



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

**Um Sistema Gerenciador de Workflows Científicos
Para a Plataforma de Nuvens Federadas BioNimbuz**

Vinicius de A. Ramos

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora

Prof.^a Dr.^a Aletéia Patrício Favacho de Araújo

Brasília
2016



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Sistema Gerenciador de Workflows Científicos Para a Plataforma de Nuvens Federadas BioNimbuz

Vinicius de A. Ramos

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof.^a Dr.^a Aletéia Patrício Favacho de Araújo (Orientadora)
CIC/UnB

Prof. Dr. ^a Maria Emilia Machado Telles Walter Prof. Dr. ^a Genaina Nunes Rodrigues
CIC/UnB CIC/UnB

Prof. Dr. Andre Drummond
Coordenador do Curso de Engenharia da Computação

Brasília, 12 de fevereiro de 2016

Dedicatória

"À minha família e amigos, que sempre estiveram comigo me apoiando e dando forças."

- Vinícius de Almeida Ramos

Agradecimentos

"Agradeço à ..."

- Vinícius de Almeida Ramos

Resumo

A necessidade por maior poder de processamento e armazenamento, consequência da complexidade das atuais aplicações e sistemas, tem dado espaço para o desenvolvimento de novos paradigmas na Computação. Com isso, criou-se o conceito de Computação em Nuvem. Essa nova forma de se prover serviços computacionais tem possibilitado o desenvolvimento e a criação de diversas aplicações que compartilham diferentes tecnologias e provedores de serviços. Neste cenário, aplicações em Bioinformática tem se beneficiado dessa nova plataforma, devido à exigência de quantidades cada vez maiores de processamento. O BioNimbuz, plataforma de execução de *workflows* em Bioinformática desenvolvida na Universidade de Brasília pelo aluno Hugo Saldanha, utiliza o paradigma de Computação em Nuvem para processar fluxos de aplicações em diferentes provedores de serviços computacionais, como Microsoft Azure e Amazon EC2. Dessa forma, faz-se necessário o gerenciamento da execução desses *workflows* desde sua submissão ao sistema até sua completude, tal como o provimento de uma interface para que o usuário possa ter acesso a esses serviços. Este trabalho propõe melhorias no gerenciamento e controle dos *workflows* submetidos à plataforma BioNimbuz implementando um Sistema Gerenciador de *Workflows* Científicos baseado em tecnologias *web*.

Palavras-chave: Computação em Nuvem, *Workflow* Científico, Sistema Gerenciador de *Workflows* Científicos, Tecnologias *web*

Abstract

The need for greater processing power and storage, caused by the complexity of today's applications and systems, has given space to the development of new paradigms in computing. Thus, it created the concept of Cloud Computing. This new way of providing computing services has enabled the development and creation of various applications that share different technologies and service providers. In this scenario, applications in Bioinformatics has benefited from this new platform due to demand increasing amounts of processing. The BioNimbuz, a Bioinformatics workflow execution platform developed at the University of Brasilia by the student Hugo Saldanha uses the Cloud Computing paradigm to process application flows in different computer services providers, such as Microsoft Azure and Amazon EC2. Thus, it is necessary to manage the execution of these flows (worflows) since its submission to the system until their completion, such as provide an interface for the user to have access to these services. This paper proposes improvements in the management and control of workflows undergoing BioNimbuz platform implementing a Scientific Workflow Management System based on web technologies.

Keywords: Cloud Computing, Scientific Workflow, Scientific Workflow Management System, web technologies

Sumário

1	Introdução	1
1.1	Motivação	2
1.2	Problema	3
1.3	Objetivo	3
1.3.1	Principal	3
1.3.2	Específicos	3
1.4	Organização do Trabalho	4
2	Computação em Nuvem	5
2.1	Sistemas Distribuídos	5
2.1.1	<i>Cluster</i> Computacional	6
2.1.2	<i>Grid</i> Computacional	8
2.2	Nuvem Computacional	9
2.2.1	Tipos de Nuvens	11
2.2.2	Arquitetura	12
2.2.3	Modelos de Serviço	13
2.3	Federação de Nuvens	14
3	Workflow Científico	17
3.1	Conceitos e Definições	17
3.2	Ciclo de Vida de um <i>Workflow</i>	18
3.3	Sistemas Gerenciadores de <i>Workflows</i> Científicos	21
3.4	Exemplos de Sistemas Gerenciadores de <i>Workflows</i> Científicos	22
3.5	Vantagens na Utilização de um Sistema Gerenciador de <i>Workflows</i> Científicos	26
4	BioNimbuz	27
4.1	Principais Características	27
4.1.1	Apache Avro	28
4.1.2	Apache ZooKeeper	29
4.2	Arquitetura do BioNimbuz	31

4.2.1	Camada de interface com o usuário	32
4.2.2	Camada de Núcleo	32
4.2.3	Camada de Infraestrutura	36
5	Sistema Gerenciador de Workflows Científicos para o BioNimbuZ	37
5.1	Do <i>terminal</i> para a Interface <i>Web</i>	37
5.2	Novos Requisitos	39
5.3	Proposta de uma Nova Arquitetura	40
5.4	Aplicação <i>Web</i>	41
5.4.1	Camada de Modelo	42
5.4.2	Camada de Visualização	42
5.4.3	Camada de Controle	43
5.5	Camada de Comunicação	44
5.6	Alterações realizadas no Núcleo do BioNimbuZ	46
6	Resultados Obtidos	48
7	Conclusões e Trabalhos Futuros	52
	Referências	57

Listas de Figuras

2.1	Arquitetura de um <i>Cluster</i>	7
2.2	Arquitetura de um <i>Grid</i> , adaptado de [7].	8
2.3	Arquitetura de uma Nuvem Computacional [15].	13
2.4	Fases da computação em nuvem, segundo White E. <i>et al.</i> [26].	14
3.1	Ciclo de Vida de um <i>Workflow</i>	19
3.2	Arquitetura multicamada do <i>DiscoveryNet</i> (adaptado de [4]).	22
3.3	Ambiente gráfico do Taverna Online (retirado de [16]).	23
3.4	Ambiente gráfico do Triana (retirado de [5]).	24
3.5	A semântica da interação entre os componentes é determinada pelo <i>director</i> , o qual controla a execução (adaptado de [13]).	25
3.6	Interface gráfica do Kepler (retirado de [11]).	25
4.1	Modelo de serviço utilizado no ZooKeeper (adaptado de [59]).	30
4.2	Exemplo de estrutura de nós do ZooKeeper.	31
4.3	Antiga arquitetura do BioNimbuZ.	31
5.1	<i>Terminal</i> do BioNimbuZ.	38
5.2	Nova Arquitetura do BioNimbuZ.	40
5.3	Modelo <i>Model-View-Controller</i>	41
6.1	Tela inicial da aplicação <i>web</i>	48
6.2	Tela de cadastro de novos usuários.	48
6.3	Tela inicial da aplicação <i>web</i>	49
6.4	Tela inicial de criação dos <i>workflows</i>	49
6.5	Tela utilizada para selecionar elementos que irão compôr o <i>workflow</i>	49
6.6	Nesta tela o usuário compõe o fluxo de seu <i>workflow</i>	49
6.7	Nesta tela o usuário define parâmetros de execução de um elemento do <i>workflow</i>	50
6.8	Tela final da composição do <i>workflow</i>	50
6.9	Tela final da composição do <i>workflow</i>	50

6.10 Tela utilizada para realizar o <i>upload</i> de arquivos.	51
6.11 Nesta tela usuários controlam seu armazenamento e verificam sua porcen- tagem.	51

Lista de Tabelas

5.1	Relação entre classes Java e páginas <i>HTML</i>	43
5.2	Relação entre classes Java e páginas <i>HTML</i>	46
6.1	Estados possíveis de um <i>Workflow</i>	51

Capítulo 1

Introdução

A busca por cada vez mais poder de processamento tem desenvolvido novos estudos e paradigmas na Computação. *Grids* computacionais, *clusters* e, mais atualmente, nuvens computacionais têm surgido como alternativa para suprir essa necessidade de maneiras distintas. Em 1969, Leonard Kleinrock, um dos cientistas que chefiou o projeto ARPANET (o qual se tornou a base da Internet) disse que a computação funcionaria a partir de um modelo como é visto atualmente na telefonia e na eletricidade, um serviço. Neste modelo, usuários acessam esses serviços independentemente de onde estão localizados ou de qual maneira são entregues, abstraindo diversas características do ambiente envolvido. Esse modelo se aproxima do conceito de Computação em Nuvem.

Entretanto, não existe na literatura consenso para a definição de computação em nuvem, porém algumas características fundamentais [2] estão presentes na maioria delas, como virtualização, escalabilidade, interoperabilidade, qualidade de serviço (QoS), gerenciamento de falhas, transparência, elasticidade. Esses aspectos fundamentais constituem o modelo de nuvens computacionais como serviços, em que o sistema passa a ilusão ao usuário de que este possui acesso a recursos ilimitados de software e hardware. Nesse contexto, surgiram os tipos de nuvens, os quais são a nuvem privada, pública, híbrida, comunitárias e, mais atualmente, a federação de nuvens [16].

O mais novo destes tipos, a Federação de Nuvens, comprehende o uso de diversos provedores em um único serviço. Isso provê características adicionais aos modelos anteriores de nuvem pública, privada e híbrida, tais como migração de recursos (como imagens de máquinas virtuais), redundância de dados, processamento paralelo, replicação de recursos, combinação de serviços complementares, fragmentação de dados (por exemplo, itens do tipo 1 são armazenados no provedor A enquanto itens do tipo 2 são armazenados no provedor B. Isso se torna útil quando requisitos funcionais e não funcionais diferem para tipos de dados diferentes). Adicionalmente, uma federação de nuvens possibilita o desenvolvimento de sistemas flexíveis e interoperáveis, o que diminui os custos de desenvolvimento

e facilita sua expansão, com o custo de se adicionar complexidade ao sistema.

Neste cenário, a Bioinformática tem se beneficiado com esse conceito de nuvens computacionais pela sua característica de tratar grandes quantidades de dados, produzidas pelas modernas máquinas que executam algoritmos de sequenciamento genômico. Dessa forma, diversas ferramentas foram projetadas e implantadas tirando proveito dos recursos disponibilizados pela computação em nuvens. O BioNimbuz, projeto desenvolvido por Hugo Saldanha [3], faz uso da infraestrutura de uma federação de nuvens para executar *workflows* em Bioinformática de maneira transparente, flexível, eficiente e tolerante à falhas, com acesso a grande poder de processamento e armazenamento.

Na Bioinformática, um *workflow* é um conjunto de diversas fases em que análises computacionais são executadas a partir de dados obtidos por meio de sequenciadores automáticos. Cada pesquisa implica em uma combinação de diferentes ferramentas já existentes ou a serem desenvolvidas, o que adiciona complexidade ao sistema, pois torna-o mutável a cada nova pesquisa. Sistemas científicos que gerenciam *workflow* [4] devem automatizar a execução de *workflows* científicos, suportando usuários na montagem, composição e verificação da execução do workflow gerado pelo usuário.

Este trabalho propõe um sistema gerenciador de *workflows* científicos que trata o problema de manutenção dos *workflows* submetidos ao ambiente de nuvem federada BioNimbuz, além de implementar uma interface baseada em tecnologias *web* para que os usuários possam acessar os serviços disponibilizados pelo BioNimbuz através da Internet.

1.1 Motivação

Com o número crescente de sistemas computacionais utilizados na Bioinformática para execução de *workflows* científicos, percebeu-se a necessidade de criação de sistemas que gerenciem o ciclo de vida destes *workflows*. Este ciclo de vida é composto por cinco fases [4], que são: Criação e Composição, Planejamento de Recursos, Execução, Análise da Execução, Compartilhamento de Resultados.

Diante desse contexto, a implementação gráfica de um sistema de gerenciamento de *workflows* para o ambiente de nuvem federada BioNimbuz acessível pela Internet se faz necessário, uma vez que ainda não existe essa abordagem. Enxergou-se a possibilidade de integração desse sistema à uma interface com tecnologia *web*, que possa garantir o gerenciamento do ciclo de vida dos *workflows* submetidos ao sistema.

1.2 Problema

Atualmente, não existe a concepção de *workflow*, ou fluxo de trabalho, na plataforma BioNimbuZ dado que a execução de uma sequência de passos é realizada de maneira sequencial, isto é, o usuário deve escolher o software computacional, indicar suas entradas e saídas, repetindo esse processo até a completude do fluxo desejado. Dessa forma, não há suporte à criação e gerenciamento de *workflows*, pois o usuário intervém em todos os passos deste fluxo.

1.3 Objetivo

1.3.1 Principal

Propor e implementar um Sistema Gerenciador de *Workflows* Científicos acessível pela Internet, para que o usuário possa: compor um *workflow* de maneira gráfica, enviar arquivos necessários à sua execução, enviá-lo ao núcleo BioNimbuZ para ser executado, salvar seu estado para que o usuário tenha acesso posterior ao término da execução e, por fim, permitir que o usuário visualize o resultado para realizar pós-análise.

1.3.2 Específicos

Para propor e implementar um Sistema Gerenciador de *Workflows* Científicos e cumprir o objetivo principal definido na Seção 1.3.1, este trabalho tem os seguintes objetivos específicos:

- Desenvolver uma aplicação *web* utilizando a linguagem de programação Java para permitir que usuários possam acessar os recursos disponíveis pela plataforma BioNimbuZ;
- Desenvolver uma camada de comunicação utilizando *webservices REST (Representational State Transfer)* para transferir arquivos, recuperar dados, solicitar a execução de *workflows*, entre outras funcionalidades para o núcleo do BioNimbuZ;
- Desenvolver mecanismos que possibilitem aos usuários do sistema desenhar, compor, iniciar a execução e verificar os resultados obtidos a partir da execução desse *workflow*;
- Utilizar Bancos de Dados *MySQL* para prover a persistência dos dados enviados ao BioNimbuZ.

1.4 Organização do Trabalho

Este trabalho foi dividido em mais 6 capítulos assim distribuídos: no Capítulo 2 são apresentados os conceitos básicos para compreensão do contexto atual da Computação em Nuvem, como a concepção de Sistemas Distribuídos, *Clusters* e *Grids* computacionais. Além disso, expõe algumas tecnologias utilizadas em nuvens computacionais e detalha os principais aspectos da Federação de Nuvens.

No Capítulo 3 será abordado o conceito de *workflows* científicos, suas características e ciclo de vida. Também serão apresentados exemplos de Sistemas Gerenciadores de *Workflows* Científicos que nortearam o desenvolvimento do presente trabalho.

O Capítulo 4 detalha a plataforma de nuvens federadas BioNimbuz, retratando sua arquitetura anterior ao desenvolvimento deste trabalho. Também serão descritas tecnologias utilizadas em sua construção e que otimizaram pontos como tolerância à falhas e comunicação entre componentes de sua arquitetura.

O Capítulo 5 apresenta o Sistema Gerenciador de *Workflows* Científicos proposto neste trabalho, com uma nova proposta de arquitetura para o BioNimbuz. Nele serão retratados detalhes de sua implementação, a escolha das tecnologias utilizadas, bem como compara a antiga maneira de se acessar os serviços providos pelo BioNimbuz com o novo método, baseado em tecnologias *web*.

No Capítulo 6 são mostrados os resultados obtidos com o desenvolvimento do Sistema Gerenciador de *Workflows* Científicos para o BioNimbuz. E por fim, no Capítulo 7 são apresentadas as conclusões obtidas e demais trabalhos futuros, que poderão melhorar e otimizar o proposto no presente trabalho.

Capítulo 2

Computação em Nuvem

O objetivo deste capítulo é mostrar os conceitos envolvidos no paradigma de computação distribuída, mais especificamente os de Computação em Nuvem, abordando suas características e suas particularidades. Também descreve as chamadas Federações de Nuvens Computacionais e como este trabalho pretende utilizá-las para integrar um Sistema Gerenciador de *Workflows* Científicos à atual plataforma do BioNimbuZ. Para isso, a Seção 2.1 mostra o histórico da computação distribuída e seus principais elementos (como *Grids* e *Clusters*), a Seção 2.2 apresenta os conceitos básicos que norteiam o entendimento sobre nuvens computacionais, mostrando como diversas pesquisas contribuíram para o conceito atual desta tecnologia. Por fim, a Seção 2.3 mostra os detalhes da chamada Federação de Nuvens Computacionais, quais seus objetivos, suas características e o que a distingue dos outros modelos de computação distribuídas.

2.1 Sistemas Distribuídos

A Internet é utilizada por bilhões de pessoas com vários propósitos diferentes, como ler e-mails, visualizar conteúdo multimídia, fazer compras ou apenas realizar uma busca por um assunto de interesse. Isso passa ao usuário a ilusão de que a informação e o sistema que a provê se encontram localmente em sua máquina. Mas, a Internet representa um enorme sistema distribuído que se parece como um recurso único disponível em um conjunto mínimo de configuração de conexão e em apenas poucos cliques [7].

O conceito de sistema distribuído possui diversas definições e pontos de vista. Colouris [7] define um sistema distribuído como “um sistema em que componentes de hardware e software comunicam e coordenam suas ações apenas trocando mensagens”; Por outro lado, Tanenbaum [8] o define como “uma coleção de computadores independentes que aparecem ao usuário do sistema como um único computador”. E Keith Marzullo [9], define um sistema distribuído como “Uma coleção de processos sequenciais P_1, P_2, \dots, P_n

e uma rede capaz de implementar canais de comunicação unidirecionais entre pares de processos para troca de mensagem”.

Dessa forma, as definições convergem para um conjunto de aspectos comuns e essenciais que nos ajuda a distinguir sistemas distribuídos:

- São formados por um conjunto de computadores (ou unidades de processamento);
- São conectados por uma rede, sua comunicação é feita através de troca de mensagens e, portanto, não compartilham memória;
- São vistos pelos usuários como um recurso único (transparência [8]);
- Facilitam a utilização de recursos por usuários (ou aplicações);
- São escaláveis, isto é, possibilitam o aumento ou a diminuição de usuários ou recursos de maneira facilitada;
- Possuem desafios de temporização e sincronização.

Assim, embora seja possível perceber os desafios inerentes à implementação deste tipo de sistema, o ganho de poder computacional vale à pena esse esforço.

Nas próximas Seções serão apresentados os conceitos e principais aspectos de um *Cluster* computacional e Computação em *Grid*, os quais são tidos como base para o conceito de Nuvem Computacional.

2.1.1 *Cluster* Computacional

A primeira iniciativa na construção do conceito e implementação de *Clusters* Computacionais foi realizada pela IBM em meados dos anos 60 como alternativa de interligar grandes *Mainframes* para prover aos seus usuários uma forma comercial mais eficiente de paralelismo. Contudo, o conceito de clusterização não ascendeu como previsto até o surgimento de outras três tecnologias: microprocessadores de alta-performance, redes de alta velocidade e protocolos para computação distribuída de alta-performance [13].

Os recentes avanços dessas três tecnologias, somadas à sua acessibilidade (baixo preço decorrente da alta demanda) facilitaram a implementação de *clusters* computacionais como solução do antigo problema da eficiência de custos na busca de sistemas paralelos de alta performance. Para se construir um *cluster*, os computadores componentes devem ter alguns elementos essenciais [14]:

- Vários computadores de alta performance (PCs, *Workstations*, *Mainframes*);
- Sistemas Operacionais;

- Conectados por uma rede de alta performance (como *Gigabit Ethernet*);
- *Middleware* de gerenciamento de *Clusters* (serviços e abstrações que facilitam o desenvolvimento de aplicações distribuídas. O *middleware* permite ao *cluster* manter a uniformidade na presença de diferentes hardwares e SOs);
- Ambiente de computação paralela;
- Aplicações.

A Figura 2.1 apresenta a arquitetura conceitual de um *cluster*, mostrando suas camadas e alguns dos elementos descritos acima:

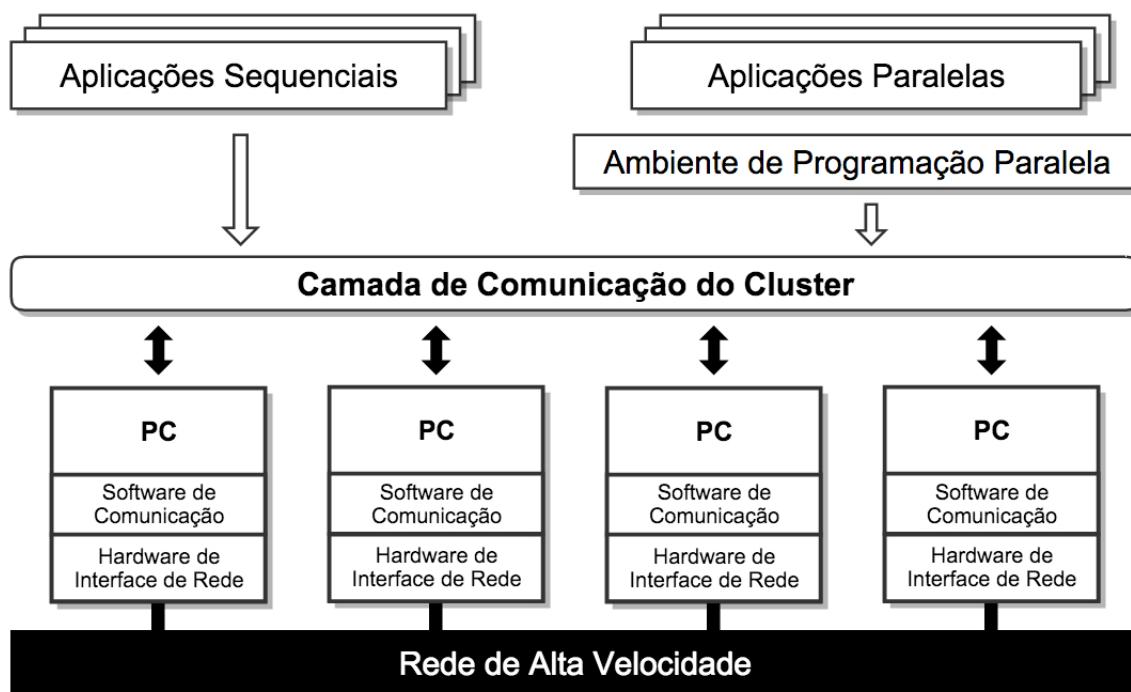


Figura 2.1: Arquitetura de um *Cluster*.

Na busca para provar o ganho de performance de *clusters* sobre as plataformas tradicionais de sistemas distribuídos, diversos projetos acadêmicos surgiram, como o Beowulf [14], Berkeley NOW [15] e HPVM [16].

Beowulf foi um *cluster* construído em 1994 por Thomas Sterling e Don Becker e consistia de 16 processadores DX4 conectados por um canal Ethernet, dedicados à programação paralela [14]. O *cluster* criado foi muito bem visto pela área acadêmica e comercial, tanto que a NASA se interessou por este novo modelo. Em pouco tempo, o *cluster* Beowulf foi bastante difundido, tornando-se “Projeto Beowulf” e passou a ser visto como gênero dentro da comunidade de Computação de Alta Performance (*High Performance Computing*).

Berkeley NOW (*Network of Workstations*) difere do *cluster* Beowulf pois seus nós computacionais são computadores completos - computadores pessoais com teclado, mouse e som - conectados pela Internet (os nós de um *cluster* Beowulf são nós especializados, *workstations* que ficam dia e noite apenas processando informações relacionadas ao *cluster*, não servindo, portanto, como um computador pessoal). Na maioria dos casos, nós NOW são utilizados para processamento a noite ou nos fins de semana, quando não estão sendo acessados pelos usuários, ou durante o dia, quando o *cluster* utiliza parte do poder de processamento do computador pessoal do usuário, utilizando ciclos ociosos de CPU e agregando-os pela Internet.

Esse modelo de computação distribuída tinha diversas limitações quanto ao tipo de aplicação que poderia ser executado no cluster NOW. Nesse contexto, o *cluster* Bewoulf tinha um melhor desempenho pois possuía as seguintes características:

- Utilizava processadores dedicados;
- Utilizava redes privadas de alta performance;
- Seu software era customizável;
- Softwares podiam ser clonados e enviados pela internet.

2.1.2 *Grid* Computacional

Em meados dos anos 90 o termo *Grid* Computacional foi criado para descrever tecnologias que possibilitariam usuários obterem poder computacional sobre demanda, assim como a energia é disponibilizada pela rede elétrica. Distingue do modelo tradicional de computação distribuída pelo foco em compartilhamento de grandes quantidades de recursos e, geralmente, por ser orientado à computação de alta performance. É uma forma de computação que envolve coordenar e compartilhar recursos computacionais, aplicações, dados, armazenamento e/ou recursos de rede através de sistemas dinâmicos e geograficamente dispersos [11], de maneira flexível, segura e coordenada servindo às chamadas Organizações Virtuais [12]. Uma Organização Virtual é um conjunto de indivíduos e/ou instituições interessados em acessar diretamente máquinas, software, dados e outros recursos. Esse compartilhamento é extremamente controlado, com regras definindo o que está, com quem e as condições daquilo que está sendo compartilhado.

Um sistema gerenciador de um *Grid* Computacional não está somente voltado ao processamento de dados, mas também ao gerenciamento de recursos de todo o sistema, seja ele software (aplicações, protocolos, sistemas operacionais) ou hardware (armazenamento, consumo de CPU, utilização de memória). Foster *et al.* [7] dividem sua arquitetura em 5 camadas, tomando como base a arquitetura do Protocolo de Internet (*Internet*

Protocol). Essa arquitetura é apresentada na Figura 2.2, na qual estão presentes as seguintes camadas:

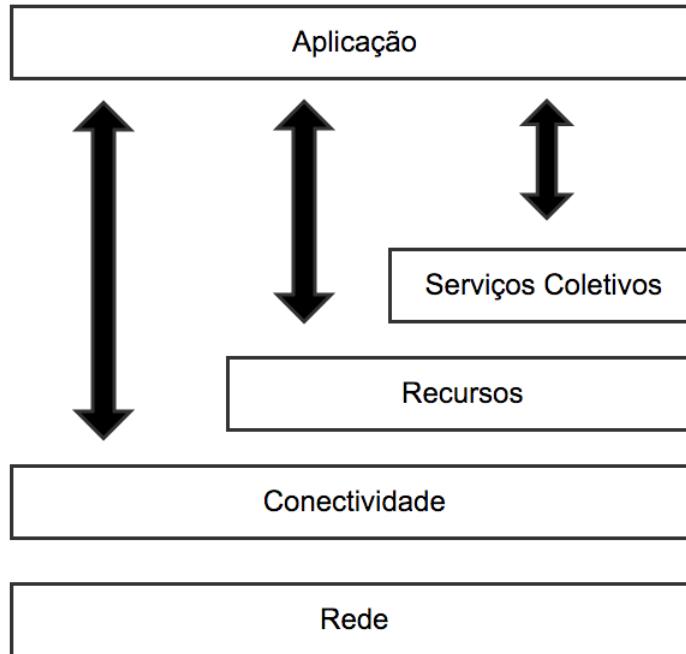


Figura 2.2: Arquitetura de um *Grid*, adaptado de [7].

- **Camada de Rede:** Fornece acesso compartilhado aos recursos computacionais a partir dos protocolos de *Grid*. Um recurso pode ser uma entidade lógica, como um sistema de arquivos distribuídos, por exemplo um *cluster*.
- **Camada de Conectividade:** Define o método de comunicação e protocolos de autenticação para transações específicas da rede interna do *Grid*. Além disso, provê mecanismos de segurança criptográfica para verificar a identidade de usuários e recursos.
- **Camada de Recursos:** Define protocolos para transações, inicialização, monitoração, controle e custos de operações que utilizam recursos individuais. Esta camada se preocupa apenas com o seu conjunto de recursos, não se importando com o estado global dos recursos compartilhados.
- **Camada de Serviços Coletivos:** Enquanto a Camada de Recursos está ligada à interações com recursos únicos, a Camada de Serviços Coletivos tem o objetivo de capturar e gerenciar transações entre conjuntos de recursos.
- **Camada de Aplicação:** A última camada na arquitetura de Grids abrange as aplicações que operam dentro do ambiente de uma Organização Virtual, ou seja,

esta camada está preocupada nas aplicações clientes que são executadas dentro de um grupo que compartilha o mesmo recurso.

Na próxima Seção serão apresentados os principais conceitos sobre Nuvens Computacionais, além das definições presentes na bibliografia, suas vantagens e desvantagens sobre os modelos de computação distribuída mostrados nessa Seção.

2.2 Nuvem Computacional

Com o rápido desenvolvimento de tecnologias de processamento e de armazenamento de dados e o crescimento da Internet, recursos de computação tornaram-se mais acessíveis, mais poderosos e mais disponíveis do que nunca. Essa tendência tecnológica permitiu a realização de um novo modelo de computação chamada computação em nuvem, em que os recursos computacionais (CPU, rede, memória, armazenamento) são fornecidos como serviços que podem ser alugados e liberado pelos usuários através da Internet em um modelo sob-demanda. Esse conjunto de tecnologias provê aos usuários uma gama enorme de opções de usos dessa nova plataforma. Com ela, os serviços são disponibilizados de forma transparente, representando uma nova maneira de se utilizar recursos computacionais.

A ideia principal por trás da computação em nuvem não é exatamente nova. Em meados dos anos 60, John McCarthy já havia previsto que o modelo de computação seria provido ao público de uma maneira similar ao que acontece nas redes elétricas, por exemplo. Isso aconteceu quando grandes empresas capazes de implantar enormes *datacenters* a custos competitivos resolveram adotar esse novo paradigma da computação. Em 2006, o termo “nuvem” ganhou popularidade quando o CEO (*Chief Executive Officer*) da Google, Eric Schmidt, o utilizou para descrever o modelo de negócios para prestação de serviços pela internet. Desse momento em diante, o termo “nuvem” realmente começou a ficar popular com ações de *marketing* e com o nascimento de diversas empresas especializadas.

A partir deste ponto, diversas definições surgiram em busca de um padrão para o conceito de computação em nuvem. Foster *et al.* [7] a definem como um paradigma de computação distribuída em larga escala movida pela economia da indústria, na qual um grande conjunto de recursos computacionais são provados sob-demanda a usuários externos pela Internet. Isso reforça a ideia de McCarthy de computação provida como serviços. Armburst *et al.* [17] trazem a definição de computação em nuvem como a união das aplicações disponibilizadas como serviços pela Internet e o hardware nos *datacenters* que as provém.

NIST (*National Institute of Standards and Technology*) a define como “um modelo que possibilita o acesso ubíquo, conveniente, sob-demanda a um conjunto configurável de

recursos computacionais (redes, servidores, armazenamento, aplicações, serviços) que podem ser rapidamente provisionados e liberados com um esforço mínimo de gerenciamento ou interação do provedor de serviço” [16].

Na indústria de Tecnologia da Informação (TI), o surgimento do modelo de Computação em Nuvem tem causado um enorme impacto nos últimos anos, onde grandes empresas como Google [45], Amazon [46] e Microsoft [47] se esforçam para fornecer plataformas de nuvem cada vez mais poderosas, disponíveis, de confiança e com melhor custo-benefício. Ao mesmo tempo, empresas procuram reformular seus modelos de negócios para se utilizarem dos benefícios deste novo paradigma. Dessa forma, a computação em nuvem fornece vários recursos interessantes que a torna atraente para empresas, tais como [5]:

- **Não necessita de investimento inicial:** A computação em nuvem usa um modelo de precificação em que o usuário paga apenas pelo recurso computacional utilizado. Um provedor de serviço não precisa investir na infraestrutura para começar a se utilizar dos ganhos tecnológicos de uma nuvem computacional. Ele simplesmente aluga os recursos de acordo com suas próprias necessidades e paga por aquilo que utilizar.
- **Reduz o custo operacional:** Recursos em um ambiente de nuvem podem ser rapidamente alocados e desalocados sob-demanda. Assim, um provedor de serviços não precisa mensurar sua capacidade de acordo com a carga de pico (carga máxima). Isso proporciona uma enorme economia uma vez que os recursos podem ser liberados para economizar custos de serviço quando a demanda for baixa.
- **Altamente escalável:** Provedores de infraestrutura possuem grandes quantidades de recursos a partir de *datacenters* e os tornam facilmente acessíveis. Um provedor pode facilmente expandir sua capacidade, a fim de lidar com um rápido aumento na carga de serviço.
- **Acessibilidade:** Serviços hospedados na nuvem são geralmente baseados na *web*. Assim, são facilmente acessíveis através de uma grande variedade de dispositivos com conexões de Internet. Estes dispositivos não só incluem computadores *desktop* e *notebooks*, mas também celulares e *tablets*.
- **Reduz os riscos de negócios e despesas de manutenção:** Por terceirizarem os serviços de infraestrutura para as nuvens, provedores de serviços transferem os riscos empresariais (tais como falhas de hardware) para os provedores de infraestrutura, que muitas vezes têm um maior conhecimento e estão melhor equipados para o gerenciamento desses riscos.

No entanto, embora a computação em nuvem tenha mostrado consideráveis oportunidades para a indústria de TI, também traz muitos desafios únicos que precisam ser cuidadosamente abordados, tais como confidencialidade das informações, integridade de dados, disponibilidade dos serviços, tolerância a falhas e um dos pontos mais discutidos que é a segurança.

2.2.1 Tipos de Nuvens

As nuvens computacionais podem ser divididas em quatro tipos diferentes de implantação [16]: nuvens públicas, privadas, comunitárias ou híbridas, as quais são descritas a seguir:

- **Nuvens Públicas:** Sua infraestrutura é disponibilizada para o público em geral, mantida e gerenciada por uma instituição acadêmica, comercial ou governamental e não impõe condições para sua utilização. Seus recursos computacionais são provisionados dinamicamente e são acessíveis através da Internet (geralmente, com a utilização de *webservices*).
- **Nuvens Privadas:** A utilização da infraestrutura desse tipo de nuvem é exclusivo de uma única companhia ou grupo de empresas, compreendendo múltiplos usuários. Seu gerenciamento pode ser feito pela própria instituição, por outra empresa ou pela junção de ambas as partes.
- **Nuvens Comunitárias:** Esse tipo de nuvem é provisionado para a utilização exclusiva de uma comunidade específica de consumidores de uma organização com interesses em comum (por exemplo, organizações com o mesmo requisito em segurança, mesmos requisitos em performance). Sua implantação pode ocorrer dentro ou fora da organização.
- **Nuvens Híbridas:** Esse tipo de infraestrutura é uma composição de dois ou mais tipos de nuvens (públicas, privadas ou comunitárias) em que o provedor de serviços disponibiliza tecnologias proprietárias que permitem portabilidade de dados e aplicações entre as nuvens que a compõe. Essa infraestrutura é muito útil quando, por exemplo, se quer trafegar dados sensíveis e sigilosos em um ambiente controlado e gerenciado internamente (utilização de uma nuvem privada), enquanto outros dados e aplicações podem trafegar em nuvens públicas ou comunitárias.

2.2.2 Arquitetura

A arquitetura de uma nuvem computacional pode ser dividida em quatro camadas distintas [7][15]: camada de hardware (também chamada de camada de *datacenter*), camada de

infraestrutura, camada de plataforma e a camada de aplicação. Elas podem ser descritas da seguinte maneira:

- **Camada de Hardware:** Esta camada é responsável pela gestão dos recursos físicos da nuvem, incluindo servidores, roteadores, *switches*, sistemas de energia e refrigeração. Na prática, a camada de hardware é implementada em enormes centros de dados (*datacenters*), os quais geralmente contém milhares de servidores organizados em *racks* e interconectados através de *switches*, roteadores ou outros dispositivos de rede. Problemas típicos na camada de hardware incluem configuração de hardware, tolerância a falhas, gestão, monitoração e refrigeração.
- **Camada de Infraestrutura:** Também conhecida como a camada de virtualização (*virtualization layer*). A camada de infraestrutura cria um *pool* de armazenamento e recursos de computação por meio do compartilhamento dos recursos físicos que utilizam tecnologias de virtualização como o Xen [17], o KVM [18] e o VMware [19]. A infraestrutura é uma camada essencial da computação em nuvem, uma vez que muitas características chaves, tais como a atribuição dinâmica de recursos, são apenas disponibilizados através de tecnologias de virtualização.
- **Camada de Plataforma:** Construída no topo da camada de infraestrutura, a camada de plataforma consiste de sistemas operacionais e *frameworks* de aplicação. A finalidade da camada de plataforma é minimizar o ônus da implantação de aplicativos diretamente em *containers* VM (método em que a camada de virtualização é executada dentro do sistema operacional em que reside).
- **Camada de Aplicação:** Reside no nível mais alto da hierarquia, a camada de aplicação consiste nas aplicações reais sendo executadas em nuvem. Diferentes de aplicações tradicionais, aplicações em nuvem podem aproveitar o recurso de escalonamento automático para alcançar melhor desempenho, disponibilidade e custo operacional mais baixo.

2.2.3 Modelos de Serviço

A computação em nuvem emprega um modelo de negócios orientado a serviços, ou seja, recursos de hardware e software são disponibilizados sob-demanda. Conceitualmente, cada camada da arquitetura descrita na Seção 2.2.2 pode ser implementada como um serviço à camada acima. Em outras palavras, a camada acima será a consumidora dos serviços providos pela camada imediatamente abaixo. Esse modelo em camadas está descrito na Figura 2.3.

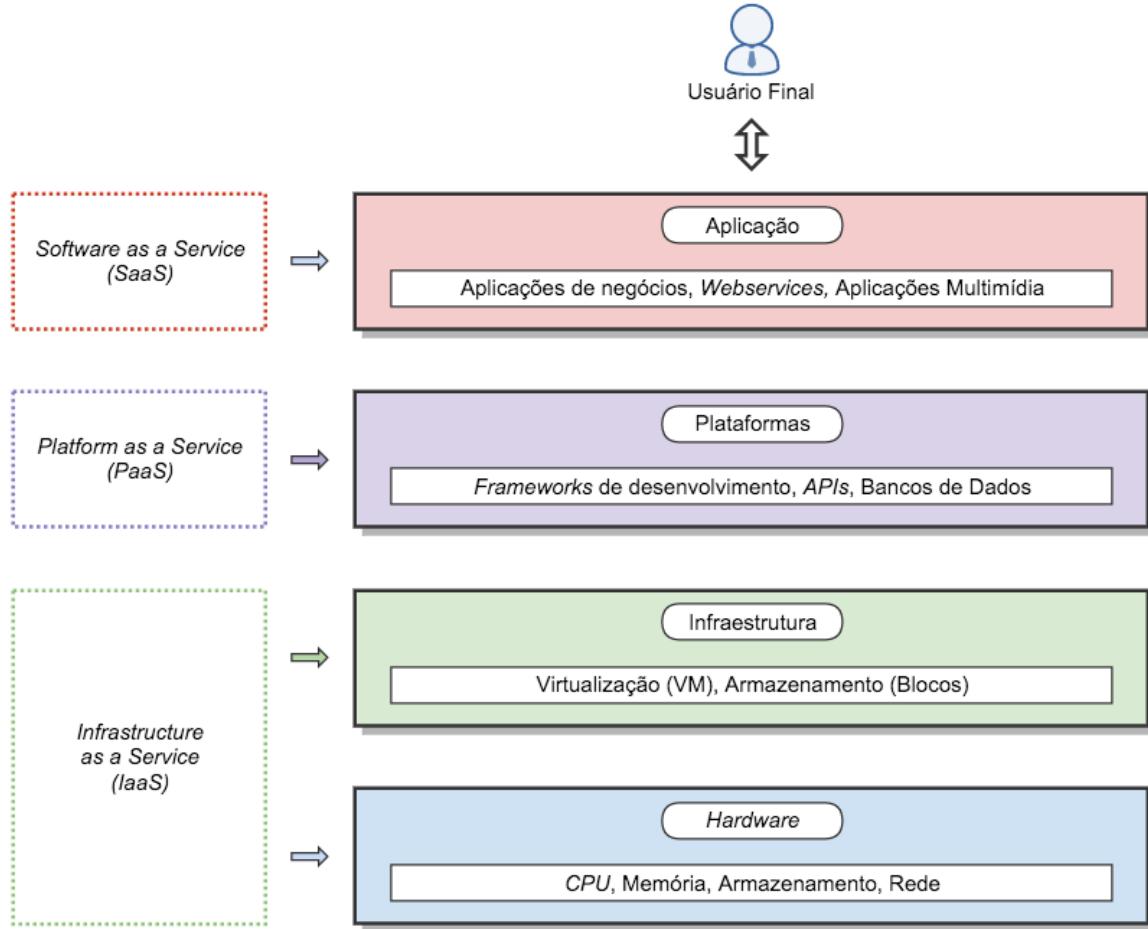


Figura 2.3: Arquitetura de uma Nuvem Computacional [15].

Esses serviços são categorizados no seguintes três modelos [7][5][15]:

- **Infrastructure-as-a-Service (IaaS)**: provê recursos computacionais ao usuários, tais como poder computacional, armazenamento de dados e redes virtuais para que possam implantar e executar qualquer tipo de software. A tecnologia de virtualização é essencial para este modelo, pois dá ao provedor deste serviço a habilidade de que vários usuários possam compartilhar recursos de uma mesma máquina física. Exemplos de provedores deste modelo são: Amazon EC2 [28], Google Compute Engine [48] e GoGrid [49].
- **Platform-as-a-Service (PaaS)**: Dá ao consumidor deste serviço a capacidade de implantar aplicações criadas pelo usuário ou adquiridas de terceiros, criadas a partir de linguagens de programação, bibliotecas de software, *APIs* (*Application Programming Interface*) e ferramentas suportadas pelo provedor. Nesse modelo, o usuário não gerencia aspectos do hardware e do software necessários para execução

da infraestrutura da nuvem. Alguns exemplos são: Heroku [29], CloudForge [50] e AppScale [51].

- **Software-as-a-Service** (SaaS): Provê aplicações que estão sendo executadas em algum ponto da Internet por um provedor IaaS, eliminando a necessidade de instalar e executar a aplicação no computador do usuário. São independentes de plataforma e geralmente provêm uma interface para que o usuário possa acessar e utilizar esse serviço. Algumas empresas que implementam esse modelo de serviço são: Abiquo [52], SalesForce [53] e Zendesk [54].

A próxima Seção traz os conceitos de um novo modelo de estrutura de nuvens computacionais, a federação de nuvens, a qual traz inovações e satisfaz necessidades trazidas pelos modelos de nuvem descrito nesta Seção.

2.3 Federação de Nuvens

Com o passar dos anos, diversas necessidades surgiram no contexto da computação em nuvem e a sua utilização de forma isolada passou a não ser mais suficiente para algumas aplicações. Assim, medidas foram tomadas como forma de supri-las, tais como a criação de *datacenters* espalhados ao redor do mundo para aumentar a tolerância a falha de suas aplicações, prevendo possíveis quedas de seus servidores e a redundância de dados, visando manter a taxa de disponibilidade dos serviços providos. Dessa forma, integrar diferentes serviços de nuvens para continuar atendendo às expectativas de qualidade de serviço (*QoS - Quality of Service*) passou a ser um requisito necessário para continuar fornecendo serviços de forma rápida, eficiente e escalável.

Assim, a federação de nuvens é uma área de pesquisa interessante na computação em nuvem e está muito focada na continuidade do *QoS*, empregando-se no desenvolvimento de mecanismos para manter o serviço disponibilizado e sua qualidade.

Diversas pesquisas foram desenvolvidas nessa área e novos termos foram criados para tratar desse conceito, como *Intercloud* (“Pense nas ilhas de nuvens existentes que se fundem em uma nova *Intercloud* interoperável nas quais os aplicativos podem ser modos para ela e operarem em múltiplas plataformas ...” [2]) e *Cross-Cloud* (“Para o benefício da sociedade humana e do desenvolvimento da computação em nuvem, uma plataforma uniforme e interoperável *Cross-Cloud* certamente nascerá em um futuro próximo ... ” [3]).

Em um cenário de federação de nuvens, cada provedor de nuvem é capaz de ampliar de forma transparente a sua própria quantidade de recursos de virtualização (ou seja, o aumento do número de máquinas virtuais instanciadas) requerendo maior poder computa-

cional e capacidades de armazenamento para outras nuvens. Por conseguinte, o operador de nuvem é capaz de satisfazer qualquer solicitação de alocação enviado por seus clientes.

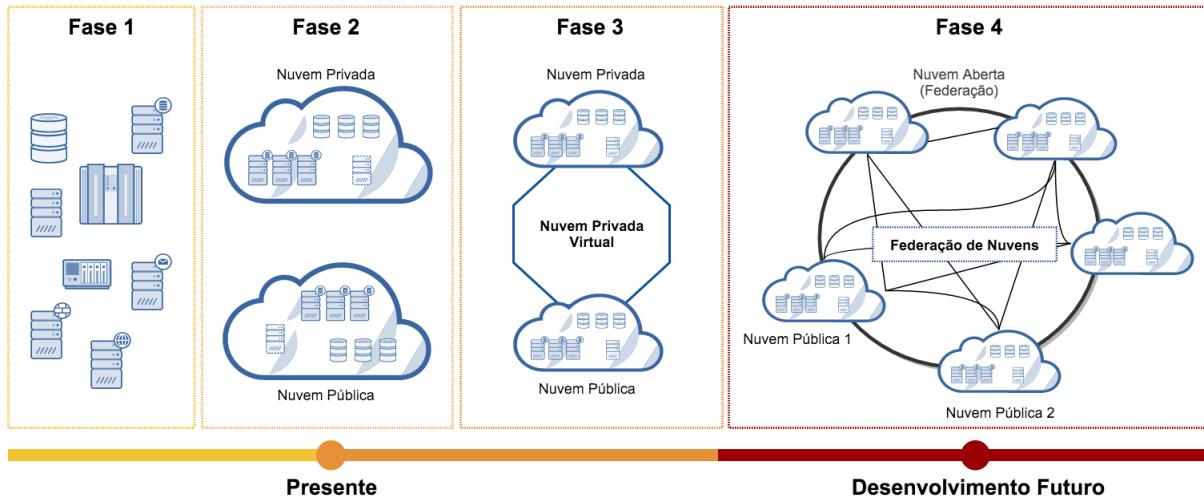


Figura 2.4: Fases da computação em nuvem, segundo White E. *et al.* [26].

A federação de nuvens passou por quatro fases. A Figura 2.4 apresenta claramente essas quatro fases de implantação, as quais são"

- **Primeira Fase:** É possível notar *datacenters* desconexos, sem utilizar o conceito de nuvem computacional, ou seja, estão espalhados pelo mundo e realizam todo o trabalho sozinhos, contando apenas com a sua própria infraestrutura e mecanismo de gerenciamento. Assim, estão mais propensos à falhas e não compartilham recursos computacionais com outros *datacenters*;
- **Segunda Fase:** Nessa fase da computação em nuvem, as nuvens já possuem tecnologia e mecanismos para disponibilizar seus serviços pela Internet, estando acessíveis a usuários finais. É o que vemos em serviços como Dropbox (*Software-as-a-Service*) [27], Amazon EC2 (*Infrastructure-as-a-Service*) [28] ou Heroku (*Platform-as-a-Service*) [29];
- **Terceira Fase:** As nuvens já conseguem compartilhar recursos entre si por meio de um provedor virtual que realiza todo o balanceamento das nuvens, de forma que nenhum provedor fique ocioso enquanto há outros que se encontram utilizando sua carga máxima de recursos. Celesti *et al.* [20] as tipificou em duas definições: *Home Clouds*, nuvens com sua capacidade computacional saturada e *Foreign Clouds*, aquelas com recursos ociosos disponíveis e que podem ser utilizados por outras nuvens. Nesse caso, uma solicitação seria enviada para a nuvem com recursos ociosos, e esses seriam disponibilizados para a nuvem sobrecarregada;

- **Quarta Fase:** Atualmente os provedores de serviços estão nessa última fase, na qual encontram-se as várias federações de nuvens espalhadas pelo mundo, trabalhando de forma interoperável entre si, ou seja, dividindo os seus recursos quando ociosos, e requisitando recursos quando saturados. Esse conceito de “nuvem de federações” forma um cenário no qual os recursos desperdiçados na nuvem são escassos, dada sua utilização por outros provedores. Isso traz um melhor balanceamento de carga, com poucos serviços indisponíveis por uma possível saturação, fazendo com que os usuários experimentem uma melhor experiência de aplicações hospedadas nas nuvens.

imagem de uma federação de nuvens

Apesar das vantagens desse novo modelo de computação em nuvem, a implementação de tal cenário de federação de nuvens não é trivial, pois as nuvens são mais complexas do que os sistemas tradicionais. De fato, enquanto as nuvens são tipicamente compostas por recursos heterogêneo (diversos sistemas operacionais, configurações, poder computacional, latência de rede) e dinâmicos, os modelos existentes de federação são projetados para ambientes estáticos nos quais acordos prévios entre as partes são necessários para estabelecer a federação. Assim, a federação de nuvens precisa atender aos seguintes requisitos [20]:

- **Automatismo e Escalabilidade:** Usando mecanismos de descoberta, a nuvem computacional com a utilização de recursos computacionais saturada deve ser capaz de escolher a melhor nuvem que satisfaça as suas exigências de recursos;
- **Segurança Interoperável:** Se faz necessária a integração de diferentes tecnologias de segurança entre nuvens computacionais, permitindo uma nuvem ser capaz de juntar-se à federação sem alterar as suas próprias políticas de segurança.

Capítulo 3

Workflow Científico

Neste Capítulo serão apresentados os conceitos e principais aspectos relacionados à *Workflows*, *Workflows* científicos e como os chamados Sistemas Gerenciadores de *Workflows* Científicos ajudam cientistas na construção destes fluxos de trabalho e no gerenciamento de seu ciclo de vida. Também apresenta alguns Sistemas de *Workflows*, mostrando suas características, assim como suas vantagens e desvantagens. Para isso, a Seção 3.1 mostra os conceitos básicos de um *Workflow* Científico, a Seção 3.2 apresenta o seu ciclo de vida, mostrando suas fases e principais características. A Seção 3.3 introduz os conceitos e definições de um Sistema Gerenciador de *Workflows* Científicos e na Seção 3.4 são apresentados alguns exemplos desses sistemas, consolidados nos meios acadêmicos e comercial. Por fim, na Seção 3.5 são descritas as vantagens percebidas ao se utilizar um Sistema Gerenciador de *Workflows* Científicos.

3.1 Conceitos e Definições

Um *workflow* é uma sequência de passos bem definidos usados para automatizar algum processo, de acordo com um conjunto de regras definidas, permitindo que estes possam ser executados por uma outra pessoa gerando a mesma saída. Nessa automação, documentos, informações ou tarefas são processados, de acordo com um conjunto de procedimentos. Um *workflow* é composto de grupos de dados, fases de análise, fluxos e ferramentas ordenados de maneira a se atingir o objetivo desejado. Nesse contexto, existem diversos software computacionais, que utilizam uma gama enorme de tecnologias diferentes, que possuem a tarefa de gerenciar o ciclo de vida de um *workflow*. Esses sistemas são geralmente conhecidos como Sistemas Gerenciadores de *Workflow* (*WfMS* - *Workflow Management Systems*), e garantem que os passos que automatizam os processos ocorrem na sequência correta.

No contexto da Bioinformática, as análises computacionais dos dados obtidos por meio de sequenciadores automáticos são realizadas em diferentes fases, ou passos. Para cada fase, existe um conjunto de ferramentas de Bioinformática a ser utilizado. Entretanto, cada tipo de pesquisa implica numa combinação diferente de ferramentas, de acordo com os objetivos da pesquisa, e este fluxo de passos é chamado de *workflow* de Bioinformática. Esse dinamismo gera uma complexidade adicional ao projeto, pois é necessário implementar mecanismos de gerenciamento dessa entidade (*workflow*) a cada nova pesquisa, assim como os dados utilizados em suas entradas (*inputs*) e saídas (*outputs*).

De acordo com Singh *et al.* [21][22] um *workflow* científico é definido como “*uma série de atividades estruturadas e cálculos que surgem na resolução de problemas científicos*”, e um “*processo automatizado que combina dados e processos em um conjunto estruturado de passos para implementar soluções computacionais para um problema científico*”. *Workflows* científicos distinguem-se de *workflows* de negócio (*business workflows*) pois contém fluxos centrados nos dados [23][24], são mais flexíveis [25] e são utilizados principalmente para descrever a execução de experimentos científicos [25].

Um dos objetivos de se utilizar *workflows* científicos é apoiar e, sempre que possível, automatizar fases como visualização, tarefas repetitivas, o acesso aos dados, transformação, análise e que, se feitas de outra maneira, estariam propensas a erros. Assim, *workflows* científicos são muitas vezes usados para encadear aplicativos especializados e de novos métodos de análise de dados. No entanto, assim como ocorre em *workflows* de negócios, *workflows* científicos não são apenas mecanismos de gerenciamento da execução de uma sequência de passos. Outras áreas como modelagem, *design*, análise e reutilização de *workflows* estão se tornando cada vez mais importante neste contexto. Assim, as principais vantagens ao se utilizar *workflows* científicos são: *(i)* auxiliar pesquisadores permitindo que eles se concentrem no domínio específico do seu trabalho, ao invés de perder tempo lidando com complexas questões de gestão de dados e de software complicados; e *(ii)* evitar desperdícios de poder computacional, otimizando a execução de *workflow* sobre os recursos disponíveis.

A Seção seguinte trata da conceituação do ciclo de vida de um *workflow*, suas etapas e principais aspectos.

3.2 Ciclo de Vida de um *Workflow*

As várias fases e etapas associadas ao desenvolvimento, implantação e execução de *workflows* científicos compreendem o seu ciclo de vida [30]. Essas fases são em grande parte suportadas por sistemas gerenciadores de workflow existentes (*WfMS - Workflow Management System*), utilizando uma ampla variedade de abordagens e técnicas. A Figura 3.1

demonstra o ciclo de vida de um *workflow*, e suas fases podem ser descritas da seguinte maneira:

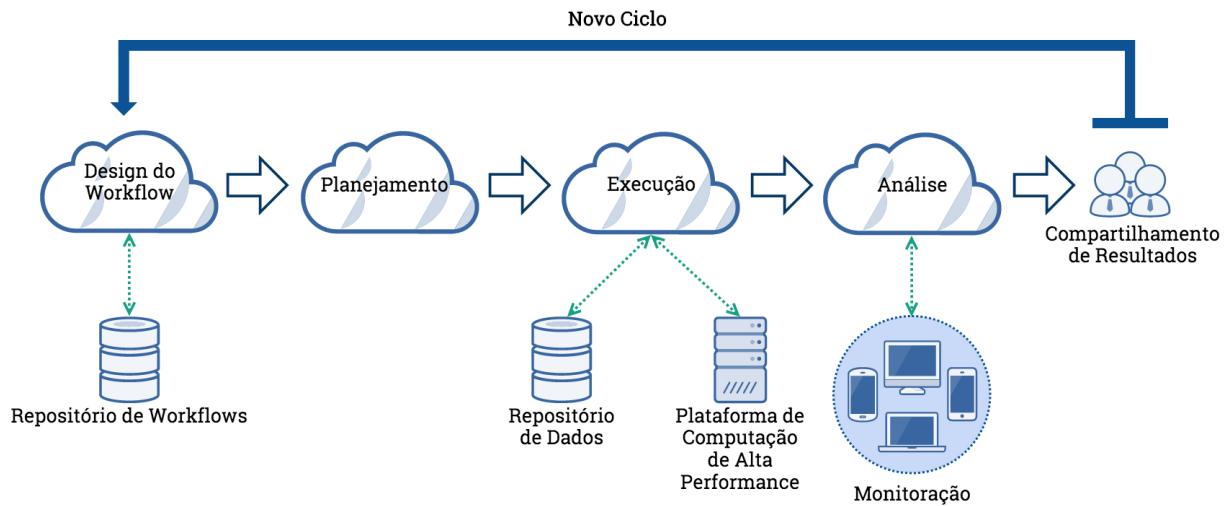


Figura 3.1: Ciclo de Vida de um *Workflow*.

- **Projeto e Composição:** O desenvolvimento de *workflows* científicos geralmente começa com o levantamento de requisitos dos cientistas envolvidos em uma pesquisa. Uma especificação das funcionalidades do *workflow* desejado é criada, e então um *workflow* é montado com base nesta especificação. O desenvolvimento de um *workflow* difere da programação tradicional de muitas maneiras. Normalmente, seu desenvolvimento abrange a composição e a configuração de um *workflow* para fins especiais de componentes, *subworkflow* e serviços pré-existentes. A construção de *workflows* assemelha-se mais à programação *Script* do que o desenvolvimento de aplicações convencional. Durante a composição do *workflow*, o usuário ou cria um novo *workflow* modificando um existente, ou então compõe um novo *workflow* a partir do zero usando componentes e *subworkflows* obtidos a partir de um repositório externo. Em contraste com o conceito de *workflows* de negócios, onde padrões foram desenvolvidos ao longo dos anos (por exemplo, mais recentemente WS-BPEL 2.0 [JE07]), sistemas de *workflow* científicos tendem a usar a sua própria linguagem e formatos de troca (por exemplo, SCUFL [Tav], GPEL [WHH05], e MOML [BLL + 08], entre outros). As razões para esta divergência são a vasta gama de modelos de computação (Computação em Nuvem, por exemplo) utilizados na construção de *workflows* científicos e o foco do desenvolvimento das funcionalidades envolvidas, que no caso são orientadas para o cientista.

- **Planejamento de Recursos:** Assim que a descrição do *workflow* é construída, sistemas gerenciadores de *workflows* científicos muitas vezes fornecem várias funcionalidades antes da execução. Essas funções podem incluir a validação (por exemplo, verificação de tipo), a alocação de recursos, a programação, a otimização, os parâmetros de execução e a escolha de dados de entrada. Mapeamento de *workflows* é por vezes utilizado para se referir às decisões de otimização e programação feitas durante a fase de planejamento de recursos. Em particular, durante a fase de concepção e composição, os recursos de destino a serem utilizados para a execução não são tipicamente escolhidos. Mapeamento de fluxo de trabalho, então, refere-se ao processo de geração de um *workflow* executável baseada em uma descrição abstrata do *workflow* independente de recurso [DBG + 03]. Em alguns casos, o usuário executa o mapeamento diretamente escolhendo os recursos adequados. Em outros casos, o sistema gerenciador do *workflow* executa automaticamente o mapeamento. Neste último caso, os usuários tem permissão para construir *workflow* em um nível de abstração acima, não definindo aspectos do ambiente de execução.
- **Execução:** Uma vez que um *workflow* é mapeado e dados foram selecionados e colocados à disposição do sistema gerenciador de *workflows*, o mesmo pode ser executado. Durante a execução, um sistema gerenciador de *workflows* pode registrar o histórico dos processos, bem como fornecer funções de monitoramento e *failover* (recuperação em caso de falha) em tempo real. Dependendo do sistema, geralmente é realizada a gravação dos passos que foram invocadas durante a execução do *workflow*, os dados consumidos e produzidos por cada passo, um conjunto de dependências, e assim por diante. Um *workflow* pode mudar durante sua execução (por exemplo, devido a mudanças na disponibilidade de recursos), assim, a evolução deste *workflow* é dinâmica e deve ser registrada, a fim de dar suporte a análise posterior de sua execução.
- **Análise da Execução:** Após a execução do *workflow*, os cientistas envolvidos muitas vezes precisam inspecionar e interpretar os resultados gerados pelo *workflow*. Isso envolve a avaliação daquilo que foi produzido (ou seja, se o resultado faz sentido), examinar os rastros de execução do *workflow* (verificar o passo a passo da execução até se chegar ao resultado), depurar o *workflow* (verificar possíveis erros), e análise de desempenho (examinar o tempo gasto por cada passo).
- **Compartilhamento dos Resultados:** Dados e o produto do *workflow* podem ser publicados e compartilhados entre o grupo de pessoas envolvidas. A partir do momento em que os dados gerados pela execução do *workflow* são enviados para

um repositório compartilhado, novas iterações do ciclo de vida do *workflow* podem começar novamente.

Outro aspecto muito importante na manutenção de *workflows* científicos é definir os papéis dos usuários do sistema (*User Roles*) [30]. Cada usuário possui uma função específica e bem definida no controle do ciclo de vida de um *workflow*. Dessa forma, os usuários de sistemas gerenciadores de *workflows* científicos podem desempenhar uma série de funções diferentes dentro das fases indicadas. Um *designer* de *workflows* é geralmente um cientista que desenvolve um novo protocolo experimental ou analítico (ou uma nova variante de um método existente).

Como mencionado, um projeto de *workflow* é muitas vezes provocado por alguma forma de análise de requisitos. Assim, o desenho e os requisitos associados podem ser usados por um engenheiro de *workflows* para a implementação da descrição abstrata ou executável do *workflow* associado. Um operador do *workflow* é um usuário que executa *workflows* utilizando as entradas desejadas. Um operador pode iniciar um *workflow* diretamente através de um sistema gerenciador de *workflows* científicos, ou indiretamente através de outro aplicativo (por exemplo, dentro de um portal web), monitorar a sua execução (por exemplo, através de um *dashboard*) e, posteriormente, validar os resultados obtidos. Os papéis de usuário acima descritos não são necessariamente disjuntos. Por exemplo, uma única pessoa pode assumir os papéis de *designer*, de engenheiro e de operador do *workflow*.

Dessa forma, sistemas gerenciadores de *workflows* científicos visam tornar o projeto, a execução e o resultado da análise mais fácil em comparação com abordagens baseadas em *script* tradicionais para automação de processos científicos.

Na Seção 3.3 serão abordados características dos chamados sistemas gerenciadores de *workflows* científico

3.3 Sistemas Gerenciadores de *Workflows* Científicos

A tecnologia da informação está revolucionando a forma que muitas pesquisas são conduzidas, com novas técnicas e modelos de computação, em campos multidisciplinares, tais como Bioinformática, Informática Biomédica, Quimioinformática, Geoinformática, etc. Para avançar ainda mais nessa nova ciência orientada a dados e à informação através da utilização de avançada infraestrutura de TI, grandes investimentos são realizados. Enquanto muitos esforços se concentram no desenvolvimento dessa infraestrutura, com novas tecnologias de *grid* e, mais recentemente, a utilização de nuvens computacionais, cientistas estão interessados em ferramentas, como bancos de dados distribuídos e recursos de *grids*

computacionais, que permitam-lhes projetar, montar e executar seus próprios *workflows* científicos.

Idealmente, o cientista deve ser capaz de ligar qualquer recurso de dados científicos e serviços computacionais em um *workflow* científico, inspecionar e visualizar dados à medida que são processados, fazer alterações de parâmetros quando necessário executar novamente apenas os componentes afetados por um possível erro. Assim, um sistema de *workflows* científicos torna-se um ambiente de resolução de problemas científicos, sintonizado cada vez mais com a utilização de infraestruturas de sistemas distribuídos como *grids* e nuvens computacionais.

3.4 Exemplos de Sistemas Gerenciadores de *Workflows* Científicos

Nesse cenário de utilização de infraestrutura de TI para otimizar o projeto, a execução e a análise de *workflows* científicos, diversas propostas de sistemas surgiram utilizando uma vasta gama de técnicas, infraestruturas e tecnologias, tais como: DiscoveryNet [34], Taverna [35], Triana [36], Kepler [37] e Pegasus [38][39]. No que diz respeito a esses sistemas gerenciadores, seguem abaixo algumas características e aspectos principais.

DiscoveryNet

O sistema *Discovery Net* foi concebido em torno de um modelo de *workflows* científicos para a integração de fontes de dados distribuídos e ferramentas analíticas dentro de uma infraestrutura de *Grid*. O sistema foi originalmente desenvolvido como parte do projeto financiado de *e-Science* do Reino Unido, *Discovery Net* (2001-2005) com o objetivo de produzir uma plataforma orientada para aplicações de alto nível, com foco na capacitação de cientistas na derivação de novos conhecimentos a partir de dispositivos, sensores, bancos de dados, componentes de análise e recursos computacionais acessíveis pela Internet. Seu conjunto dedicado de componentes para a mineração de dados tem sido usado como base para inúmeros projetos de diversos domínios do conhecimento. Muitas das ideias de pesquisa desenvolvidas no âmbito do *Discovery Net* também foram incorporadas dentro do sistema IDBS (antigo InforSense KDE) [44], um sistema de mineração de dados e *workflows* de gerenciamento comercial que tem sido amplamente utilizada para aplicações de negócios orientados.

A Figura 3.2 fornece uma visão geral de alto nível do *Discovery Net*. O sistema é baseado em uma arquitetura multicamada, com um servidor de *workflows*, fornecendo funções necessárias para a criação e execução de *workflows*, tais como a integração e acesso a dados

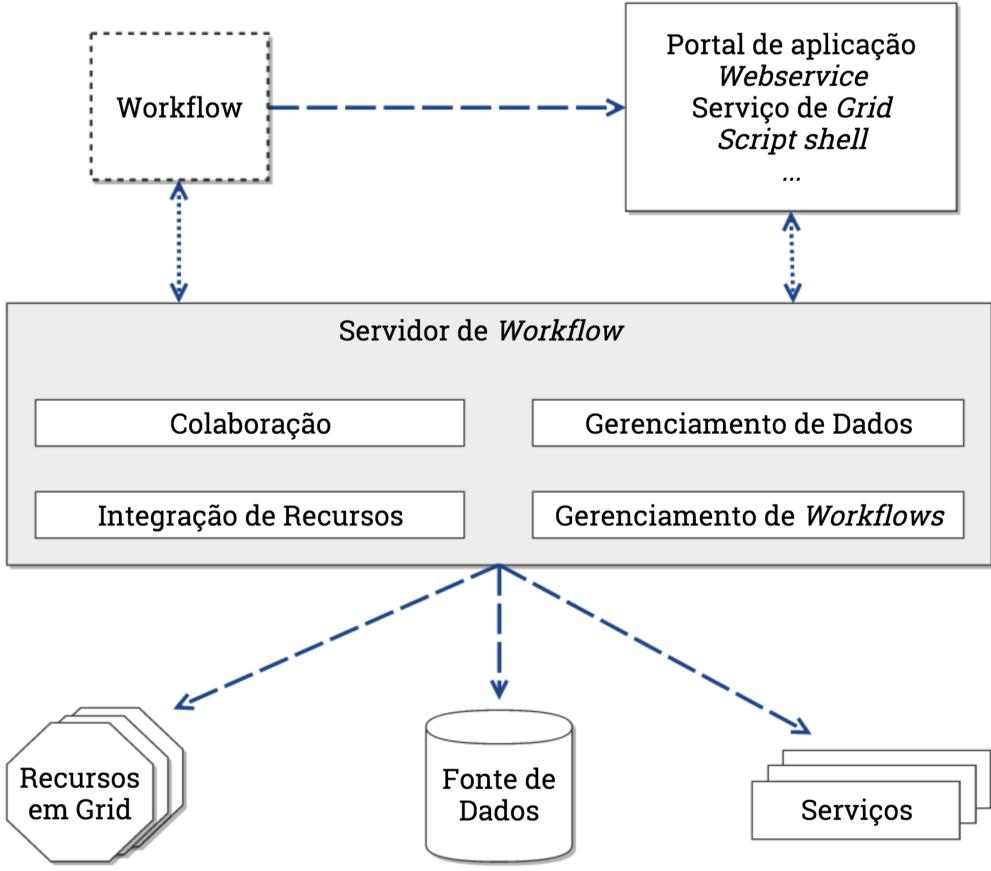


Figura 3.2: Arquitetura multicamada do *DiscoveryNet* (adaptado de [4]).

e recursos computacionais remotos, ferramentas de colaboração, publicação e mecanismos de visualização. O projeto do sistema tem como alvo os especialistas do domínio, ou seja, cientistas, ao invés do foco em desenvolvedores de sistemas distribuídos. Os *workflows* criados desta forma também podem ser executado a partir de interfaces especializadas baseadas em tecnologias *web*. A implementação do sistema em si tem evoluído ao longo dos últimos anos, a partir de um protótipo direcionados a projetos específicos de um sistema de força industrial amplamente utilizado por organizações comerciais e acadêmicas.

Taverna

O principal objetivo do Taverna é satisfazer as necessidades dos cientistas de bioinformática que precisam construir *workflows* científicos a partir de inúmeros *webservices* disponíveis remotamente através da Internet. Dessa forma, existe um esforço significativo no desenvolvimento do sistema Taverna para a aquisição e organização desses *webservices* em uma coleção útil de componentes. Seus principais requisitos são:

- 1.

Taverna foi construído utilizando-se o [myGrid](#) [], que é um projeto de construção de *middleware* que suporta o desenvolvimento de *workflows* baseados em experimentos *in silico* (experimentos realizados através de simulações computacionais) de Biologia. Financiado pelo Programa *e-Science* do Reino Unido a partir de 2001, myGrid tem desenvolvido um conjunto de componentes de código aberto que pode ser usado de forma independente e em conjunto. Estes incluem um diretório de serviços, ferramentas de pesquisa atuantes em descrições semânticas de recursos e dados externos; repositórios de dados e metadados semanticamente dirigidos para a gravação de um *workflow* e seu ciclo de vida, bem como outros componentes, tais como processamento distribuído de consultas (*queries*) e notificação de eventos. Dessa forma, Taverna foi projetado para ser o ambiente de desenvolvimento e execução de *workflows* baseado em componentes do projeto myGrid.

Taverna é um nome comum usado para o sistema de *workflows* científicos que compreende o *Workbench* Taverna, que é uma ferramenta gráfica de criação de *workflows*, o servidor Taverna, utilizado para execução remota de *workflows*, seu conjunto de ferramentas de linha de comando (*Command Line Tool*) e, mais recentemente, o Taverna *Online*, que é a versão deste sistema disponível pela Internet. A Figura 3.3 mostra a interface percebida pelo usuário ao se utilizar o Taverna em sua versão online.

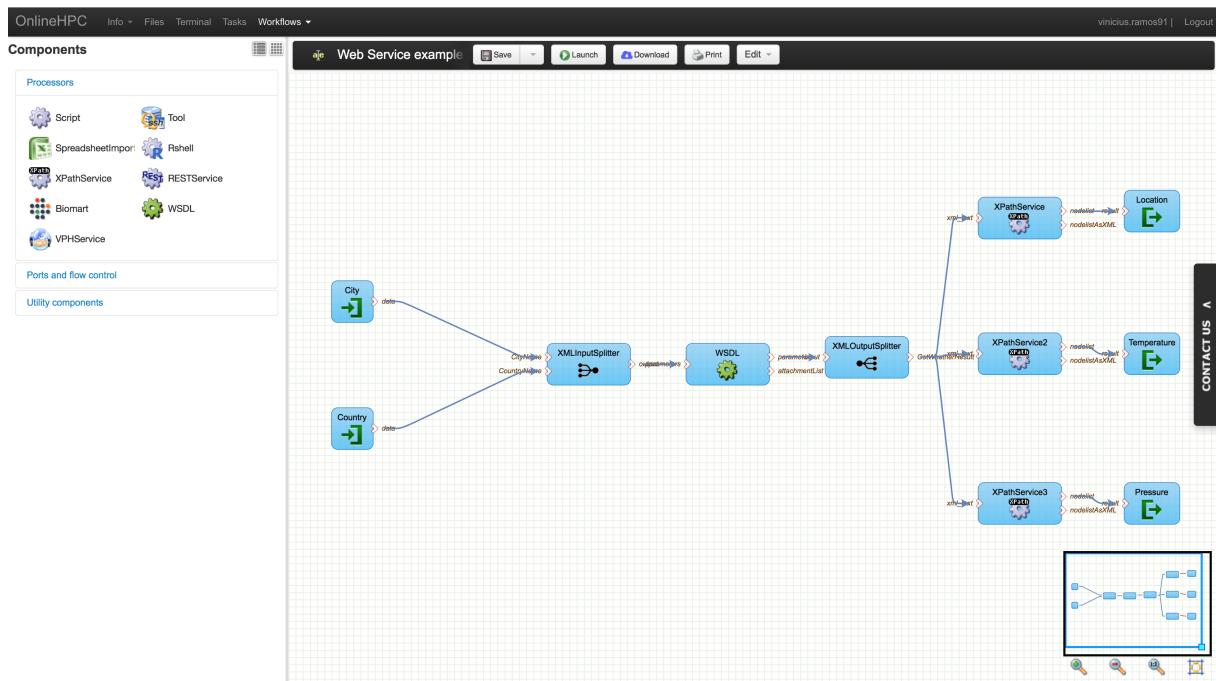


Figura 3.3: Ambiente gráfico do Taverna Online (retirado de [16]).

Triana

Triana [14] [4] é uma plataforma *open source*, distribuída, independente de plataforma, escrita na linguagem de programação Java utilizada como Ambiente de Resolução de Problemas (*PSE - Problem Solving Environment*). Um *PSE* é um ambiente de computação completo e integrado para composição, compilação e execução de aplicativos em uma área específica [9].

O Triana é um ambiente gráfico interativo que permite aos usuários compor aplicações e especificar seu comportamento de maneira distribuída. Conforme pode ser visto na Figura 3.4, o usuário cria um *workflow* arrastando as unidades desejadas para a área de trabalho e interliga estes arrastando uma ligação entre eles. Embora o foco aqui seja a interface gráfica, Triana é composto por um conjunto complexo de componentes de interação com potencial de criar sistemas completos ou qualquer subconjunto dele.

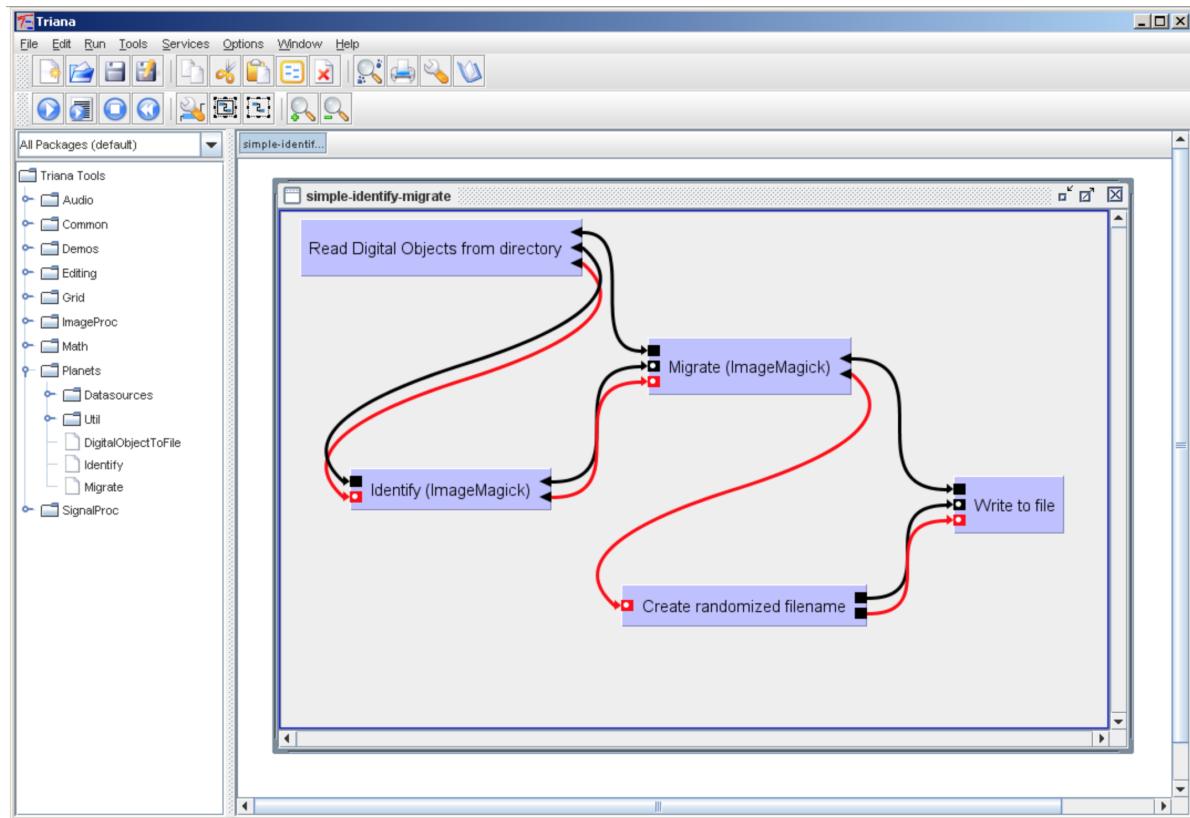


Figura 3.4: Ambiente gráfico do Triana (retirado de [5]).

Esta abordagem dá ao Projeto Triana a flexibilidade necessária para ser aplicado a muitos cenários diferentes e em muitos níveis diferentes. Por exemplo, ele pode ser usado como uma base para aplicações de *workflows* em *grids*, para conectar componentes de *grid* orientados a dados e gerenciar o *workflow* entre eles, ou como um sistema de análise de dados para aplicações de imagem, processamento de sinais ou texto, permitindo que

um cientista possa aplicar rapidamente algoritmos em conjuntos de dados e visualizar os resultados.

Triana também pode ser usado como um editor gráfico de *scripts* de alto nível para a criação de *workflows* baseados em *scripts*.

Kepler

Kepler é um sistema para construção, composição e mecanismo de orquestração de *workflows* científicos, desenvolvido a partir do Sistema *Ptolemy II* [3], que é uma ferramenta de modelagem orientada ao “ator” (*actor-oriented modeling tool*) [4].

O foco de Kepler é na análise de dados e modelagem de maneira genérica. Assim, Kepler é adequado para modelar processos em uma ampla variedade de domínios científicos, desde modelos físicos até *webservices* de bioinformática. Ao invés de tentar fornecer uma semântica genérica para todos possíveis tipos de processos encontrados nestes domínios, Kepler separa o mecanismo de execução do *workflow* e atribui um modelo de computação, chamado diretor (*director*) [13], para cada *workflow*. Os componentes do *workflow* no Kepler são chamados de atores (*actors*) e representam operações ou fontes de dados, com um determinado número de portas, podendo ser entrada, saída, ou ambos, os quais atuam como pontos finais (*end-points*) para conexões. A interação entre esses componentes pode ser vista na Figura 3.5.

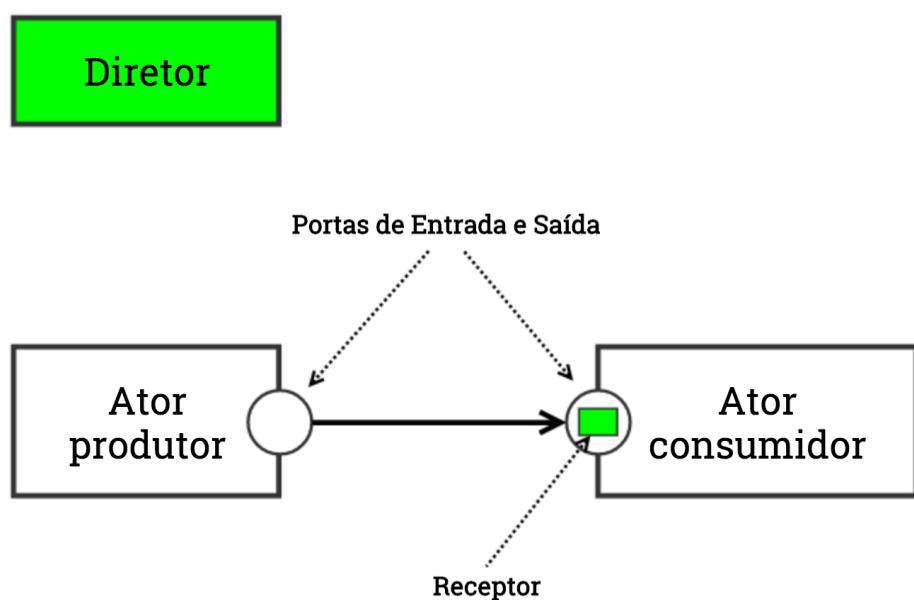


Figura 3.5: A semântica da interação entre os componentes é determinada pelo *director*, o qual controla a execução (adaptado de [13]).

A interação mais simples consiste em um *actor* consumir um conjunto de dados sobre

cada porta de entrada e produzir um sinal de saída sempre que ele executa. No entanto, existem numerosos casos onde mais do que um sinal pode ser consumido (ou produzido) para cada execução.

Director é um conceito-chave no sistema Kepler. Enquanto *actors* e relações, juntos, constituem um modelo de *workflow*, os *directors* se encontram em um nível acima, formando seu modelo de execução ou modelo de computação (*MoC - Model of Computation*). Nesta configuração, um *actor* só tem conhecimento da operação a ser executada e que saídas serão produzidas. A decisão de como e quando escalar a execução de cada *actor* é deixado para o *director*. Por exemplo, uma operação de adição pode aceitar dados de entrada fornecidos por diversos mecanismos, incluindo eventos discretos, envio de dados e passagem assíncrona de mensagens (*asynchronous message passing*). Portanto, dependendo do *director* utilizado, os *actors* podem ter *threads* de controle separadas, ou eles podem ter suas execuções desencadeada pela disponibilidade de uma nova entrada.

Usando Kepler, os cientistas podem capturar seus *workflows* em um formato que pode ser facilmente compartilhado, arquivado, versionado, e executado. A interface gráfica intuitiva de Kepler (herdada do sistema *Ptolemy*), a qual pode ser vista na Figura 3.6, e sua modelagem orientado à atores (*actor-oriented modeling*) o tornaram uma ferramenta muito versátil para o gerenciamento do ciclo de vida de um *workflow* científico para ambos engenheiros de *workflows* e usuários finais.

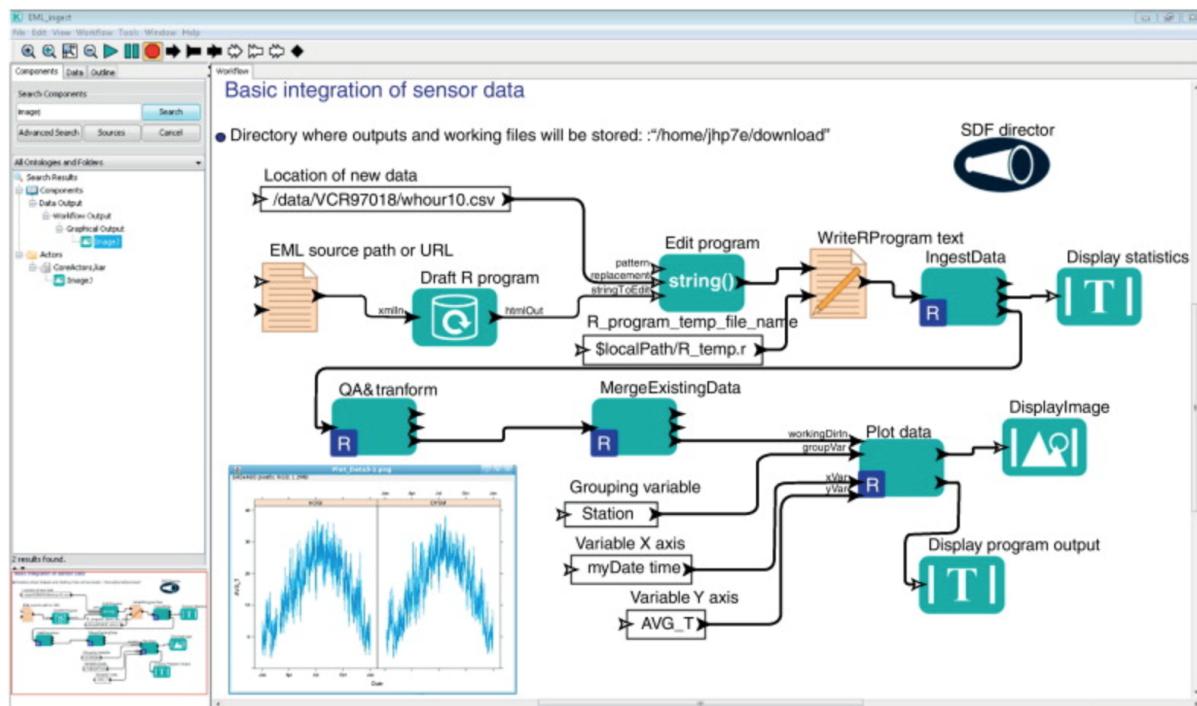


Figura 3.6: Interface gráfica do Kepler (retirado de [11]).

Os *workflows* gerados pelo sistema Kepler podem ser compartilhados a partir de uma representação XML (*Extended Markup Language*) gerado pela própria linguagem de modelagem do *Ptolemy* (*MOML - Modeling Markup Language*).

3.5 Vantagens na Utilização de um Sistema Gerenciador de *Workflows* Científicos

Workflows científicos são projetados para ajudar os cientistas a realizar experimentos computacionais eficientes, proporcionando um ambiente que simplifica o desenho experimental, implementação e documentação. A crescente utilização de ambientes e sistemas de *workflows* científicos é devido a uma série de vantagens que estes sistemas podem oferecer sobre abordagens alternativas, como:

- *Workflows* científicos automatizam tarefas repetitivas, permitindo que o usuário se concentre na ciência por trás do experimento, em vez de gastar tempo projetando o fluxo de dados e a gestão dos processos que compõem o *workflow*. Por exemplo, a repetição de um mesmo experimento com o intuito de se calibrar parâmetros, processo este que pode ser repetido de à milhares de vezes, pode ser alcançado muito mais facilmente através da utilização de sistemas de *workflows* do que com as abordagens convencionais de programação.
- *Workflows* científicos documentam, explicitamente, o processo científico que está sendo executado, o que pode levar à uma melhor comunicação entre, por exemplo, uma equipe de cientistas ou uma comunidade acadêmica, colaboração (por exemplo, o compartilhamento de *workflows* entre cientistas) e a reproduzibilidade dos resultados.
- Sistemas de *workflows* científicos podem ser usado para monitorar a execução de *workflows* e registrar a proveniência dos resultados de fluxo de trabalho. Proveniência, em particular, fornece uma forma de documentação que pode ser usado para validar e interpretar os resultados produzidos por (muitas vezes complexas) processos científicos.
- Sistemas de *workflows* científico muitas vezes podem otimizar e executar de forma mais eficiente os processos científicos, por exemplo, ao expor e explorar várias formas de paralelismo inerente nos processos científicos orientados por dados, bem como pelo emprego de outras técnicas de gestão eficiente dos recursos.

- Ambientes de execução de *workflows* incentivam a reutilização de artefatos de conhecimento (atores, fluxos de trabalho, conjuntos de dados, etc.) desenvolvidos ao se automatizar um processo científico, tanto dentro como entre disciplinas.

Capítulo 4

BioNimbuz

Este Capítulo tem como objetivo apresentar as características da plataforma de federação de nuvens BioNimbuz, no qual será desenvolvido o sistema gerenciador de *workflows* científicos projetado neste trabalho. Além disso, serão detalhados aspectos de sua arquitetura atual e o funcionamento dessa plataforma. Para isso, na Seção 4.1 serão apresentados os principais aspectos da plataforma BioNimbuz, seus objetivos e uma visão geral de seu funcionamento. A Seção 4.2 traz a arquitetura do BioNimbuz, apresentando suas camadas e serviços. Por último, a Seção ?? mostra a disposição dos serviços na plataforma e a integração com as nuvens computacionais, formando sua federação de nuvens.

4.1 Principais Características

O BioNimbuz é uma plataforma de execução de *workflows* que utiliza uma infraestrutura provida por uma federação de nuvens híbridas proposta originalmente por Saldanha [6], e que tem sido aprimorada constantemente em outros trabalhos [55] [56]. O BioNimbuz foi desenvolvido para suprir a demanda de plataformas de nuvens federadas, tendo em vista que a utilização de nuvens de forma isolada não atende, em muito casos, às necessidades de processamento e de armazenamento na execução das aplicações de Bioinformática.

A plataforma BioNimbuz permite a federação de nuvens de diversos tipos, tanto privadas quanto públicas. Dessa forma, cada provedor pode manter suas características e políticas internas, e oferece ao usuário transparência e ilusão de recursos infinitos. Assim, o usuário pode usufruir de diversos serviços sem se ter que gerenciar a infraestrutura que está sendo utilizada.

Um aspecto importante na arquitetura do BioNimbuz é a flexibilidade na inclusão de novos provedores, pois são utilizados mecanismos desenvolvidos em sua implementação inicial, chamados *plugins* de integração. Estes tem como objetivo ser a interface de comunicação entre o um provedor e os demais componentes da arquitetura BioNimbuz,

e também entre ele e os demais provedores da federação. Para que a comunicação seja feita com sucesso, o *plugin* precisa mapear as requisições vindas dos componentes do BioNimbuZ para ações correspondentes a serem realizadas na infraestrutura do provedor de serviço. Isso é fundamental para que requisitos como escalabilidade e flexibilidade possam ser alcançados.

Em sua implementação inicial, a comunicação realizada no BioNimbuZ era feita por meio de uma rede *Peer-to-Peer* (P2P) [57]. Porém, para alcançar os requisitos de escalabilidade e flexibilidade, percebeu-se a necessidade de alterar a forma de comunicação entre os componentes do BioNimbuZ, pois a utilização de uma rede de comunicação *Peer-to-Peer* (P2P) não estava mais suprindo as necessidades desses dois requisitos. É importante ressaltar que os outros objetivos propostos por Saldanha [6], tais como obter uma plataforma com grande poder computacional e armazenamento disponíveis, que suportasse uma diversidade de provedores homogêneos de infraestrutura e que fosse tolerante à falhas, foram mantidos e otimizados.

À fim de suprir essas necessidades, e também realizar a troca de mensagens de forma transparente, os trabalhos [55] [56] propuseram uma nova forma de comunicação, utilizando Chamadas de Procedimento Remoto (*RPC*) [58]. Isso possibilitou a chamada de procedimentos localizados remotamente sem que o usuário perceba. Nesse contexto, foi escolhido o *framework RPC* da Fundação Apache [38], o Avro [60]. Visando resolver também outros problemas de organização e coordenação dos serviços disponibilizados na plataforma BioNimbuZ, foi utilizado um serviço voltado à sistemas distribuídos, chamado ZooKeeper [59], também da Fundação Apache.

Nas Seções 4.1.1 e 4.1.2 serão mostrados detalhes do funcionamento dos sistemas Apache Avro e ZooKeeper.

4.1.1 Apache Avro

Um *framework RPC* é uma ferramenta utilizado para que máquinas possam realizar procedimentos em outras máquinas, de maneira remota. Esse conceito faz parte do paradigma de sistemas distribuídos e, possibilita ações como, por exemplo, a mesma instrução ser executada em conjuntos de dados diferentes em locais diferentes, ou uma máquina disparar a mesma requisição de procedimento em diversas máquinas geograficamente espalhadas.

O fato de ser um software livre, utilizando mais de um protocolo de transporte de dados em rede, e com suporte à mais de um formato de serialização de dados são algumas das vantagens na utilização do *framework* Avro. Quanto ao formato dos dados, ele dá suporte à dados binários e à dados no formato JSON [40]. Avro se baseia no protocolo HTTP [63] para realizar chamadas de procedimentos remotos.

O Projeto Avro foi criado com o objetivo de ser utilizado com um grande volume de dados. Esses dados dentro do *framework* Avro são representados como uma rica estrutura de dados com tipos primitivos (como inteiros, *strings*, caracteres, etc) e tipos complexos (como *unions*, *record*, *enum*, matrizes, mapas, etc), um formato de dados compacto, rápido e binário e a integração de forma simples com diversas linguagens de programação. Essas características fazem do Avro uma ferramenta muito eficiente e leve, não onerando assim o poder computacional de sistemas distribuídos.

Em comparação com outros sistemas de chamadas *RPC*, como Thrift [65] e Protocol Buffers [66], o Avro difere nos seguintes aspectos fundamentais:

- **Tipagem Dinâmica:** O Avro não requer que a sua representação dos dados seja gerada. Os dados são sempre acompanhados por um esboço (*schema*), que permite o processamento completo desses dados, sem a geração de código adicional, tipos de dados estáticos, etc. Isso facilita a construção de sistemas de processamento de dados e linguagens de genéricos.
- **Dados sem *overhead*:** Como o *schema* do Avro está presente no momento de leitura dos dados, informações consideravelmente menores precisam ser codificadas juntamente com os dados. O Avro fornece um esquema com dados binários que permite que cada dado seja escrito sem sobrecarga (*overhead*), resultando em uma codificação de dados mais compacta, e, consequentemente, um processamento de dados mais rápido.
- **Atribuição automática de ID's de campos:** Quando um *schema* muda, o antigo e o novo estão sempre presentes durante o processamento dos dados, de modo que as diferenças podem ser resolvidas simbolicamente, usando-se os nomes de campo atribuídos pelo próprio Avro, o que resulta em uma maior flexibilidade no desenvolvimento de soluções, tendo em vista que o usuário não fica preso à um *schema* antigo, ou legado.

Portanto, o *framework RPC* Avro foi escolhido como *middleware* de Chamadas Remotas de Procedimentos para o BioNimbuZ, por ser livre, flexível, eficiente e possuir integração com várias linguagens.

4.1.2 Apache ZooKeeper

ZooKeeper [59] é um serviço distribuído de coordenação de serviços, de código aberto geralmente utilizado para aplicações distribuídas. Sua forma de funcionamento é simples e permite que os processos distribuídos sejam coordenados por meio de um espaço de

nomes (*namespace*) hierárquico e compartilhado, e sua organização é similar a de um sistema de arquivos padrão.

Conforme mostrado na Figura 4.1, o ZooKeeper é composto por diversos servidores. O ZooKeeper possui um algoritmo de eleição que faz com que um dos servidores seja escolhido o líder, e os outros passam a ser chamados de seguidores (*followers*). Caso o líder desconecte-se do ZooKeeper, por exemplo por uma possível queda do servidor, o algoritmo de eleição é executado novamente para um dos servidores restantes se torne o novo líder.

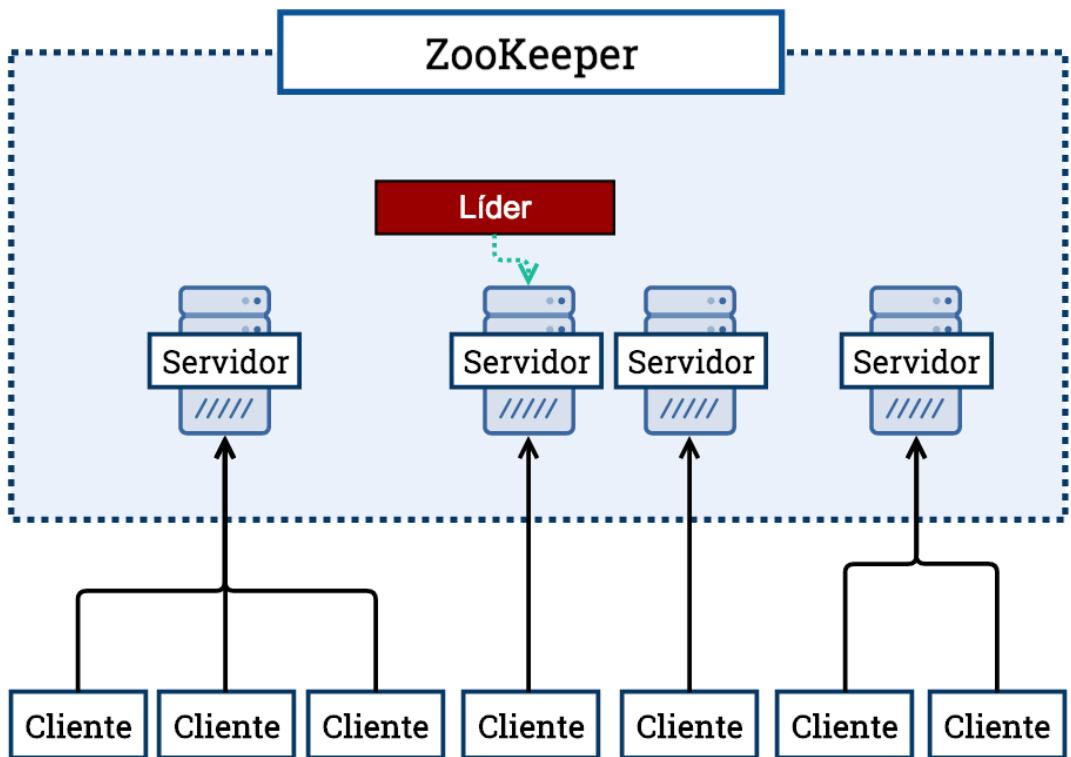


Figura 4.1: Modelo de serviço utilizado no ZooKeeper (adaptado de [59]).

Com relação aos dados mantidos no ZooKeeper, eles são dispostos em estruturas chamadas ***znodes***, e estes podem ser comparados à arquivos e diretórios dentro de um sistema de arquivos tradicional.

Ao contrário de um sistema de arquivo comum, que é projetado para o armazenamento dos dados, as informações gravadas no ZooKeeper são pequenos (armazenamento máximo limitado à 1 Megabyte) e são mantidos na memória, o que faz com que ele atinja uma baixa latência e altas taxas de transferência. Assim como os servidores coordenados pelo ZooKeeper são replicados para garantir a tolerância à falhas, seu próprio serviço também é replicado em diversas máquinas, a fim de se manter sempre ativo, independente de quedas e falhas.

As máquinas coordenadas (clientes) conectam-se a um único servidor ZooKeeper. O cliente mantém uma conexão TCP na qual envia requisições, recebe respostas e envia ***heartbeats***, que são notificações periódicas enviadas do cliente ao servidor para avisá-lo que está *online*, e, caso o tempo de resposta ultrapasse um determinado *timeout* (configurado como 2 segundos no ZooKeeper), o servidor assume que o cliente está *offline*. No caso de queda da conexão TCP entre o cliente e o servidor, o cliente tentará se conectar a outro servidor.

O espaço de nomes implementado pelo ZooKeeper é muito parecido com o de um sistema de arquivos padrão, ou seja, um caminho absoluto é uma sequência de caminhos relativos separados por uma barra (/). Cada nó no espaço de nomes do ZooKeeper é identificado por um caminho. A Figura 4.2 mostra um exemplo de uma possível estrutura hierárquica de *znodes* do ZooKeeper.

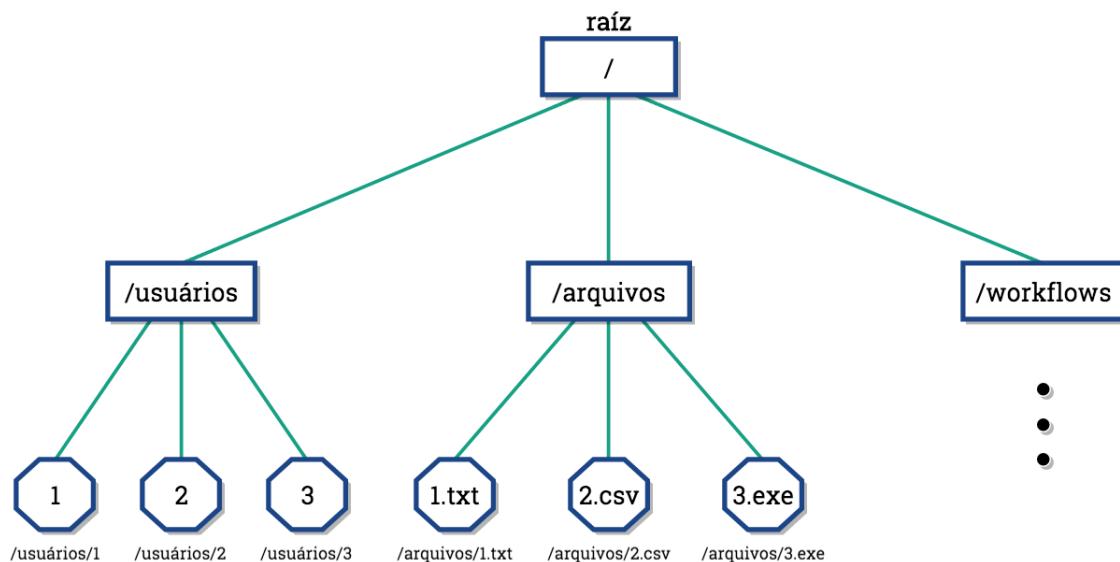


Figura 4.2: Exemplo de estrutura de nós do ZooKeeper.

Com relação à notificação de eventos aos clientes, o ZooKeeper implementa o conceito de ***watchers***. Um *watcher* é um aviso e serve para notificar alguma entidade do sistema que um *znode* foi alterado. Dessa forma, para que essa entidade possa notar uma atualização em um *znode*, é necessário adicionar um *watcher* à ele. Assim, dada uma alteração em um nó, apenas quem inseriu um *watcher* nele será notificado.

Soma-se às características apresentadas, o fato de o mesmo apresentar uma interface de programação (*API - Application Programming Interface*) muito compacta, suportando operações simples, como:

- ***create*:** Cria um *znode* na estrutura do ZooKeeper;
- ***delete*:** Exclui um *znode*;

- ***getData***: Obtém os dados de um *znode*;
- ***setData***: Grava dados em um *znode*;
- ***getChildren***: Recupera uma lista contendo os nós filhos de um determinado *znode*;
- ***exists***: Verifica a existência de um nó dado um caminho.

Devido às características apresentadas anteriormente, o ZooKeeper então foi escolhido para coordenar os serviços e processos da plataforma BioNimbuZ.

Na Seção 4.2 será apresentada a arquitetura do BioNimbuZ anterior ao desenvolvimento do sistema gerenciador de *workflows* científicos, objetivo do presente trabalho.

4.2 Arquitetura do BioNimbuZ

A arquitetura apresentada nesta Seção representa a antiga arquitetura do BioNimbuZ. Uma nova arquitetura foi proposta no presente trabalho e será apresentada e descrita no Capítulo 5.

O BioNimbuZ utilizava uma arquitetura hierárquica, conforme mostrado na Figura 4.3 e possuía três camadas principais: interface, núcleo e infraestrutura, todas gerenciadas pelo ZooKeeper. Suas principais responsabilidades eram as seguintes:

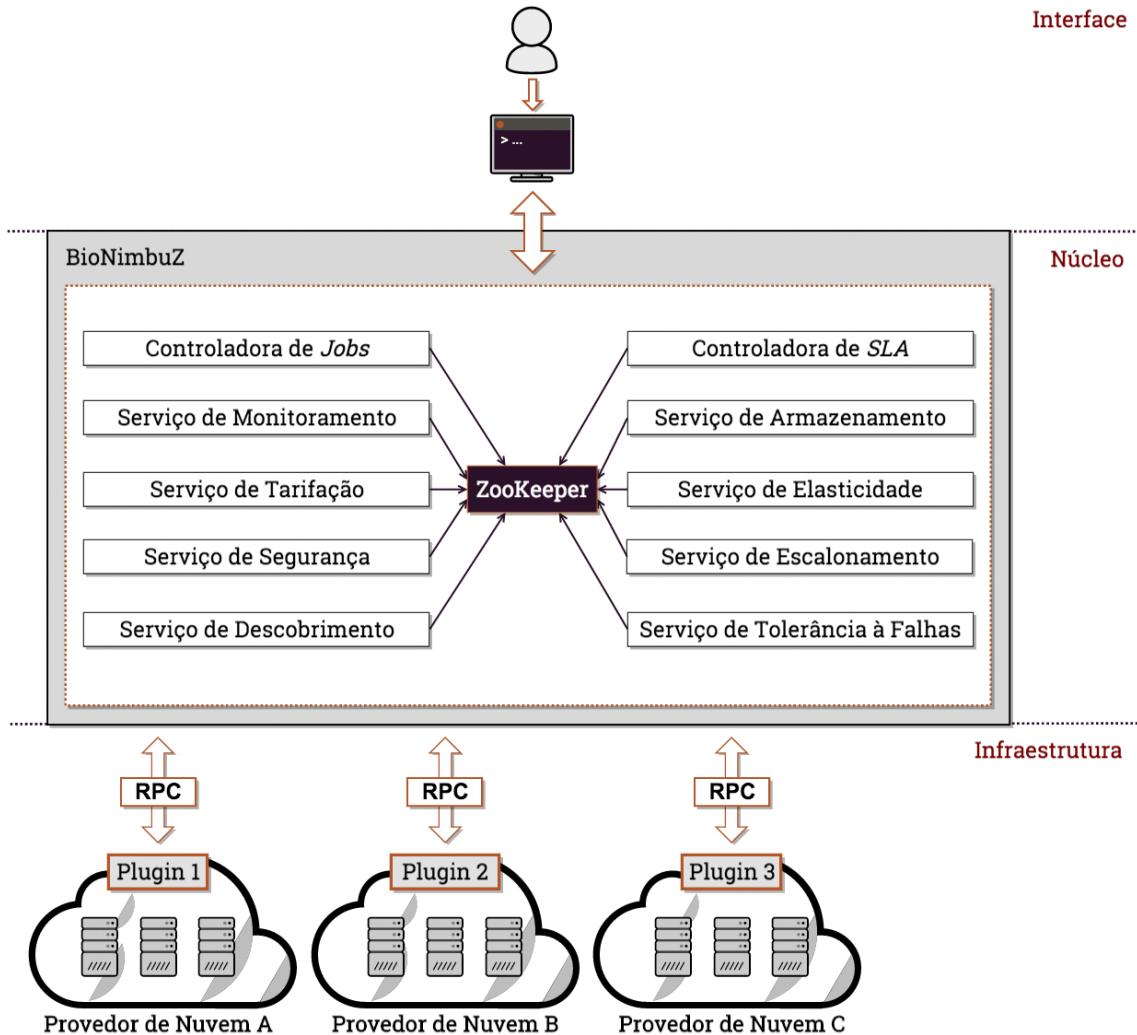


Figura 4.3: Antiga arquitetura do BioNimbuZ.

- **Camada de interface com o usuário:** Seu principal objetivo era controlar a interação com o usuário, recebendo e processando seus comandos (utilizando um *terminal*) e repassando-os ao núcleo para que fossem devidamente executados;
- **Camada de Núcleo:** Responsável por gerenciar todos os serviços presentes na plataforma (os quais serão descritos na Seção 4.2.2) e também por gerir toda a federação de nuvens utilizada pelo BioNimbuZ.
- **Camada de Infraestrutura:** Sua principal responsabilidade é prover uma interface de comunicação entre o BioNimbuZ e os provedores de nuvens. Consiste em todos os recursos computacionais que os provedores colocam à disposição da federação somados aos seus respectivos *plugins* de integração.

As Seções 4.2.1, 4.2.2 e 4.2.3 descrevem os detalhes específicos das camadas apresentadas até aqui.

4.2.1 Camada de interface com o usuário

Camada responsável por interagir com o usuário, recebendo seus comando e devolvendo o resultado do processamento realizado pelo núcleo, podendo ser desenvolvida de diversas maneiras: páginas *web*, linhas de comando, interfaces gráficas, sistemas gerenciadores de *workflows*, etc. Para tal, esses serviços de interação precisam conectar-se à Internet e se comunicar com a camada abaixo da arquitetura, o núcleo do BioNimbuz.

Os comandos enviados para o núcleo podiam realizar uma série de tarefas, tais como: listas os arquivos armazenados na federação, realizar o *upload* de arquivos, submeter um *job* para ser executado, etc. Assim, nota-se que essa camada era responsável por mostrar ao usuário uma forma de se comunicar com o BioNimbuz.

4.2.2 Camada de Núcleo

Esta camada tem a atribuição de agregar os serviços providos pelo BioNimbuz, servir como núcleo processador dos comandos provenientes da camada acima (camada de interface com o usuário, descrito na Seção 4.2.1) e gerenciar a infraestrutura de federação de nuvens, que é a camada abaixo (camada de infraestrutura, descrita na Seção 4.2.3).

O chamado Núcleo do BioNimbuz possuía oito serviços principais: serviço de descobrimento, serviço de monitoramento, serviço de tolerância a falhas, serviço de escalonamento, serviço de segurança, serviço de armazenamento, serviço de elasticidade e serviço de tarifação. Além destes, possuía mais duas controladoras: controladora de *jobs* e controlador de *SLA*. A fim de exercerem suas funcionalidades, os serviços e as controladoras interagiam entre si por meio do gerenciamento de troca de mensagens provido pelo ZooKeeper. No que tange aos serviços e controladoras, abaixo segue uma descrição de suas respectivas funções.

Controladora de Jobs

É responsável por fazer a ligação entre a camada de interface com o usuário e o núcleo do BioNimbuz. Realiza o controle de acesso dos usuários ao submeter *jobs* aos recursos disponíveis na federação controlada pelo BioNimbuz. Para realizar a verificação das credenciais do usuário, a Controladora de *Jobs* acessa o Serviço de Segurança. Além disso, a Controladora de *Jobs* é responsável por gerenciar os pedidos dos vários usuários, de forma a fazer o controle por usuário, mantendo os resultados para posterior consulta.

Controladora de SLA

Atualmente, há um esforço muito grande por parte dos provedores de nuvens computacionais para que a qualidade dos serviços (*Quality of Service*) prestados percebidos pelo usuário sejam os melhores possíveis. E isso apenas é possível a partir de acordos de níveis de serviço (*Service Level Agreement*) assinados entre os usuários e os provedores.

Nesse contexto, o BioNimbuZ também tem essa preocupação com a qualidade dos serviços prestados aos seus usuários. Assim, quando um usuário submete um *job* para ser executado pela federação do BioNimbuZ, por meio da interface com o usuário, ele deve preencher um *template* de SLA, que representa, entre outras coisas, os parâmetros de *QoS* desejados pelo usuário. Estes parâmetros podem descrever desde requisitos funcionais, como número de núcleos de CPU, tamanho de memória, tamanho de armazenamento, até requisitos não funcionais, como custo a pagar e taxa de transferência.

Portanto, a Controladora de SLA tem a responsabilidade de verificar se os requisitos especificados no *template* preenchido pelo usuário podem ser suportados pela federação de nuvens naquele dado momento e, para isso, a controladora utiliza os dados de SLA informados pelo *plugin* de integração de cada provedor.

Serviço de Tolerância à Falhas

Em ambientes de computação em nuvem, falhas em máquinas são fatos comuns e ocorrem com frequência. Logo, uma federação de nuvens deve sempre levar em consideração requisitos de tolerância à falhas e desenvolvimento de ambientes de alta disponibilidade.

O Serviço de Tolerância à Falhas tem como objetivo principal garantir que todos os serviços do BioNimbuZ estejam sempre disponíveis, e em caso de falhas, inicie alguma ação de recuperação. Essa recuperação exige que todos os componentes do sistema que foram afetados pela falha voltem ao seu estado anterior. O serviço de tolerância a falhas deve atuar de forma distribuída na plataforma, e estar presente em outros serviços, monitorando seu estado.

No Serviço de Armazenamento, por exemplo, quando um alerta de indisponibilidade de um recurso é lançado por um *watcher*, é iniciada uma rotina de recuperação para os arquivos que este recurso continha. Como os arquivos são replicados na federação, a recuperação ocorre de forma a identificar quais réplicas foram perdidas, realizando uma nova duplicação em outro servidor do BioNimbuZ. A tolerância à falhas no BioNimbuZ está intimamente ligada aos *watchers* do ZooKeeper, pois são estes que disparam os alertas de indisponibilidade, notificando o sistema sobre problemas em algum recurso da federação.

Serviço de Escalonamento

É o serviço responsável por receber o pedido de execução de algum *job* e distribuí-las dinamicamente para os provedores disponíveis, dividindo-as em instâncias menores - chamadas *tasks*. O serviço de escalonamento também é responsável por acompanhar toda a execução de um *job*, e manter um registro das execuções já escalonadas.

Para realizar a distribuição de tarefas na federação, algumas métricas são levadas em consideração, como latência, balanceamento de carga, tempo de espera e capacidade de processamento dos recursos disponíveis, entre outras, visando atender o que foi determinado no acordo de SLA.

Serviço de Segurança

Segurança em nuvens federadas é uma área de estudo em constante evolução, e este serviço deve trabalhar em diversos pontos para fornecer um serviço efetivamente seguro, mitigando possíveis pontos de falhas no sistema.

Dois pontos constituem o cerne deste serviço: autenticação e autorização. O primeiro, está focado em saber se o usuário que está tentando acessar algum recurso na federação é quem ele realmente diz ser. Depois de estar autenticado no sistema, o segundo passo é a autorização, que consiste em verificar se o usuário pode realizar aquela ação que está demandando.

Muitos outros aspectos podem ser abordados por este serviço, como a criptografia de mensagens, para garantir a confidencialidade na troca de informações entre provedores, por exemplo, e também a verificação de integridade de arquivos, de modo que seja possível garantir que um arquivo não seja alterado por fatores externos à federação, como por exemplo um componente de rede gerando erros nos dados trafegados.

Serviço de Armazenamento

Responsável pela estratégia de armazenamento dos arquivos que são utilizados ou mantidos pelo BioNimbuZ, o serviço deve decidir sobre a distribuição dos arquivos entre os diferentes provedores da federação. Para isso, o serviço de armazenamento comunica-se com o serviço de descobrimento para obter acesso as informações sobre a federação. Com isso, o serviço saberá as condições atuais de armazenamento de cada um dos provedores que faz parte da federação.

O armazenamento deve ocorrer de forma eficiente, para que as aplicações possam utilizar os arquivos com o menor custo possível. Este custo é calculado utilizando-se algumas métricas, tais como, latência de rede, distância entre os provedores e capacidade de armazenamento. Também é adotada a replicação dos arquivos em mais de um provedor,

também visando a diminuição de custos envolvidos.

Serviço de Elasticidade

Sua função é, dinamicamente, aumentar ou diminuir o poder computacional total da federação. Para tal, gera novas instâncias ou desliga instâncias ociosas de máquinas virtuais, ou então reajusta as configurações de CPU, de memória, de largura de banda, entre outros recursos que são disponibilizados pelos provedores de nuvem, a fim de se obter uma melhor utilização da infraestrutura disponível.

A elasticidade pode ser vertical, quando há um redimensionamento das configurações da máquina virtual, como também pode ser horizontal, quando é altera-se o número de máquinas virtuais instanciadas nos provedores.

Serviço de Tarifação

Tem a responsabilidade de calcular o quanto os usuários devem pagar pela utilização dos serviços oferecidos na plataforma BioNimbuZ. Para que isto seja possível, este serviço se mantém em constante contato com o serviço de monitoramento, para obter informações, tais como tempo de execução e quantidade de máquinas virtuais alocadas das tarefas que foram executadas por determinado usuário. Dessa forma é possível verificar quais recursos foram alocados para determinada carga de trabalho submetida pelo usuário, obtendo-se assim, o preço devido.

Serviço de Descobrimento

Responsável por identificar e manter informações como poder computacional, capacidade de armazenamento e latência de rede, à respeito dos provedores de nuvem presentes na federação. Também mantém detalhes sobre parâmetros de execução e arquivos de entrada e de saída dos softwares disponíveis para execução pelo usuário.

Toda vez que um provedor é incluído na federação de nuvens, seus dados são gravado em um novo *znode* no ZooKeeper. Assim, para qualquer outro serviço obter informações acerca dos provedores presentes na federação, basta realizar uma consulta simples no ZooKeeper. Sempre que as informações acerca de um provedor são atualizadas (por exemplo com a diminuição do poder computacional ou falta de espaço de armazenamento), outros serviços são notificados, a partir dos *watchers* que foram adicionados. Dessa forma, todo o sistema tem acesso à essas informações.

Serviço de Monitoramento

Este serviço monitora a nuvem federada verificando as aplicações e os *jobs* para a execução. Ao receber um pedido de execução de um *job*, vindo da Controladora de *Jobs*, o Serviço de Monitoramento identifica se o software computacional relacionado àquele *job* está disponível em algum provedor de serviço, e então redireciona o pedido para o Serviço de Escalonamento, o qual processa o pedido garantindo que todas as requisições sejam devidamente atendidas e executadas.

A fim de retribuir informações acerca da execução das tarefas, mensagens periódicas são enviadas para os recursos gerenciados pelo BioNimbuz. Além disso, notifica a Controladora de *Jobs* sobre o estado da execução de determinado *Job*.

Dessa forma, é por meio dessas mensagens periódicas que o Serviço de Monitoramento é capaz de verificar se uma execução é finalizada com sucesso sem violar os parâmetros de SLA acordados com o usuário.

4.2.3 Camada de Infraestrutura

A camada de infraestrutura consiste nos recursos computacionais disponibilizados na federação por meio dos provedores de nuvem, visíveis ao BioNimbuz através dos *plugins* de integração. Essa camada provê meios para que haja a comunicação entre o BioNimbuz e os provedores presentes na federação, mapeando os comandos vindos da arquitetura do BioNimbuz para comandos específicos de cada provedor.

Capítulo 5

Sistema Gerenciador de Workflows Científicos para o BioNimbuZ

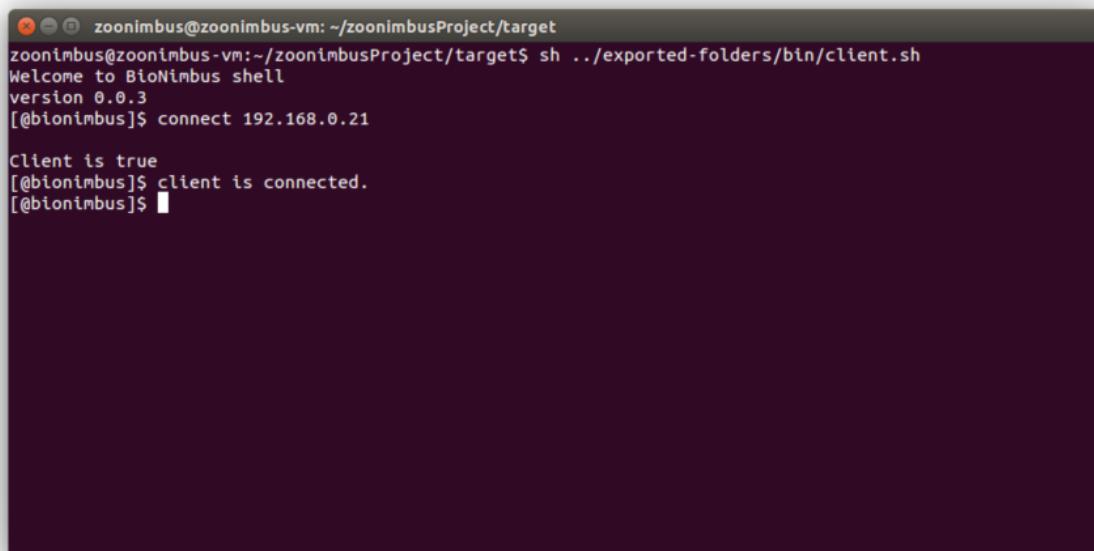
Este capítulo trata das etapas do desenvolvimento do Sistema Gerenciador de *Workflows* Científicos para a plataforma BioNimbuZ, e as alterações necessárias para que ele fosse suportado pelo atual estado do sistema. Primeiramente, na Seção 5.1 é apresentado o funcionamento do BioNimbuZ antes do desenvolvimento do sistema gerenciador de *workflows*, o qual era executado através de um *terminal*. Os novos requisitos do sistema são apresentados na Seção 5.2. Com esses requisitos, foi proposta uma nova arquitetura para o sistema, a qual é apresentada na Seção 5.3. A Seção 5.4 mostra os detalhes de como a Aplicação *Web* foi desenvolvida para dar suporte ao Sistema Gerenciador de *Workflows*. Na Seção 5.5, são expostos os motivos da contrução de um novo componente da arquitetura de *software*, chamada Camada de Comunicação, responsável por realizar a troca de mensagens entre o Núcleo do BioNimbuZ e a Aplicação *Web* utilizando *webservices REST*. Por último, na Seção 5.6 são apresentadas as alterações feitas no BioNimbuZ para suportar esse novo modo de acesso, composto pela aplicação *web* e a camada de comunicação.

5.1 Do *terminal* para a Interface *Web*

O BioNimbuZ primeiramente projetado e implementado por Hugo Saldanha [3] era acessado via *terminal*. Nele, o usuário digitava uma série de comandos, os quais eram processados e executados pelo servidor do BioNimbuZ, e então recebia de volta o resultado daquele comando. Essa forma não era nem intuitiva nem amigável para o usuário, pois exigia que o mesmo conhecesse a execução de comandos através de um *terminal*. Para, por exemplo, submeter uma sequência de passos, ou *jobs*, formando assim um *workflow*, o usuário deveria executar diversos comandos em sequência, ou seja, o usuário era quem

gerenciava todos os passos de seu *workflow*. Outro ponto negativo dessa abordagem era o fato de que o usuário não poderia acessar o sistema, ver os *jobs* submetidos ou visualizar seus arquivos de qualquer lugar, pois era necessário realizar o *download* do executável do BioNimbuz em sua máquina local. Uma das vantagens da nova abordagem de sistema baseado em tecnologias *web* é a acessibilidade provida pela *Internet*. Dessa forma, o usuário pode, por exemplo, iniciar um *workflow* em seu laboratório, concluí-lo e submetê-lo em casa e verificar sua execução pelo celular.

Na abordagem anterior, baseada em *terminal*, era necessário realizar o *download* do *JAR* (*Java Archive*) do projeto e executá-lo via linha de comando. A Figura 5.1 mostra a interface percebida pelo usuário ao se executar o BioNimbuz no sistema operacional Linux:



```

zoonimbus@zoonimbus-vm: ~/zoonimbusProject/target
zoonimbus@zoonimbus-vm:~/zoonimbusProject/target$ sh ..../exported-folders/bin/client.sh
Welcome to BioNimbuz shell
version 0.0.3
[@bionimbus]$ connect 192.168.0.21

Client is true
[@bionimbus]$ client is connected.
[@bionimbus]$ █

```

Figura 5.1: *Terminal* do BioNimbuz.

Dessa forma, utilizando a interface mostrada na Figura 5.1, para o usuário submeter um *workflow* com dois passos, a seguinte sequência de comandos deveria ser executada:

1. Executar o comando ***connect*** para se conectar ao BioNimbuz, passando o IP da máquina servidora como argumento do comando;
2. Enviar cada arquivo a partir do comando ***upload***, passando o caminho do arquivo à ser enviado;
3. Com todos os arquivos enviados, submeter o comando ***start*** passando o identificador do primeiro serviço (nesse caso, serviço se refere ao software computacional a ser executado naquele *job* e uma lista contendo os serviços disponíveis era mostrada a

partir do comando ***services***), a lista de arquivos de entrada e a lista de arquivos de saída;

4. A partir do início de um *job* era possível verificar seu *status* com o comando ***status***, informando o número do *job*.

Os comandos acima descrevem o passo a passo da execução de apenas um *job*. Para um segundo passo, seria necessário submeter todos esses comandos novamente, com os dados de entrada do segundo *job*. Assim, o método acima, via *terminal*, não dava suporte à execução de *workflows*, pois não era possível ligar elementos, formar dependências e controlar o fluxo de dados de maneira única. Eram necessárias diversas iterações da sequência de comandos acima descrita para se ter uma ligação entre os *jobs* que o usuário quisesse executar. Dessa forma, o usuário deveria ter informações que, nem sempre, eram de seu conhecimento, como o *IP* da máquina servidora do BioNimbuz, o caminho do arquivo ou os argumentos de um serviço provido.

A próxima Seção apresenta os requisitos que foram verificados e implementados, com o objetivo de se desenvolver o sistema gerenciador de *workflows* científicos proposto no presente trabalho.

5.2 Novos Requisitos

De acordo com as deficiências notadas do BioNimbuz, foi verificado que o acesso aos serviços providos pelo BioNimbuz via *terminal* poderia ser melhorado caso alguns requisitos fossem implementados, tais como:

- **Acesso via *Internet*:** Prover de um meio de acesso à plataforma do BioNimbuz à qualquer pessoa conectada à *Internet*;
- **Composição do *workflow* de maneira gráfica:** Um *workflow* científico é composto por uma sequência de passos interligados, formando uma cadeia de execução com entradas e saídas. A forma mais viável de projetá-lo seria a partir de uma interface que possibilitasse o desenho de um *workflow*;
- **Acesso à plataforma utilizando-se usuário e senha:** O acesso via *terminal* não dava suporte à múltiplos usuários. Cada usuário deveria ter o BioNimbuz instalado em seu computador e submetia sua sequência de *jobs* da sua máquina para o servidor do BioNimbuz. Assim, por questões de segurança, fez-se necessário o desenvolvimento de um método de controle de usuários, verificando a autenticidade e a autorização no momento do *login*;

- **Facilidade no envio dos arquivos:** O método de envio dos arquivos poderia ser facilitado, caso fosse desenvolvido uma maneira gráfica de enviá-los ao BioNimbuz;
- **Controle do *status* dos *workflows* submetidos:** Método de visualização do *status* de cada *workflow* que o submeteu para ser processado pela plataforma do BioNimbuz.
- **Separação entre aplicação cliente e aplicação servidora:** Evitaria um ponto único de falhas, pois as aplicações poderiam ser executadas em ambientes diferentes, em infraestruturas diferentes e, no caso de falha de uma delas, a outra não seria prejudicada. Com esse requisito também seria possível reduzir custos, pois a infraestrutura necessária para executar a aplicação cliente não tem a necessidade de ser tão robusta quanto a aplicação servidora, pois apenas esta executaria os serviços computacionais provados pelo BioNimbuz.

Na próxima Seção, será apresentada a nova arquitetura, contendo os novos componentes de software e seu detalhamento.

5.3 Proposta de uma Nova Arquitetura

Tendo em vista os requisitos levantados, descritos na Seção 5.1, e para que o BioNimbuz se tornasse acessível através da Internet, foi necessário a implementação de um método de acesso baseado em tecnologias *web*, substituindo o método antigo via *terminal*. Para isso, foi desenvolvida um sistema gerenciador de *workflows* basado em interface gráfica utilizando a linguagem de programação Java, em conjunto com *frameworks web*. Para tal, uma nova proposta de arquitetura foi concebida, englobando as modificações necessárias (principalmente, na parte de comunicação entre os seguintes componentes do sistema: Aplicação *Web* e Núcleo).

Nessa nova arquitetura, o BioNimbuz não será composto apenas pelas três camadas projetadas anteriormente [3] (Aplicação, Núcleo e Infraestrutura). Será acrescentado um novo componente chamado Camada de Comunicação, devido à importância para a troca de mensagens entre a aplicação *web* e o núcleo do BioNimbuz nessa nova abordagem. Assim, a Figura 5.2 apresenta essa nova proposta de arquitetura e uma breve descrição dos componentes (sua explicação completa será apresentada na próxima Seção, 5.3):

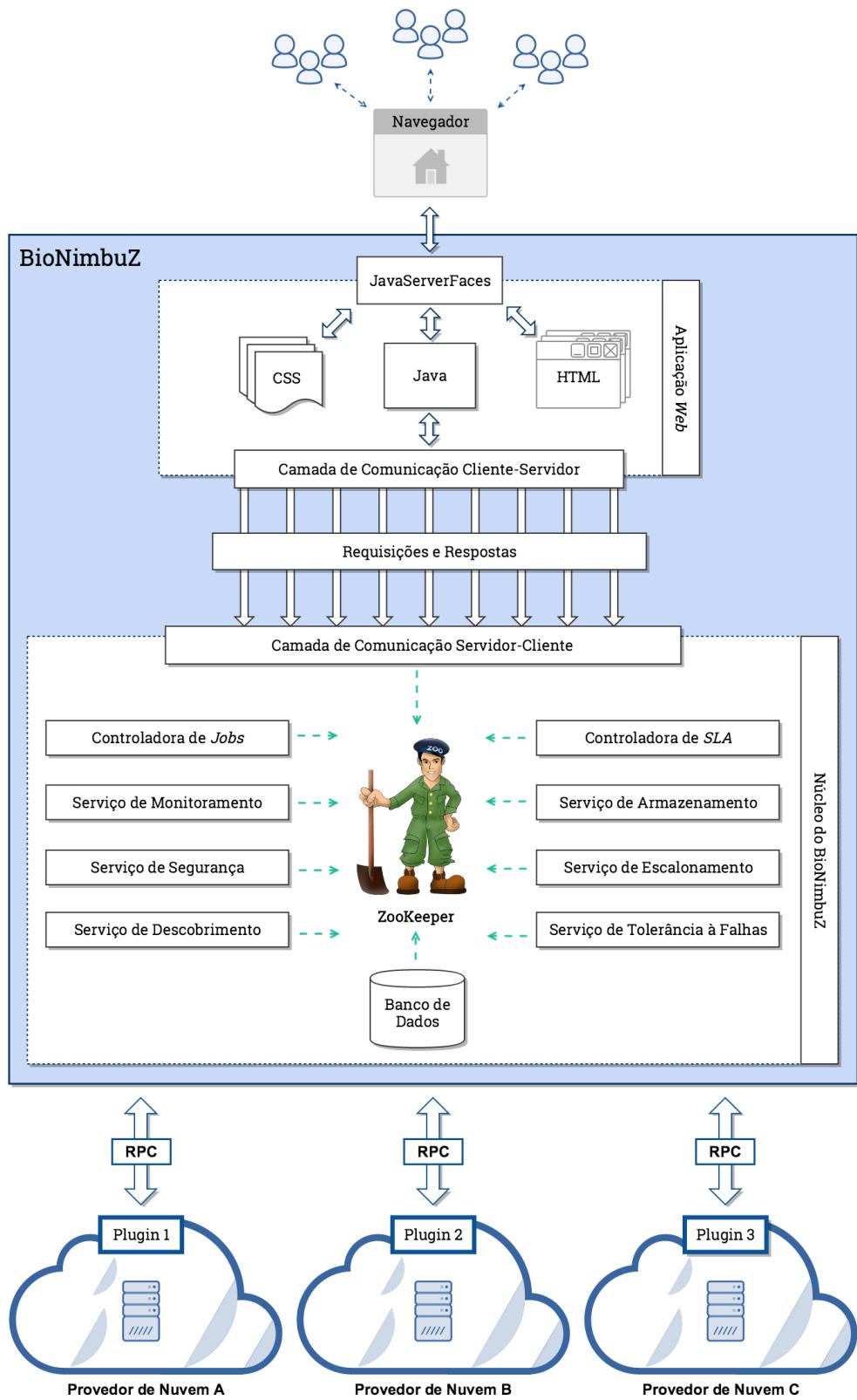


Figura 5.2: Nova Arquitetura do BioNimbuz.

- **Aplicação Web:** Implementa o sistema gerenciador de *workflows* científicos e utiliza a camada de comunicação para enviar requisições e receber respostas do núcleo.
- **Camada de Comunicação:** Implementa a troca de mensagens entre o núcleo e a aplicação *web* utilizando *webservices*, disparando requisições da aplicação *web* para o núcleo e recebendo respostas do núcleo para a aplicação.
- **Núcleo:** Responsável pelo armazenamento de arquivos, escalonamento de *jobs*, execução de *workflows*, acesso ao banco de dados, descobrimento dos provedores e o gerenciamento de possíveis falhas.
- **Infraestrutura:** Composta pelos provedores de serviço utilizados pelo BioNimbuz e que compõem sua federação de nuvens.

As Seções 5.3, 5.4 e 5.5 trazem detalhes das camadas supracitadas.

5.4 Aplicação Web

Este componente comprehende o Sistema Gerenciador de *Workflows* Científicos e deve prover meios para que usuários possam se logar no sistema BioNimbuz através de uma interface gráfica acessível pela Internet e utilizem suas funcionalidades como envio de arquivos (que serão utilizados como entrada dos *workflows* criados pelo usuário), exclusão de arquivos, visualização dos dados de saída gerados pela execução de seus *workflows*, por exemplo. Pela interface, o usuário também deve ser capaz de criar e projetar seus *workflows* de maneira gráfica, ligando passos, indicando dependências, incluindo argumentos e indicando quais arquivos de entrada serão utilizados.

Com os requisitos descritos na Seção anterior, foi projetada a implementação desse componente através da linguagem de programação Java e *frameworks Web* (bibliotecas de *software* que facilitam a implementação de funcionalidade relacionadas à sistemas voltados para *internet*).

Na implementação de uma aplicação *Web*, o desenvolvimento é dividido em camadas e utiliza-se, normalmente, o padrão de projeto *MVC* (*Model-View-Controller*) [?]. O padrão *MVC* é utilizado no desenvolvimento de interfaces pela sua característica de estar intimamente mapeado nos aspectos principais de uma interface:

- **Visualização:** Fornece a apresentação dos dados presentes no Modelo.
- **Controlador:** Recebe os dados e comandos do usuário e determina o que isso significa para o Modelo.

- **Modelo:** Contém todos os dados e estados persistidos, geralmente em Bancos de Dados.

Dessa forma, o padrão *MVC* contém exatamente esse três componentes: o ***Model*** (Modelo), o ***Controller*** (Controlador) e o ***View*** (Visualização). A interação entre esses componentes pode ser descrita da seguinte maneira:

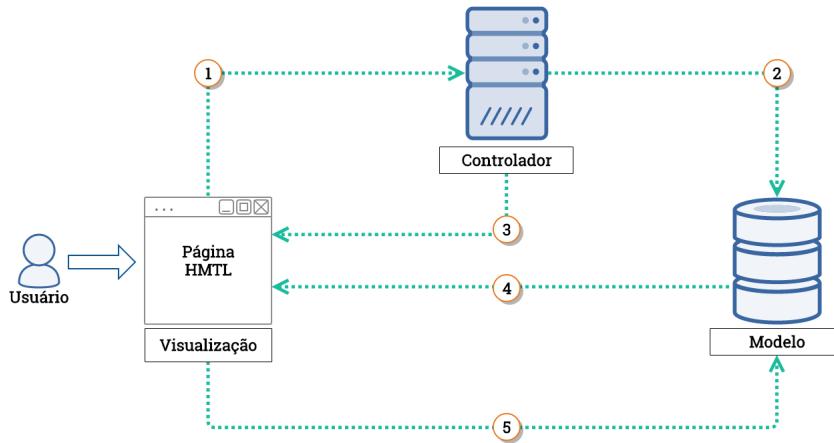


Figura 5.3: Modelo *Model-View-Controller*.

Na Figura 5.3, o fluxo pode ser descrito como se segue:

- (1): O usuário realiza uma ação na página HTML.
- (2): O Controlador processa o comando e muda o estado de algum objeto no Modelo.
- (3): O Controlador envia um comando à Visualização para modificar os dados exibidos.
- (4): O Modelo notifica a Visualização de que os dados foram modificados.
- (5): A Visualização envia uma requisição ao Modelo pedindo o estado de algum dado.

Nesse contexto, diversos *frameworks* surgiram voltados para o desenvolvimento de aplicações *web*, tais como: *Spring MVC* [41], *Struts* [42], *JSF (JavaServerFaces)* [32], *Play* [43], entre outros. Para esse projeto, o *framework* escolhido foi o *JSF*.

O *JavaServerFaces* é uma especificação Java para a construção de interfaces de usuário baseadas em componentes para aplicações *web*. Possui um modelo de programação dirigido a eventos, abstraindo os detalhes da manipulação e organização dos componentes, permitindo que o programador se concentre na lógica da aplicação.

Dessa forma, utilizando-se o padrão *MVC* descrito anteriormente, foi projetada e implementada a aplicação *web* contendo os componentes necessários para o desenvolvimento do sistema gerenciador de *workflows* científicos.

5.4.1 Camada de Modelo

5.4.2 Camada de Visualização

Primeiramente, foi desenvolvida a camada de visualização do modelo *MVC*. Para tal, foram implementadas páginas em **HTML** [1] (*Hypertext Markup Language*) utilizando componentes visuais da biblioteca *Primefaces* [2]. O framework **Primefaces** possibilita utilizar componentes visuais ricos e de maneira simples, sem se preocupar em ajustar parâmetros visuais ou de programação.

Isso significa que, utilizando o *Primefaces*, não é necessário desenvolver os códigos de estilo *CSS* [12] (*Cascading Style Sheets* - especificação que define como os elementos que compõem uma página, um documento ou aplicação *web* serão exibidos). Também não é necessário desenvolver código *JavaScript* [10] (linguagem de programação baseada em *scripts*, utilizada na parte cliente de páginas *web*, isto é, código executado no navegador do usuário), descrevendo a interação entre os elementos visuais. Essas duas características do *Primefaces* facilitam o desenvolvimento de páginas *web* interativas e dinâmicas.

A Seção ?? trata da Camada de Controle, a qual também compõe o Modelo *MVC*.

5.4.3 Camada de Controle

Conforme o que foi exposto previamente, a camada de controle tem a responsabilidade de interpretar os dados enviados pelo usuário, a partir da camada de visualização (que neste projeto foi desenvolvida a partir de páginas *HTML*), sinalizando ao sistema o que aqueles comandos representam.

Assim, foram desenvolvidas classes Java controladoras para tratar as requisições disparadas nas páginas *web* de forma que houvesse uma ligação entre as classes Java e suas respectiva páginas *HTML*. Essa correspondência é mostrada na Tabela 5.1 e descrita a seguir:

Classe Java	Página relacionada
SessionBean	Página de <i>login</i>
SignUpBean	Página de cadastro de novos usuários
FileUploadBean	Página relativa à <i>upload</i> de arquivos
DeleteFileBean	Página de deleção de arquivos do usuário
WorkflowComposerBean	Página de composição, <i>design</i> e submissão de <i>workflows</i>
WorkflowStatusBean	Página para visualização do <i>status</i> dos <i>workflows</i> gerenciados pelo usuário
WorkflowHistoryBean	Contém o histórico de execução de um determinado <i>workflow</i>

Tabela 5.1: Relação entre classes Java e páginas *HTML*.

- **SessionBean:** Esta classe tem como função principal realizar ações relacionadas à sessão do usuário no sistema, como por exemplo executar o *login* e o *logout*. Também deve manter as informações acerca do usuário (dados pessoais, lista de arquivos, lista de *workflows* submetidos, etc) para consulta por outros componentes do sistema, funcionando, portanto, como um repositório de dados de usuários;
- **SignUpBean:** Controla o fluxo do sistema quando um usuário deseja se cadastrar na plataforma, isto é, verifica se o usuário já foi cadastrado previamente, e, em caso negativo, realiza sua inclusão, persistindo-o no banco de dados;
- **FileUploadBean:** Responsável por enviar os arquivos dos usuários à plataforma BioNimbuZ. Também realiza o tratamento do arquivo, verificando aspectos tais como:
 1. **Extensão:** Verifica se a extensão do arquivo satisfaz àquela esperada pelo sistema;
 2. **Tamanho:** Certifica-se de que o tamanho do arquivo enviado não ultrapassa o limite estipulado;
 3. **Nome de Arquivo:** Verifica se aquele nome de arquivo já não foi previamente enviado ao sistema.
- **DeleteFileBean:** Sua função é deletar um arquivo de usuário, enviando a requisição de exclusão para o núcleo do BioNimbuZ e mostrando o resultado ao usuário (arquivo excluído ou erro no processamento);
- **WorkflowComposerBean:** Essa classe é a principal classe do sistema gerenciador de *workflows* desenvolvido. Ela controla a página relativa à composição dos *workflows*, fazendo toda verificação e validação dos dados de entradas, dos parâmetro de execução, do *design* do *workflow*, etc;
- **WorkflowStatusBean:** Requisita ao núcleo do BioNimbuZ o estado de um determinado *workflow*, enviando, para isso, seu ID. Formata os dados de retorno, enviando-os à página *HTML* para serem mostrados ao usuário;
- **WorkflowHistoryBean:** Envia requisições ao BioNimbuZ pedindo o passo-a-passo da execução de um dado *workflow* (*log*). Com este *log*, o usuário pode, por exemplo, rastrear possíveis erros, propor melhorias ao seu fluxo ou analisar o passo-a-passo de execução de seu *workflow*. Por fim esta classe formata a resposta enviada pelo núcleo e a disponibiliza à página *web*.

A listagem acima demonstra a maioria das classes Java desenvolvidas com objetivo de controlar a camada de visualização, mas não todas. Outras classes auxiliares também foram implementadas, mas, por possuírem menor importância, não foram aqui citadas.

Dessa forma, essas classes realizam a função de controle do modelo *MVC* que norteia o desenvolvimento da aplicação *web*. A próxima Seção, [5.4](#), trata da solução desenvolvida à fim de tratar o problema de comunicação entre a aplicação *web* e o núcleo do BioNimbuZ.

5.5 Camada de Comunicação

Um dos requisitos da solução proposta era que os componentes de software compostos pela aplicação *web* e pelo núcleo do BioNimbuZ fossem desenvolvidos de maneira separada. Com isso, surgiu o problema de comunicação entre esses dois elementos.

Nesse contexto, a solução desenvolvida tem como base a comunicação utilizando-se *webservices REST*(**R***Epresentational State Transfer*). *REST* é definido como *"estilo de arquitetura para sistemas de hipermídia distribuídos, descrevendo os princípios de engenharia de software que guiam o e as restrições de interação escolhidos para manter esses princípios, enquanto contrastando-as com as restrições de outros estilos arquiteturais."* [6]. Voltado para sistema baseados na Internet, tem sido amplamente utilizado na integração de sistemas, pois utiliza operações definidas no protocolo *HTTP* (tais como *PUT*, *GET* e *DELETE*). Assim, softwares que "entendam" o protocolo *HTTP*, podem utilizar, geralmente, soluções baseadas em *REST*.

Dessa forma, para possibilitar a troca de mensagens entre a aplicação *web* e o núcleo do BioNimbuZ, foi necessário o desenvolvimento de três entidades principais: **Requisições** (*requests*), **Respostas** (*responses*) e **Ações** (*actions*). Estas foram implementadas como `interface` Java, à fim de manter o código o mais genérico possível. Suas responsabilidades são as que seguem:

- **Ações:** Definem o comando à ser executado pelo núcleo, enviando-lhe uma requisição, à fim de se obter uma resposta com os dados requeridos;
- **Requisições:** De acordo com [6], toda requisição feita à um servidor deve conter todo o conjunto de informações necessárias para sua completude, pois a comunicação não deve manter estado (*stateless*). Assim, foram desenvolvidas requisições que contivessem todos os dados necessários para a execução daquela Ação requisitada pela aplicação *web*;
- **Respostas:** Definem a mensagem devolvida pelo núcleo do BioNimbuZ para uma dada ação. É verificada o código *HTTP* de resposta (*status code*), esperando-se, geralmente, o código *HTTP* 200, que indica sucesso na requisição. Outros código são

comuns, tais como o código HTTP 500, sinalizando erro interno do servidor e HTTP 400, que significa que uma requisição foi mal montada (*Bad Request*).

Para permitir a execução de uma determinada *Action*, e seus respectivos *Requests* e *Responses*, foram desenvolvidas duas classes Java para controlar seu fluxo, executando uma ação, enviando sua respectiva requisição e aguardando uma resposta. As responsabilidades dessas classes, chamadas `RestService` e `RestCommunicator`, estão listadas abaixo:

- **RestService:** Contém a lista de métodos disponíveis na Camada de Comunicação. Outros componentes de software só podem requisitar um comando *REST* a partir da instanciação desta classe e caso o respectivo método ter sido previamente implementado.
- **RestCommunicator:** É responsável por prover um passo a passo a ser seguido para realização da comunicação via *REST* da classe `RestService`. Utilizando o padrão de projeto *Strategy* [8] em conjunto com o conceito de polimorfismo, executa os seguinte métodos definido na interface *Action*:
 1. **ping()**: Verifica o estado do núcleo do BioNimbuz antes de toda requisição, efetuando o comando *ping* [15], podendo retornar verdadeiro (`true`) ou falso (`false`). Em caso negativo, a requisição falha e retorna uma mensagem de erro ao usuário com a mensagem "*Erro Interno do Servidor*", sinalizando a falha na comunicação com o núcleo;
 2. **setup()**: Realiza a configuração prévia do objeto à ser enviado ao núcleo do BioNimbuz, suprindo necessidades da comunicação via *REST*, tais como: instanciação do cliente *REST*, *set* das informações necessárias para a requisição, etc;
 3. **prepareTarget()**: Configura o endereço alvo (*target*) daquela requisição. Aponta para uma *URL* contendo o endereço IP do núcleo juntamente com o caminho do recurso *REST* [6] que irá recepcionar a requisição setada;
 4. **execute()**: Executa a Ação desejada, realizando a operação *REST* definida pela requisição (*GET*, *POST*, *DELETE*, etc), retornando uma resposta para as classes de controle.

A partir destas classes, foram desenvolvidas as implementações das interfaces `Action`, `ResponseInfo` e `RequestInfo`. A Tabela 5.2 apresenta a responsabilidade de cada relação de classes `Action` x `ResponseInfo` x `RequestInfo`.

Action	ResponseInfo	RequestInfo
DeleteFile	DeleteFileResponse	DeleteFileRequest
GetWorkflowHistory	GetWorkflowHistoryResponse	GetWorkflowHistoryRequest
GetWorkflowStatus	GetWorkflowStatusResponse	GetWorkflowStatusRequest
Login	LoginResponse	LoginRequest
Logout	LogoutResponse	LogoutRequest
SignUp	SignUpResponse	SignUpRequest
StartWorkflow	StartWorkflowResponse	StartWorkflowRequest
Upload	UploadResponse	UploadRequest

Tabela 5.2: Relação entre classes Java e páginas *HTML*.

5.6 Alterações realizadas no Núcleo do BioNimbuz

Compreende os serviços providos atualmente pelo BioNimbuz (como serviço de armazenamento, escalonamento, descobrimento) acrescido do código necessário para executar os comandos e requisições vindos da parte Cliente. Também deverá implementar métodos de acesso à bancos de dados relacionais, pois deverá persistir os dados trocados entre a aplicação Cliente e o Servidor do BioNimbuz (entidades como Usuário, Workflow e Arquivo).

Capítulo 6

Resultados Obtidos

telas desenvolvidas para realizar a interface com o usuário e que, juntas, representam a camada de visualização da aplicação *web*.

Tela de Login

A Figura 6.1 mostra a tela inicial da aplicação *web* desenvolvida. Nessa tela, o usuário deve entrar com seu usuário e senha cadastrados previamente para ter acesso aos recursos disponíveis no BioNimbuZ.



Figura 6.1: Tela inicial da aplicação *web*.

Tela de Cadastro

Por essa tela, novos usuários se cadastram na plataforma BioNimbuZ, incluindo seus dados, tais como telefone, nome, CPF, etc. A Figura 6.2 mostra a tela que é apresentada ao usuário.

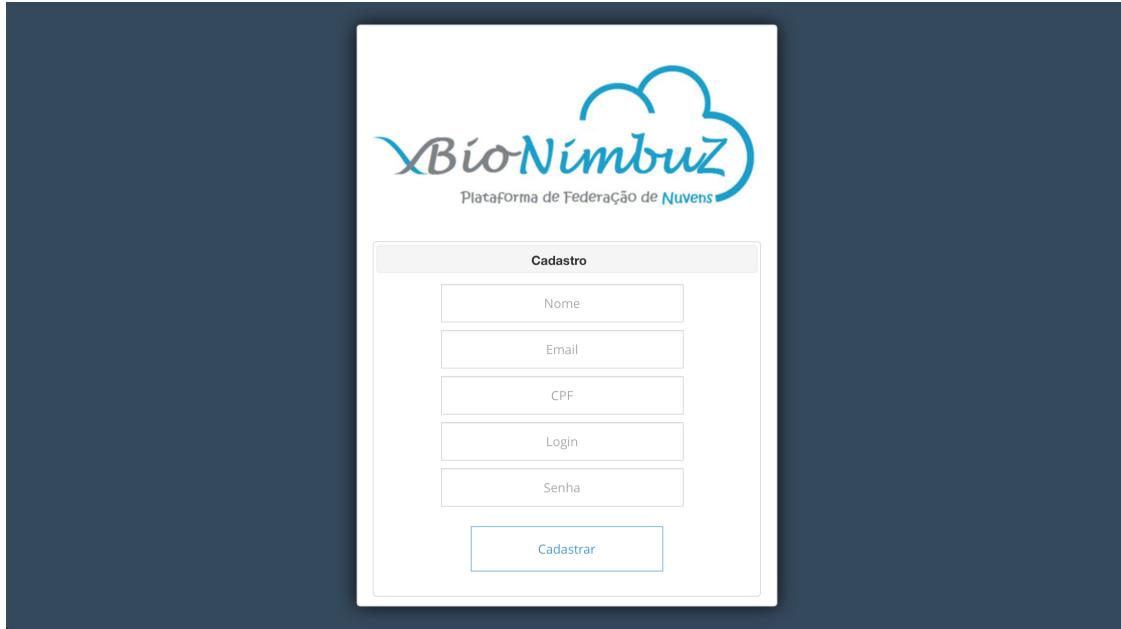


Figura 6.2: Tela de cadastro de novos usuários.

Tela Inicial

Ao realizarem o *login*, os usuários que possuírem acesso ao ambiente *web* visualizarão a tela mostrada na Figura 6.3. Essa tela dá acesso às opções disponíveis no BioNimbuZ, tais como criar um novo *workflow*, visualizar seu *status*, enviar arquivos para plataforma, etc.

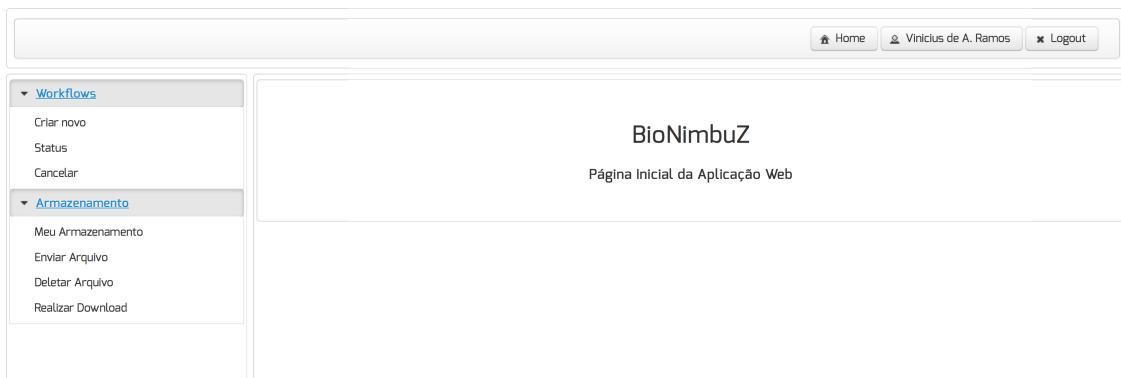


Figura 6.3: Tela inicial da aplicação *web*.

Tela inicial da criação dos *Workflows*

Nesta tela, mostrada na Figura 6.4, o usuário inicia o projeto de seu *workflow*, podendo escolher entre criar um novo *workflow* ou importar um anteriormente criado. A aplicação *web* possibilita, ao fim do projeto de um *workflow*, exportá-lo. Assim, com esse *workflow* exportado, é possível compartilhá-lo, sendo este a última fase do ciclo de vida de um *workflow* (detalhado no Capítulo 3).

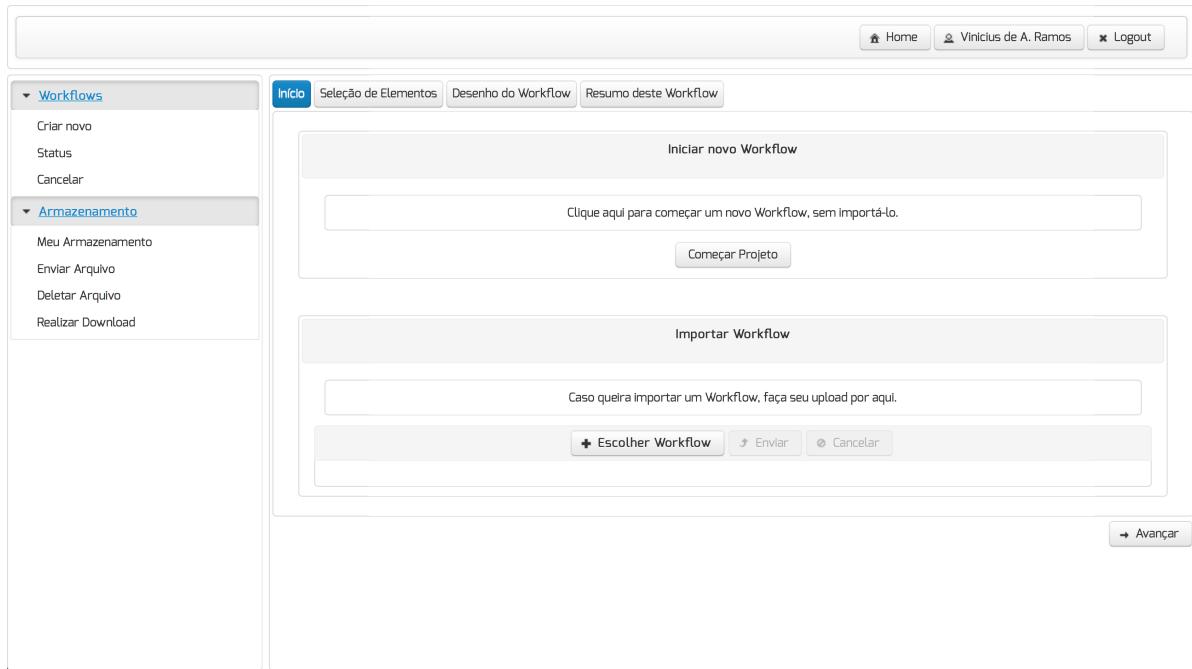


Figura 6.4: Tela inicial de criação dos *workflows*.

Tela de seleção de elementos

Ao clicar em "Começar Projeto", o usuário então é direcionado para a página de seleção de elementos, mostrada na Figura 6.5. Nela, o mesmo poderá planejar a execução de seu *workflow*, escolhendo quais elementos irão compor este fluxo. A lista de elementos disponíveis é mantida no núcleo do BioNimbuZ e é enviada à aplicação quando esta é iniciada. À medida que o usuário escolhe os elementos, eles são adicionados na listagem "Elementos Escolhidos", podendo o mesmo excluí-los, caso não os queira mais.

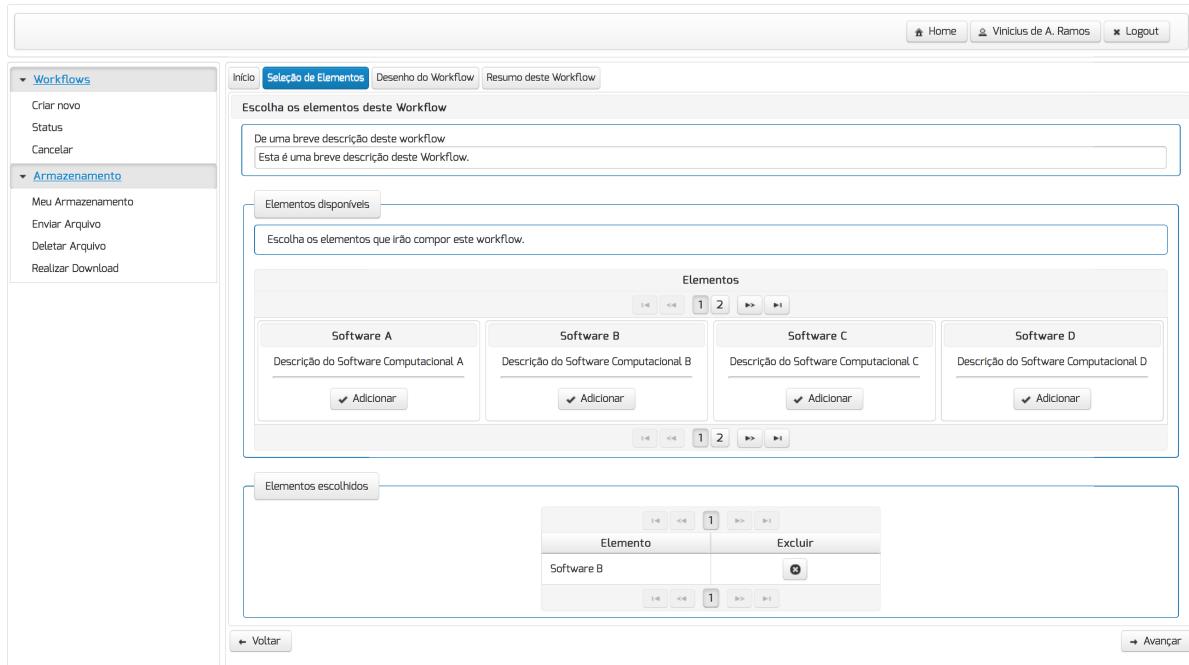


Figura 6.5: Tela utilizada para selecionar elementos que irão compôr o *workflow*.

Tela de *design* do *workflow*

Com os elementos definidos na tela anterior (Figura 6.5), o usuário é redirecionado para a tela seguinte, passando para a fase de *design* do *workflow*. Nela, são apresentados os elementos escolhidos pelo usuário dispostos de maneira gráfica, conforme mostra a 6.6.

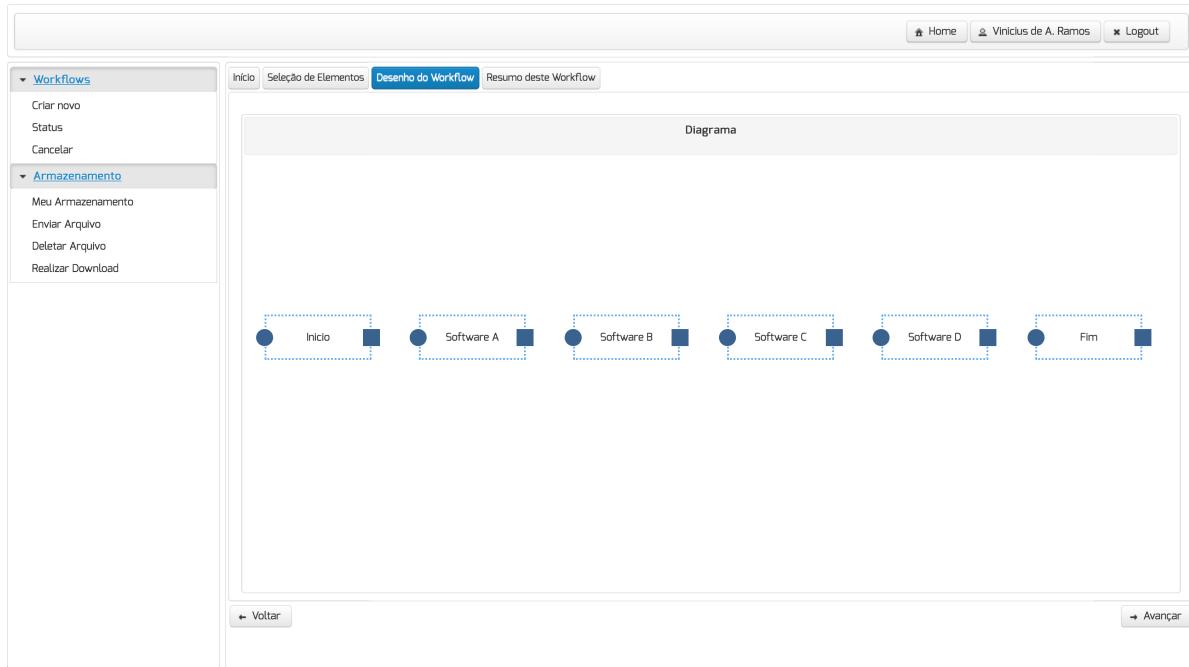


Figura 6.6: Nesta tela o usuário compõe o fluxo de seu *workflow*.

Com isso, o usuário pode definir as dependências de cada elemento, arrastando os pontos de ligação, definir suas entradas (arquivos enviados ou uma URL que contenha o caminho para realizar o download do arquivo para a plataforma BioNimbuz), definir os parâmetros de execução (como argumentos daquele componente) ou apenas sinalizar que a entrada de um determinado elemento será a saída de outro. Para isso, ao se conectar dois elementos, é mostrada uma caixa de diálogo pedindo estes dados (Figura 5.9).

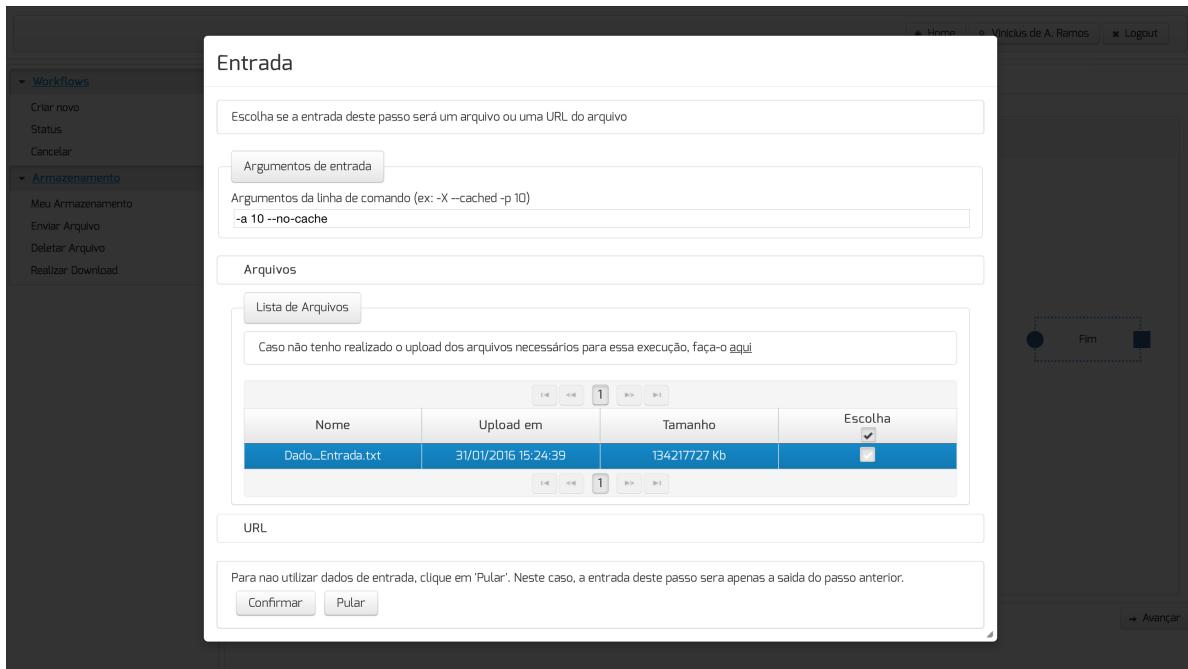


Figura 6.7: Nesta tela o usuário define parâmetros de execução de um elemento do *workflow*.

Tela de Confirmação

Com todos os elementos escolhidos e com suas dependências (entrada, saída e parâmetros de execução) satisfeitas, é apresentada uma tela de confirmação (Figura 6.8), em que o usuário pode confirmar os dados daquele *workflow* e submetê-lo à execução, ou pode apenas realizar o *download* do arquivo **.flow** (representação do objeto Java contendo os dados de um *workflow*). De posse deste arquivo, o usuário pode importá-lo em outro momento para, por exemplo, finalizá-lo, ou também pode compartilhar este arquivo com outros usuários do sistema, para que outros tenham acesso ao seu trabalho.

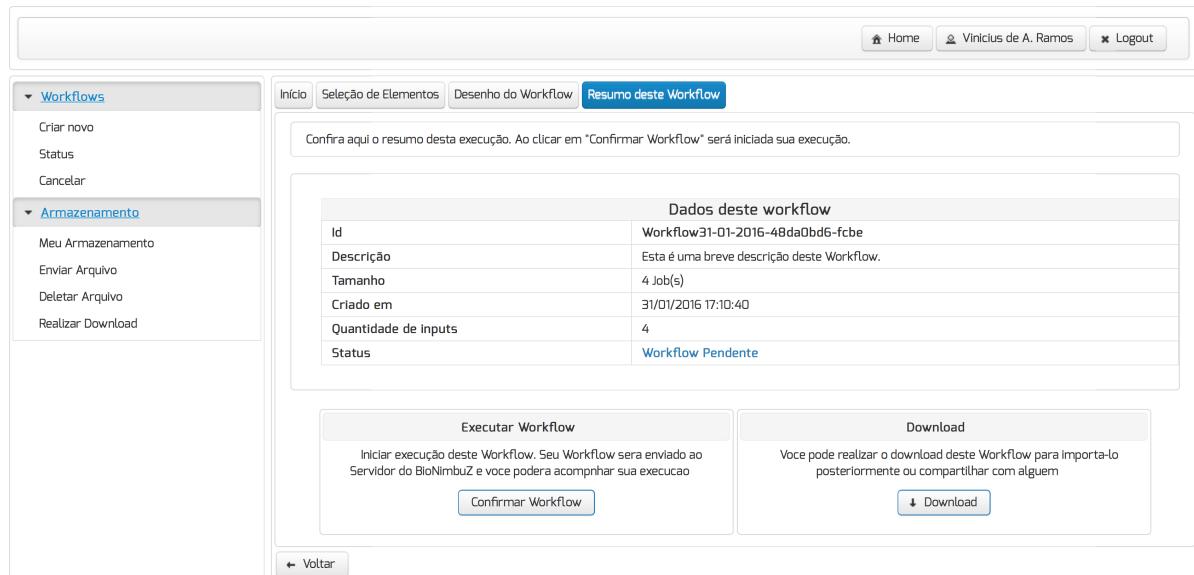


Figura 6.8: Tela final da composição do *workflow*.

Tela de *status* do *workflow*

A partir do momento que o usuário escolheu submeter o *workflow* à plataforma BioNimbuz, ele pode visualizar o estado de sua execução na tela de *status*, apresentada na Figura ??.

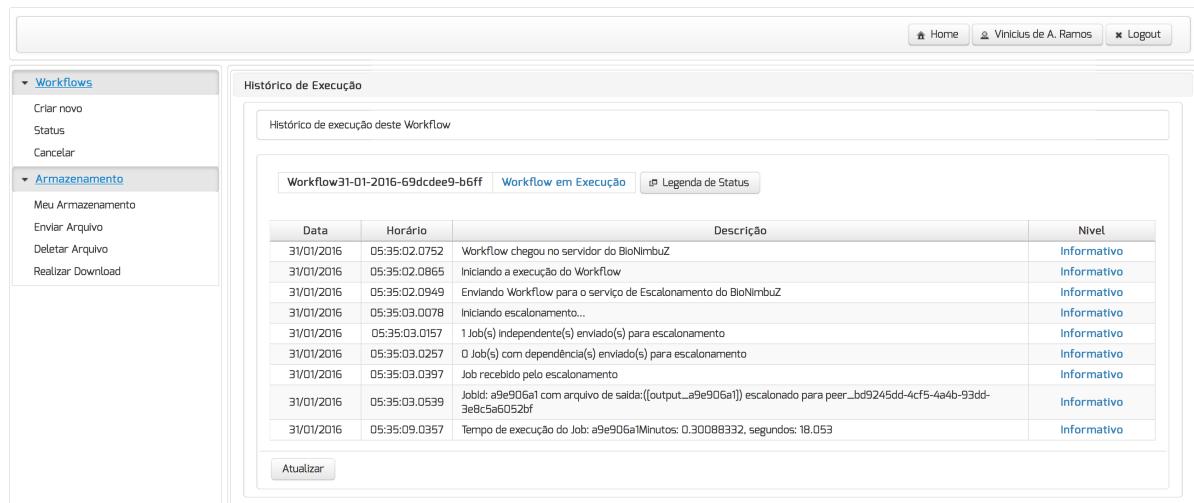


Figura 6.9: Tela final da composição do *workflow*.

Os possíveis estados de um *workflow* estão retratados na Tabela 6.1.

Status	Nível	Descrição
Workflow Pausado	Informativo	Foi requisitada a pausa na execução do Workflow.
Workflow Pendente	Informativo	Seu Workflow está aguardando execução pelo BioNimbuZ.
Workflow em Execução	Informativo	O Workflow está em plena execução.
Workflow Finalizado com Sucesso	Informativo	Seu Workflow foi finalizado sem alertas e sem erros.
Workflow Finalizado com Alertas	Alerta	Alertas foram lançados durante a execução deste Workflow.
Workflow Finalizado com Erros	Erro	Erros não permitiram que este Workflow terminasse com sucesso.
Workflow Parado com Erros	Erro	Estado parcial com erros.

Tabela 6.1: Estados possíveis de um *Workflow*

Telas de *upload* de arquivos e armazenamento

À fim de se compôr um *workflow*, o usuário tem duas formas de lhe prover dados de entrada: utilizando uma *URL* para passar o caminho de rede de onde se encontra o arquivo, para que dessa forma o núcleo do BioNimbuZ possa realizar o *download* do mesmo, ou enviar arquivos para a plataforma do BioNimbuZ, a partir da tela mostrada na Figura 6.10.

The screenshot shows a web-based application interface. At the top right are buttons for 'Home', 'Vinicius de A. Ramos', and 'Logout'. On the left, there's a sidebar with two expandable sections: 'Workflows' (containing 'Criar novo', 'Status', and 'Cancelar') and 'Armazenamento' (containing 'Meu Armazenamento', 'Enviar Arquivo' (which is expanded), 'Deletar Arquivo', and 'Realizar Download'). The main content area has a title 'Upload de Arquivos para seu Armazenamento' and a sub-instruction 'Página utilizada para realizar upload de arquivos para o servidor.' Below this is a button bar with '+ Escolher Arquivo', 'Enviar', and 'Cancelar'. At the bottom is a table titled 'Sua lista de arquivos' with columns 'Nome', 'Tamanho', and 'Upload em'. It contains one entry: 'Nome' is 'Dado...Entrada.txt', 'Tamanho' is '110217.727 kb', and 'Upload em' is '31/01/2016 15:24:39'.

Figura 6.10: Tela utilizada para realizar o *upload* de arquivos.

Nesta implementação, foi definido um armazenamento máximo de 256 *Megabytes* por usuário. Foi desenvolvida uma interface para que usuários possam ter controle de seu armazenamento, apresentada na Figura 6.11.

Workflows

- Criar novo
- Status
- Cancelar

Armazenamento

Meu Armazenamento

- Enviar Arquivo
- Deletar Arquivo
- Realizar Download

Seu Armazenamento

Seu armazenamento está em: 41%

Arquivos

Visualize aqui sua lista de arquivos enviados ao seu armazenamento

Sua lista de arquivos		
Nome	Tamanho	Upload em
Dado...Entrada.txt	110217.727 kb	31/01/2016 15:24:39

Realizar novo Upload

Figura 6.11: Nesta tela usuários controlam seu armazenamento e verificam sua porcentagem.

Desta forma, as telas apresentadas compõem a camada de visualização do Modelo *MVC*.

Capítulo 7

Conclusões e Trabalhos Futuros

Descrever o que pode ser melhorado na solução proposta e quais os próximos passos.

Referência Bibliográfica

Temporário

- 1 Cloud Computing and Grid Computing 360-Degree Compared
- 2 A Taxonomy and Survey of Cloud Computing Systems
- 3 BioNimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de Bioinformática
- 4 Scientific Process Automation and Workflow Management
- 5 A Break In The Clouds: Towards a Cloud Definition
- 6 Distributed Systems and Recent Innovations: Challenges and Benefits
- 9 Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms
- 10 Time, Clocks and the Ordering of Events in a Distributed Systems.
- 11 Cloud Computing vs. Grid Computing
- 12 The Anatomy of The Grid
- 13 Cluster Computing - High Performance, High-Availability and High Throughput Processing on a Network of Computers
- 14 Cluster Computing: Architectures, Operating Systems, Parallel Processing and Programming Languages
- 15 Cloud Computing - State of the Art and Research Challenges
- 16 The NIST Definition of Cloud Computing
- 17 Xen Project, <http://www.xenproject.org/> (acessado em 11/01/2016)
- 18 KVM, http://www.linux-kvm.org/page/Main_Page (acessado em 11/01/2016)
- 19 VMWare, <http://www.vmware.com/br> (acessado em 11/01/2016)
- 20 How to Enhance Cloud Architectures to Enable Cross-Federation
- 21 Provenance Collection Support in the Kepler Scientific Workflow System
- 22 <http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html> (acessado em 12/01/2016)
- 23 Towards a Model of Provenance and User Views in Scientific Workflows

- 24 Scientific Workflow Management and the KEPLER System
- 25 WOODSS — a spatial decision support system based on workflows
- 26 E. White, L. McMillan, P. Romanski, M. O’Gara, and J. Bloomberg. Inter-cloud peering points. <http://cloudcomputing.sys-con.com/node/1658700>, 2010. vii, 13
- 27 Dropbox <http://www.dropbox.com/>
- 28 Amazon EC2 <https://aws.amazon.com/pt/ec2/>
- 29 Heroku <https://www.heroku.com/>
- 30 Scientific Process Automation and Workflow Management
- 31 Scientific Workflows - Business as Usual?
- 32 JSF <https://javaserverfaces.java.net/>
- 33 Primefaces <http://www.primefaces.org/>
- 34 M. Ghanem, V. Curcin, P. Wendel, and Y. Guo, “Building and using analytical workflows in discovery net,” in Data mining on the Grid, W. Dubitzky, Ed. John Wiley and Sons, 2008.
- 35 D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, “Taverna: A tool for building and running workflows of services,” Nucleic Acids Research, vol. 34, pp. W729–W732, 2006, web Server Issue.
- 36 I. Taylor, M. Shields, I. Wang, and A. Harrison, “Visual Grid Workflow in Triana”, Journal of Grid Computing, vol. 3, no. 3-4, pp. 153–169, September 2005. [Online]. Available:
- [\http://www.springerlink.com/openurl.asp?genre=article&issn=1570-7873&volume=3&issue=3&spage=153](http://www.springerlink.com/openurl.asp?genre=article&issn=1570-7873&volume=3&issue=3&spage=153)
- 37 B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, “Scientific workflow management and the kepler system: Research articles,” Concurr. Comput. : Pract. Exper., vol. 18, no. 10, pp. 1039–1065, 2006.
- 38 Pegasus Mapping Scientific Workflows onto the Grid
- 39 The Pegasus Portal Web Based Grid Computing
- 40 Use a Cabeça! Padrões de Projeto
- 41 Spring MVC <https://spring.io> (acessado em 18/01)

- 42 Struts <https://struts.apache.org> (acessado em 18/01)
- 43 Play <https://www.playframework.com>
- 44 IDBS <https://www.idbs.com/en/> (acessado em 19/01)
- 45 Google www.google.com.br
- 46 Amazon www.amazon.com
- 47 Microsoft www.microsoft.com
- 48 Google Compute Engine <https://cloud.google.com/compute/>
- 49 GoGrid <https://www.datapipe.com/gogrid/>
- 50 CloudForge <http://www.cloudforge.com>
- 51 AppScale <http://www.appscale.com>
- 52 Abiquo <http://www.abiquo.com>
- 53 SalesForce <http://www.salesforce.com/br/>
- 54 Zendesk <https://www.zendesk.com.br>
- 55 Breno Rodrigues Moura and Deric Lima Bacelar. Política para armazenamento de arquivos no zoonimbus, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 20, 21, 25
- 56 Gabriel Silva Souza de Oliveira. Acosched: um escalonador para o ambiente de nuvem federada zoonimbus, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 20, 21, 24
- 57 Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, 2003. 20
- 58 Raj Srinivasan. RPC: Remote Procedure Call protocol Specification version 2 , Request For Comments (RFC 1831). <http://tools.ietf.org/pdf/rfc1831.pdf>, 1995. Acessado online em 27 de janeiro de 2016.
- 59 The Apache Software Fundation. Apache zookeeper. <http://zookeeper.apache.org/>, 2015. Acessado online em 15 de janeiro de 2016. vii, 21, 22
- 60 The Apache Software Fundation. Apache fundation. <http://apache.org/>, 2015. Acessado online em 15 de janeiro de 2016. 21

- 61 The Apache Software Fundation. Apache Avro. <http://avro.apache.org/>, 2012. Acessado online em 15 de janeiro de 2016
- 62 Douglas Crockford. The application/json Media Type for Javascript Object Notation (JSON) , Request For Coments (RFC 4627). <http://tools.ietf.org/pdf/rfc4627.pdf>, 1995. Acessado online em 05 de junho de 2015. 21
- 63 Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext Transfer Protocol–http/1.1 , Request For Coments (RFC 2616). <http://tools.ietf.org/pdf/rfc2616.pdf>, 1999. Acessado online em 05 de junho de 2015. 21
- 64 The Apache Software Fundation. Apache avro documentation. <http://avro.apache.org/docs/1.7.7> 2015. Acessado online em 05 de junho de 2015. 21
- 65 Thrift <https://thrift.apache.org>
- 66 Protocol Buffer <https://developers.google.com/protocol-buffers/>

Referências

- [1] Especificação html. <https://tools.ietf.org/html/rfc2616>. Acessado online em 31 de janeiro de 2016. [42](#)
- [2] Primefaces. <http://www.primefaces.org>. Acessado online em 31 de janeiro de 2016. [42](#)
- [3] Christopher Brooks, Edward A. Lee, Xiaojun Liu, Stephen Neuendorffer, Yang Zhao, e Haiyang Zheng. Heterogeneous concurrent modeling and design in java (volume 2: Ptolemy ii software architecture). Technical Report UCB/EECS-2008-29, EECS Department, University of California, Berkeley, Apr 2008. [24](#)
- [4] V. Curcin e M. Ghanem. Scientific workflow systems - can one size fit all? In *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International*, pages 1–9, 2008. [x](#), [22](#), [23](#), [24](#)
- [5] Research Studio Digital Memory Engineering. Research studio digital memory engineering. <http://www.rs-dme.at/>. Acessado online em 29 de janeiro de 2016. [x](#), [24](#)
- [6] Roy T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, Irvine - Irvine, CA 92697, USA, 2000. [45](#), [46](#)
- [7] Ian Foster, Yong Zhao, Ioan Raicu, e Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10. IEEE, November 2008. [x](#), [5](#), [8](#), [10](#), [12](#), [13](#)
- [8] Elisabeth Freeman, Eric Freeman, Bert Bates, e Kathy Sierra. *Head First Design Patterns*. O'Reilly & Associates, Inc., 2004. [45](#)
- [9] E. Gallopoulos, E. Houstis, e J.R. Rice. Computer as thinker/doer: problem-solving environments for computational science. *Computational Science Engineering, IEEE*, 1(2):11–23, Summer 1994. [24](#)
- [10] B. Hoehrmann. Scripting Media Types. RFC 4329 (Informational), April 2006. [42](#)
- [11] Kepler. Exemplo de interface gráfica do sistema kepler. <http://www.cell.com/cms/attachment/2002983292/2011324825/gr2.jpg>. Acessado online em 30 de janeiro de 2016. [x](#), [25](#)

- [12] H. Lie, B. Bos, e C. Lilley. The text/css Media Type. RFC 2318 (Informational), March 1998. [42](#)
- [13] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, e Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency Computat.: Pract. Exper.*, 18(10):1039–1065, August 2006. [x](#), [24](#), [25](#)
- [14] S. Majithia, M. Shields, I. Taylor, e I. Wang. Triana: a graphical web service composition and execution toolkit. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 514–521, July 2004. [23](#)
- [15] J. Quittek e K. White. Definitions of Managed Objects for Remote Ping, Traceroute, and Lookup Operations. RFC 4560 (Proposed Standard), June 2006. Acessado online em 01 de fevereiro de 2016. [46](#)
- [16] Taverna. Taverna online. <http://onlinehpc.com/>. Acessado online em 30 de janeiro de 2016. [x](#), [23](#)