalhes sobre a instalação do RC podem ser encontrados em [OMF 2012d].

3.8.3.1. Software dnsmasq e Syslinux

Além do `dnsmasq`, recomendamos a instalação do software `Syslinux` no mesmo passo, pois o serviço de inicialização (ou *boot*) remota para um sistema Linux depende de ambos. No Ubuntu 11.10, a instalação pode ser realizada da seguinte forma:

```
apt-get install syslinux dnsmasq
```

A primeira parte da configuração é realizada no arquivo `/etc/dnsmasq.conf`, o qual deve ter as seguintes linhas adicionadas ao final:

```
interface=eth1
dhcp-range=10.0.0.201,10.0.0.254,255.255.255.0,12h
dhcp-option=3
dhcp-option=option:ntp-server,10.0.0.200
dhcp-boot=net:control,pxelinux.0
enable-tftp
tftp-root=/tftpboot
```

Basicamente, o arquivo acima define a seguinte configuração: 1) interface na qual o serviço estará disponível (`eth1`); 2) não há roteador padrão para os nós; 3) faixa de endereços IP usados de maneira dinâmica (`10.0.0.201-10.0.0.254`); 4) servidor para sincronização de tempo (`10.0.0.200`); 5) nome do arquivo (`pxelinux.0`) para inicialização remota, mas apenas se o rótulo `control` estiver configurado; 6) ativação do suporte ‘a transferência de arquivo de inicialização através de TFTP (*Trivial* FTP); 7) diretório raiz para o TFTP. Nessa configuração, os endereços IP de 10.0.0.1 a 10.0.0.199 estão disponíveis para configuração automática das interfaces de controle dos nós.

A configuração automática das interfaces de controle utiliza o endereço MAC de cada uma para associar sempre o mesmo IP e nome a cada nó, facilitando a identificação dos equipamentos. A seguir é apresentado o conteúdo de um arquivo exemplo (`/etc/dnsmasq.d/mytestbed.conf`) para três nós hipotéticos:

```
dhcp-host=net:control,00:01:02:03:04:05,node1,10.0.0.1
dhcp-host=net:control,00:06:07:08:09:10,node2,10.0.0.2
dhcp-host=net:control,00:AA:BB:CC:DD:EE,node3,10.0.0.3
```

Após a criação e preenchimento adequado do arquivo, é necessário reiniciar o software `dnsmasq`:

```
/etc/init.d/dnsmasq restart
```

Por padrão, o software `dnsmasq` utiliza o arquivo `/etc/hosts` como a base de dados de nomes do domínio. Portanto, é importante adicionar entradas para os nós e servidores que estarão em operação na *testbed*.

O restante desta seção diz respeito ao software `Syslinux`, o qual permite inicializar um sistema operacional Linux através da rede. De fato, o `Syslinux` é um carregador de inicialização (*boot loader*) capaz de lidar com diferentes fontes de inicialização, como um sistema de arquivos FAT, um disco de CD-ROM e a rede. A inicialização pela rede utiliza a especificação da Intel chamada *Preboot Execution Environment* ou PXE.

Inicialmente, é necessário criar os diretórios que conterão os arquivos de configuração e inicialização usados para o *boot* remoto PXE, por exemplo:

```
mkdir -p /tftpboot/pxelinux.cfg
```

Em seguida, é necessário copiar (ou referenciar) o carregador de inicialização:

```
ln -s /usr/lib/syslinux/pxelinux.0 /tftpboot/
```

O diretório `/tftpboot` precisa receber dois arquivos: o núcleo (*kernel*) do Linux e o seu respectivo disco de RAM inicial (*initial* RAM *disk* ou `initrd`) para montagem do sistema de arquivo raiz. Os desenvolvedores do OMF oferecem ambos os arquivos para inicialização de nós x86 convencionais no *site* do projeto [OMF 2012f].

Agora, é necessário criar o arquivo `/tftpboot/pxelinux.cfg/pxeconfig` com o seguinte conteúdo:

```
SERIAL 0 19200 0
DEFAULT linux
LABEL linux
KERNEL linux-omf-pxe-3.0.4
APPEND console=tty0 console=ttyS0,19200n8 vga=normal quiet
            root=/dev/ram0 rw load_ramdisk=1 prompt_ramdisk=1
            ramdisk_size=32768 initrd=initramfs-omf-pxe-5.4.bz2
            control=eth0 xmpp=srv.fibre.ufg.br slice=pxe_slice
            hrn=omf.fibre.%hostname%
PROMPT 0
```

É importante ter cuidado para que em cada linha não haja quebras, inclusive na que se inicia com `APPEND` e que possui um grande número de parâmetros. Os arquivos indicados em `KERNEL` e `initrd` devem ser os que foram obtidos anteriormente.

Seguem algumas considerações adicionais sobre a linha dos parâmetros a serem passados para o núcleo (linha iniciada por `APPEND`). O parâmetro `control` deve ser configurado para a interface Ethernet escolhida para controle, sendo necessariamente a `eth0` se houver apenas uma placa de rede (Ethernet). Na configuração que estamos apresentando, o parâmetro `xmpp` deve referenciar o nome do servidor onde o AM está instalado (por exemplo, `srv.fibre.ufg.br`). Como o `dnsmasq` está sendo usado, a tradução desse nome deve estar no arquivo `/etc/hosts`. O importante com relação ao parâmetro `hrn` (*human readable name*) é a uniformidade ao longo de toda a configuração. Em nossa configuração, os nomes HRN definidos para os nós são `omf.fibre.node1`, `omf.fibre.node2` e `omf.fibre.node3`. Por essa razão, o parâmetro `hrn` recebeu o valor `omf.fibre.%hostname%`.

Um arquivo (`/tftpboot/pxelinux.cfg/default`) deve ser criado para indicar a configuração padrão para inicialização, tendo o seguinte conteúdo:

```
DEFAULT harddrive
LABEL harddrive
localboot 0
```

Dentro do diretório `/tftpboot/pxelinux.cfg/`, deve ser criado um *link* simbólico para cada nó com base no endereço MAC de sua interface de controle. O nome do *link* simbólico deve ser igual ao endereço MAC precedido de `01` com cada octeto separado por hífen. Por exemplo, para os nós apresentados previamente, teríamos os seguintes *links* simbólicos:

```
ln -s pxeconfig 01-00-01-02-03-04-05; ln -s pxeconfig 01-00-06-07-08-09-10; ln -s pxeconfig 01-00-AA-BB-CC-DD-EE
```

Nesse ponto, é possível inicializar os nós de testes a partir do servidor. É importante configurar cada nó para inicialização remota através da rede, deixando o *boot* pelo disco rígido local como segunda opção. Se não houver erros, os nós receberão os endereços IP configurados anteriormente e estarão acessíveis através de *shell* remoto (`telnet`). O sistema inicializado através da rede é reduzido, possuindo apenas a conta de superusuário (`root`), algumas ferramentas básicas e o RC, o qual permite que uma imagem completa de um SO seja trazida e aplicada ao disco rígido do nó. Durante a inicialização remota, as mensagens de interação entre um nó e o servidor são reportadas no arquivo `/var/log/syslog` (no servidor), podendo ser útil analisá-lo em caso de erros.

Caso a inicialização remota dos nós seja bem sucedida, os *links* simbólicos para o arquivo `pxeconfig` devem ser removidos. A princípio, esse procedimento faz os nós inicializarem por seus discos rígidos locais. No entanto, após a sua instalação completa, o OMF criará e removerá *links* simbólicos para os nós sob a orientação (implícita) do experimentador, permitindo alterar o dispositivo de *boot* (rede ou disco rígido).

3.8.3.2. Software Openfire

O software *Openfire* é uma implementação de um servidor XMPP (*eXtensible Messaging and Presence Protocol*). XMPP é um protocolo aberto baseado em XML (*eXtensible Markup Language*) para *middleware* orientado a mensagens. Conforme comentamos anteriormente, o OMF faz uso intensivo desse protocolo e, portanto, AM, EC e RC precisam ter acesso ao servidor XMPP. Novamente, utilizamos a abordagem mais simples e apresentaremos a implantação do *Openfire* no mesmo servidor do AM e do EC.

Os desenvolvedores do OMF recomendam que para sua versão 5.4 sejam instalados o `Openfire 3.7.1` e o `SUN Java 6`. Os passos a seguir ilustram como esse procedimento pode ser realizado no Ubuntu 11.10:

```
apt-get install python-software-properties; add-apt-repository ppa:ferramroberto/java; apt-get update; apt-get
install sun-java6-jre; wget http://www.igniterealtime.org/downloadServlet?filename=openfire/openfire_3.7.1_all.deb
-O openfire_3.7.1_all.deb; dpkg -i openfire_3.7.1_all.deb
```

Caso exista alguma filtragem de pacotes no servidor (por uma *firewall*), as seguintes portas TCP devem ser abertas: 5222, 5269 e 9090. A instalação do *Openfire* já deve iniciá-lo, porém pode levar algum tempo. Logo, verifique se o processo (`.../bin/java -server -Dopenfire...`) está executando antes de tentar configurá-lo.

A configuração do *Openfire* é feita através de uma interface *Web* que pode ser acessada através da porta 9090 do servidor. A primeira vez que o servidor é acessado, o usuário é conduzido por um auxiliar de configuração e deve fornecer as informações que seguem: 1) Idioma. 2) Configurações do servidor. Não há necessidade de alterar as portas e usamos o nome do servidor (`srv.fibre.ufg.br`) como o valor para `Domínio`. 3) Configurações do banco de dados. Recomendamos o uso da opção `Banco de dados interno`. 4) Configurações de perfil. A opção `Padrão` é suficiente. 5) Senha do administrador. Após esse passo, o `Openfire` está configurado para uso pelo OMF.

3.8.3.3. Instalação e Configuração do AM e do servidor OML

Como os desenvolvedores do OMF fornecem os pacotes para o Ubuntu 11.10 “oneiric”, podemos incluir o *site* de pacotes do projeto OMF na lista de repositórios do nosso servidor (`/etc/apt/sources.list`):

```
deb http://pkg.mytestbed.net/ubuntu oneiric/
```

Agora, podemos atualizar a lista de repositórios e instalar o AM e o servidor OML:

```
apt-get update; apt-get install omf-aggmgr-5.4 oml2-server
```

A seguir são apresentados os passos para a primeira parte da configuração do AM.

1. Copiar `/usr/share/doc/omf-aggmgr-5.4/examples/omf-aggmgr.yaml` para `/etc/omf-aggmgr-5.4/`

2. Alterar o arquivo `/etc/omf-aggmgr-5.4/omf-aggmgr.yaml` para indicar o nome correto do servidor XMPP (`srv.fibre.ufg.br`).

3. Ativar os serviços do OMF através da criação dos seguintes *links* simbólicos:

```
cd /etc/omf-aggmgr-5.4/enabled; ln -s ../available/cmcStub.yaml; ln -s ../available/frisbee.yaml; ln
-s ../available/pxe.yaml; ln -s ../available/inventory.yaml; ln -s ../available/result.yaml; ln -s
../available/saveimage.yaml
```

4. Por fim, dentro do diretório `/etc/omf-aggmgr-5.4/available/`, é preciso editar os arquivos `saveimage.yaml` e `frisbee.yaml` para alterar as interfaces para salvar imagens de sistemas operacionais (`saveimage`) e *multicast*, respectivamente. Em nossa configuração, ambas devem receber o endereço IP da interface de controle do AM, ou seja, `10.0.0.200`.

A segunda parte da configuração do AM diz respeito ao inventário da infraestrutura de experimentação. Inicialmente, instalamos o servidor `MySQL` e a ferramenta de gerência com interface *Web* chamada `phpmyadmin`:

```
apt-get install mysql-server libdb4.6 phpmyadmin
```

Durante a instalação do servidor `MySQL` é pedida uma senha para o administrador do banco de dados. Durante a instalação do `phpmyadmin` é questionado qual deve ser o servidor *Web* utilizado, para o qual indicamos o `apache2`. Ainda durante a instalação do `phpmyadmin`, é necessário decidir se a configuração do banco de dados será feita com o auxílio da ferramenta `dbconfig-common` ou não. A resposta deve ser negativa, pois faremos a configuração manualmente.

Com a instalação e configuração básica pronta, podemos conectar ao servidor `MySQL` (`mysql -u root -p`) e criar uma conta para o OMF da seguinte forma:

```
use mysql;
create user 'omf'@'localhost';
grant all on *.* to 'omf'@'localhost';
set password for 'omf'@'localhost'=password('omf');
quit;
```

Finalmente, podemos efetuar uma conexão ao servidor *MySQL* com o usuário OMF (`mysql -u omf -pomf`) e criar a base de dados do inventário:

```
create database inventory;
quit;
```

O pacote do AM traz um arquivo SQL de exemplo de inventário que contém todas as tabelas e algumas entradas como amostras. Esse arquivo de exemplo pode ser importado da seguinte maneira:

```
cd /usr/share/doc/omf-aggmgr-5.4/examples; zcat inventory.sql.gz | mysql -u omf -pomf inventory
```

Através da interface *Web* fornecida pelo `phpmyadmin` é possível acessar a base de dados do inventário e realizar as alterações necessárias de acordo com as características da *testbed*. A tabela `testbeds` deve ser alterada para refletir o nome da *testbed* que está sendo gerenciada, por exemplo `fibre_ufg`. Para cada nó, as seguintes tabelas também devem ser editadas e modificadas de acordo: `locations`, `motherboards`, `nodes` e `pxeimages`. Em geral, as tabelas da base de dado são autoexplicativas e uma breve descrição (um pouco desatualizada) da estrutura pode ser encontrada em [OMF 2012a].

A configuração do inventário é concluída com a verificação do arquivo `/etc/omf-aggmgr-5.4/available/inventory.yaml` para confirmar se as informações estão de acordo com a configuração do servidor *MySQL*.

A parte final da configuração do AM está relacionada à criação de tópicos de Publicação-Inscrição (*Publique-Assine* ou PubSub) no servidor XMPP para a comunicação entre as entidades do OMF (AM, OML, EC e RC). Em geral, o termo *nó* é mais usado

que o termo *tópico* no contexto XMPP, como pode ser visto em [Millard et al. 2012]. No entanto, nesta seção, vamos preferir o termo *tópico* para evitar confusão com o uso prévio de *nó* cujo o sinônimo é equipamento ou dispositivo.

Para criar a árvore de tópicos PubSub, no nosso ambiente, executamos o comando:

```
omf_create_psnode-5.4 srv.fibre.ufg.br mksys
```

Para carregar imagens de sistemas operacionais nos nós, é necessário criar uma fatia de experimentação (*slice*) com o nome `pxe_slice` e todos os nós devem ser acrescentados a ela. Em nossa configuração, o seguinte comando realiza essa tarefa:

```
omf_create_psnode-5.4 srv.fibre.ufg.br mkslice pxe_slice omf.fibre.node1 omf.fibre.node2 omf.fibre.node3
```

Caso um novo nó seja acrescentado à *testbed*, é necessário repetir o último passo com todos os nós. Esse procedimento não gera duplicatas.

Experimentos também são executados dentro de uma fatia e, portanto, os nós escolhidos para realizar um experimento devem ser previamente adicionados à fatia de experimentação. O comando a seguir ilustra a criação de uma fatia (*slice*) com os nossos nós:

```
omf_create_psnode-5.4 srv.fibre.ufg.br mkslice default_slice omf.fibre.node1 omf.fibre.node2 omf.fibre.node3
```

Esse último passo deve ser realizado para se acrescentar novos nós à fatia de experimentação. A configuração do AM está concluída, sendo suficiente reiniciar o serviço:

```
/etc/init.d/omf-aggmgr-5.4 restart
```

3.8.3.4. Instalação e Configuração do EC

O EC precisa ter acesso ao servidor XMPP e, para a maior parte dos testes, também aos nós de teste. Em nossa configuração, o EC se encontra no mesmo servidor que o AM e servidor OMF. Logo, o *site* de pacotes dos desenvolvedores do OMF já foi adicionado à lista de repositórios. Assim, a instalação do EC se resume ao seguinte comando:

```
apt-get install omf-expctl-5.4
```

Novamente, um arquivo de configuração é fornecido juntamente com o pacote do software e deve ser usado como base:

```
cd /usr/share/doc/omf-expctl-5.4/examples/; zcat omf-expctl.yaml.gz > /etc/omf-expctl-5.4/omf-expctl.yaml
```

O arquivo `/etc/omf-expctl-5.4/omf-expctl.yaml` possui múltiplas seções de configuração, mas apenas a `:default:` precisa ser alterada. Para a nossa configuração, seguem os parâmetros modificados e seus respectivos valores:

```
:domain:  'fibre_ufg'
:omluri:  'tcp:srv.fibre.ufg.br:3003'
:web:
    :host:  '10.0.0.200'
:communicator:
    :xmpp:
        :pubsub_gateway:  'srv.fibre.ufg.br'
:services:
    :uri:  'http://srv.fibre.ufg.br:5054'
```

Concluímos então a instalação e configuração do EC. Seguem alguns passos finais para concluir a configuração do OMF e verificar o funcionamento da *testbed*.

Os desenvolvedores do OMF oferecem uma imagem do sistema operacional Linux com o RC já instalado. Sugerimos obter essa imagem e armazená-la conforme descrito:

```
cd /var/lib/omf-images-5.4; wget http://pkg.mytestbed.net/files/5.4/baseline/baseline.ndz
```

A imagem padrão (`baseline.ndz`) pode ser carregada em todos os nós com o seguinte comando:

```
omf load
```

Se os nós foram inicializados pelo arquivo padrão (`/tftpboot/pxelinux.cfg/default`), provavelmente eles não possuem o RC e, portanto, é necessário reiniciá-los manualmente nesse primeiro uso. Após o *boot*, o OMF providencia a criação dinâmica dos *links* simbólicos e consequentemente da carga de um pequeno sistema operacional com o RC que providencia a aquisição da imagem do sistema operacional e sua escrita no disco rígido. Esse procedimento destrói qualquer sistema de arquivos que esteja previamente no nó. Após a conclusão o nó é reinicializado e os *links* simbólicos removidos, fazendo com que o *boot* PXE indique o disco rígido local de cada equipamento para inicialização. Cada nó inicializada com a imagem que foi previamente gravada.

Os nós estão prontos para testes. Recomendamos a leitura dos tutoriais sobre controle dos recursos da *testbed* com OMF [OMF 2012g], nos quais são apresentadas informações sobre como salvar imagens de SOs, verificar o estado dos nós, ligar/desligar nós, etc.

3.8.4. Testes

O OMF define e utiliza uma linguagem de domínio específico para descrever um experimento. Essa linguagem é chamada OEDL (OMF *Experiment Description Language*) e é baseada na linguagem Ruby. Ou seja, a OEDL estende a linguagem Ruby com comandos para oferecer funcionalidades para manipulação de topologia, agrupamento de recursos, definição aplicações, execução de tarefas, dentre outras. Mais detalhes sobre a linguagem OEDL e seus comandos podem ser encontrados em [OMF 2012c].

Após escrever a descrição de um experimento em OEDL, é possível executá-lo com um comando semelhante ao descritor a seguir:

```
omf exec exemplo.rb
```

A seguir são apresentados dois exemplos de experimentos descritos em OEDL que podem ser executados em uma *testbed* controlada pelo OMF.

3.8.4.1. Exemplo 1

O Código fonte 3.1 apresenta um exemplo muito simples que mostra como executar um comando remotamente em dois nós da *testbed*.

3.8.4.2. Exemplo 2

O Código fonte 3.2 ilustra o uso de uma versão modificada do *Iperf* que é capaz de armazenar seus resultados no servidor OML.

Após a conclusão do experimento, por padrão, é gerado um arquivo com extensão `.sq3` dentro do `/tmp`. Nesse arquivo, encontram-se os dados do experimento organizados em tabelas do banco de dados.

Código fonte 3.1. Exemplo 1

```
#Define o grupo de recursos 'teste'
defGroup('teste',omf.fibre.node2,omf.fibre.node2)
#Quando todos os nos estiverem prontos, execute '/bin/ls -l -a'
onEvent(:ALL_UP_AND_INSTALLED) do |event|
    #Imprime alguma informacao na tela
    puts "Executando o comando '/bin/ls -l -a' em todos os nos..."
    group("teste").exec("/bin/ls -l -a")

    #Aguarda 10 segundos
    wait 10

    #Encerra o experimento
    Experiment.done
end
```

Código fonte 3.2. Exemplo 2

```
#Define o grupo de recursos 'servidor'
defGroup('servidor', "omf.fibre.node1") do |node|
    node.net.e1.ip = "192.168.0.1"

    #Adiciona a aplicacao pre-definida Iperf
    node.addApplication("test:app:iperf") { |app|
        #Configura propriedades da aplicacao -- lado servidor
        app.setProperty('server', true)
        app.setProperty('port', 5000)
    }
end

#Define o grupo de recursos 'cliente'
defGroup('cliente', "omf.fibre.node2") do |node|
    node.net.e1.ip = "192.168.0.2"

    node.addApplication("test:app:iperf") do |app|
        #Configura propriedades da aplicacao -- lado cliente
        app.setProperty('client', '192.168.0.1')
        app.setProperty('port', 5000)
        app.setProperty('time', 30)
        app.setProperty('interval', 1)

        #Utiliza um ponto de coleta pre-definido na aplicacao
        app.measure('TCP_Info', :samples => 1)
    end
end

onEvent(:ALL_UP_AND_INSTALLED) do |event|
    wait 5

    group("servidor").startApplications
    wait 5

    group("cliente").startApplications
    wait 40

    Experiment.done
end
```

3.9. Conclusões

O presente minicurso introduz o estado da arte em arcabouços de controle e monitoramento para ambientes de experimentação em redes e sistemas distribuídos. No material cobrimos desde os requisitos básicos, passando por diversos exemplos de *testbeds* e seus CMFs, discutimos federação e apresentamos dois casos de uso baseados nos CMFs: OFELIA e OMF. A pesquisa em CMFs ainda está na sua infância, com o exemplo apontamos o desenvolvimento em espiral do GENI que ainda está evoluindo, e portanto é difícil prever com precisão quais as tendências futuras. Entretanto podemos, ainda assim, realizar uma tentativa de identificar as áreas mais promissoras em CMFs, como a criação de novas técnicas de virtualização que permitiriam recursos como enlaces sem fio serem compartilhados, que permitam recursos de máquinas virtuais serem mais escaláveis e flexíveis com potencial interação com computação em nuvem (IaaS). Dentro do aspecto de controle é importante o desenvolvimento de uma alocação de recursos mais dinâmica e ágil, e que a reprodutibilidade de experimentos seja padrão nas *testbeds*. Com relação a monitoramento, que seja possível ao usuário ter todo tipo de característica pertinente ao seu experimento para posterior análise. Finalmente, é preciso implantar mais e maiores *testbeds* como a do projeto FIBRE (projeto em conjunto Brasil e Europa) focado em tecnologia OpenFlow e com ilhas heterogênea com características específicas, algumas com enlaces sem fio, outras focadas em redes móveis (como as redes DTN baseadas em pessoas). Todo esse esforço pode gerar novos requisitos para *testbeds* e CMFs no futuro, e até pode precisar atender outros tipos de pesquisadores, interessados em interações humanas em larga escala e na sua influência na disseminação de dados na rede.

Agradecimentos

Os autores gostariam de agradecer às equipes FIBRE-BR da UFSCar, UFPA, UNIFACS, UFG, USP e demais membros do FIBRE-BR. Também agradecemos a Marcial Fernandez (UECE), Jorge Barros (CPqD), Mateus Cerezini (UFES). Este minicurso foi parcialmente suportado com recursos do projeto Projeto FIBRE/CNPq No. 590022/2011-3, FAPESPA e FAPEG Proc. No. 200910267000343.

Referências

[gen 2012] (2012). GENI projects: Software integration. http://groups.geni.net/geni/wiki/SpiralThree#SoftwareIntegration.

[Pri 2012] (2012). Primeproject - public - modeling and networking systems research group. https://www.primessf.net/bin/view/Public/PRIMEProject.

[Andersson and Madsen 2005] Andersson, L. and Madsen, T. (2005). Provider Provisioned Virtual Private Network (VPN) Terminology. RFC 4026 (Informational).

[Bourgeau et al. 2010] Bourgeau, T., Augé, J., and Friedman, T. (2010). TopHat: supporting experiments through measurement infrastructure federation. In *Proceedings of TridentCom'2010*, Berlin, Germany.

[Csabai et al. 2010] Csabai, I., Fekete, A., Hága, P., Hullár, B., Kurucz, G., Laki, S., Mátray, P., Stéger, J., Vattay, G., Espina, F., Garcia-Jimenez, S., Izal, M., Magana,

E., Morató, D., Aracil, J., Gómez, F., Gonzalez, I., López-Buedo, S., Moreno, V., and Ramos, J. (2010). ETOMIC advanced network monitoring system for future internet experimentation. In *Proceedings of TridentCom'2010*, Berlin, Germany.

[Duerig et al. 2012] Duerig, J., Ricci, R., Stoller, L., Strum, M., Wong, G., Carpenter, C., Fei, Z., Griffioen, J., Nasir, H., Reed, J., and Wu, X. (2012). Getting started with GENI: a user tutorial. *SIGCOMM Comput. Commun. Rev.*, 42(1):72–77.

[Dwertmann et al. 2009] Dwertmann, C., Ergin, M. A., Jourjon, G., Ott, M., Rakotoarivelo, T., Seskar, I., and Gruteser, M. (2009). DEMO: Mobile Experiments Made Easy with OMF/Orbit. In *Proceedings of the ACM SIGCOMM 2009 (Demo Session)*.

[Faber and Wroclawski 2008] Faber, T. and Wroclawski, J. (2008). Access control for federation of emulab-based network testbeds. In *Proceedings of the conference on Cyber security experimentation and test*, pages 6:1–6:6. USENIX Association.

[GENI 2009] GENI (2009). GENI control framework requirements. Technical report, GENI. Document ID: GENI-SE-CF-RQ-01.3.

[GENI 2010] GENI (2010). GENI instrumentation and measurement architecture. Technical report, GENI. Document ID: GENI-SE-IM-ARCH-1.0.

[GENI 2012] GENI (2012). Instrumentation tools for a GENI prototype (INSTOOLS). Último acesso em 21/02/12.

[Griffioen] Griffioen, J. From INSTOOLS to GEMINI. GEC12 Slides. 2011.

[Griffioen et al. 2009] Griffioen, J., Fei, Z., and Nasir, H. (2009). Architectural design and specification of the INSTOOLS measurement system. Technical report, Laboratory of Advanced Networking, University of Kentucky.

[Hanemann et al. 2005] Hanemann, A., Boote, J. W., Boyd, E. L., Durand, J., Kudarimoti, L., Lapacz, R., Swany, D. M., Trocha, S., and Zurawski, J. (2005). PerfSONAR: a service oriented architecture for multi-domain network monitoring. In *Proceedings of the Third international conference on Service-Oriented Computing*, ICSOC'05, pages 241–254, Berlin, Heidelberg. Springer-Verlag.

[Kompella and Rekhter 2007] Kompella, K. and Rekhter, Y. (2007). Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling. RFC 4761 (Proposed Standard). Updated by RFC 5462.

[Kopsel and Woesner 2011] Kopsel, A. and Woesner, H. (2011). Ofelia pan-european test facility for openflow experimentation. In *Towards a Service-Based Internet*, volume 6994 of *Lecture Notes in Computer Science*, pages 311–312. Springer Berlin / Heidelberg.

[Magedanz et al. 2008] Magedanz, T., Schreiner, F., and Wahle, S. (2008). From NGN to Future Internet Testbed Management - Collaborative Testbeds as Enabler for Cross-Technology, Cross-Layer, and Cross-Domain Communication and Network Research. *Tele Kommunikation Aktuell*, 62(5-6):20–40. ISSN 1619-2036.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74.

[Millard et al. 2012] Millard, P., Saint-Andre, P., and Meijer, R. (2012). XEP-0060: Publish-Subscribe. `http://xmpp.org/extensions/xep-0060.html`. [Último acesso: 06-Fevereiro-2012].

[Mussman 2012] Mussman, H. (2012). GENI instrumentation and measurement work in progress. Último acesso em 20/02/12.

[NITLab 2012] NITLab (2012). CM cards. `http://nitlab.inf.uth.gr/NITlab/index.php/testbed/hardware/cm-card`. [Último acesso: 20-Janeiro-2012].

[OMF 2012a] OMF (2012a). Description of the OMF Inventory Schema. `http://www.mytestbed.net/projects/omf/wiki/InstallationInventory_`. [Último acesso: 20-Janeiro-2012].

[OMF 2012b] OMF (2012b). Low Cost Node Overview. `http://www.mytestbed.net/projects/lcn/wiki`. [Último acesso: 20-Janeiro-2012].

[OMF 2012c] OMF (2012c). OEDL - The OMF Experiment Description Language. `http://www.mytestbed.net/projects/omf/wiki/The_Experiment_Controller_AP%I`. [Último acesso: 20-Janeiro-2012].

[OMF 2012d] OMF (2012d). OMF 5.4 installation guide. `http://www.mytestbed.net/projects/omf/wiki/Installation_Guide_54`. [Último acesso: 20-Janeiro-2012].

[OMF 2012e] OMF (2012e). OMF System Prerequisites. `http://mytestbed.net/projects/omf/wiki/Prerequisites`. [Último acesso: 20-Janeiro-2012].

[OMF 2012f] OMF (2012f). PXE files. `http://pkg.mytestbed.net/files/5.4/pxe/`. [Último acesso: 20-Janeiro-2012].

[OMF 2012g] OMF (2012g). Tutorials on Controlling Testbed Resources. `http://www.mytestbed.net/projects/omf/wiki/Advanced_Tutorials_for_Exper%imenters`. [Último acesso: 20-Janeiro-2012].

[OneLab2_Executive_Committee_and_guests 2009] OneLab2_Executive_Committee_and_guests (2009). Whitepaper 01 - On Federations... `http://www.onelab.eu/images/PDFs/Whitepapers/onelab2_whitepaper01.pdf`. [Último acesso: 12-Fevereiro-2012].

[Peterson et al. 2003] Peterson, L., Anderson, T., Culler, D., and Roscoe, T. (2003). A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33:59–64.

[Peterson et al. 2010] Peterson, L., Ricci, R., Falk, A., and Chase, J. (2010). *Slice-Based Federation Architecture*.

[Peterson and Roscoe 2006] Peterson, L. and Roscoe, T. (2006). The design principles of planetlab. *SIGOPS Oper. Syst. Rev.*, 40:11–16.

[Peterson et al. 2009a] Peterson, L., Sevinc, S., Baker, S., Mack, T., Moran, R., and Ahmed, F. (2009a). *PlanetLab Implementation of the Slice-Based Facility Architecture*.

[Peterson et al. 2009b] Peterson, L., Sevinc, S., Baker, S., Mack, T., Moran, R., and Ahmed, F. (2009b). *PlanetLab Implementation of the Slice-Based Facility Architecture*.

[Peterson et al. 2009c] Peterson, L., Sevinc, S., Lepreau, J., Ricci, R., Wroclawski, J., Faber, T., Schwab, S., and Baker, S. (2009c). *Slice-Based Facility Architecture*.

[Raychaudhuri et al. 2005] Raychaudhuri, D., Ott, M., and Secker, I. (2005). Orbit radio grid tested for evaluation of next-generation wireless network protocols. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, TRIDENTCOM '05, pages 308–309, Washington, DC, USA. IEEE Computer Society.

[Sanchez et al. 2011] Sanchez, A., Moerman, I., Bouckaert, S., Willkomm, D., Hauer, J., Michailow, N., Fettweis, G., DaSilva, L., Tallon, J., and Pollin, S. (2011). Testbed Federation: An Approach for Experimentation-Driven Research in Cognitive Radios and Cognitive Networking. In *Proceedings of the Future Network Summit*, pages 1–9.

[Singh et al. 2005] Singh, M., Ott, M., Seskar, I., and Kamat, P. (2005). ORBIT measurements framework and library (OML): motivations, implementation and features. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 146 – 152.

[Stasi et al. 2009] Stasi, G. D., Avallone, S., and Canonico, R. (2009). Integration of omf-based testbeds in a global-scale networking facility. In *Quality of Service in Heterogeneous Networks*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 545–555. Springer Berlin Heidelberg.

[Swany and Boyd] Swany, M. and Boyd, E. Leveraging and abstracting measurements with perfSONAR (LAMP). Último acesso em 21/02/12.

[Swany et al. 2011] Swany, M., Small, C., Boyd, E., Griffioen, J., and Fei, Z. (2011). GEMINI: A GENI measurement and instrumentation infrastructure. GEC12 Slides.

[Tronco 2010] Tronco, T. (2010). *New Network Architectures – The Path to the Future Internet*. Springer-Verlag Berlin Heidelberg, Warsaw, Poland, 1st edition.

[Wahle 2008] Wahle, S. (2008). Teagle - The Central Testbed Federation Tool. In *Panlab Seminar - Testbed Federation in Europe*, Dinard, France.

[Wahle et al. 2008] Wahle, S., Gavras, A., Gouveia, F., Hrasnica, H., and Magedanz, T. (2008). Network Domain Federation - Infrastructure for Federated Testbeds. In *2008 NEM Summit - Towards Future Media Internet*, pages 179 – 184, Saint-Malo, France. Eurescom GmbH. ISBN 978-3-00-025978-4.

[White et al. 2002] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36:255–270.

[x667 2004] x667 (2004). *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components*.

[Zink 2011] Zink, M. (2011). GIMI: Large-scale GENI instrumentation and measurement infrastructure. GEC12 Slides.