
# NUMERUSCLOUD: UM SISTEMA PARA AUTOMAÇÃO DE IMPLANTAÇÃO DE ROTINAS NUMÉRICAS USANDO COMPUTAÇÃO EM NUVEM

João Carlos Purificação dos Santos

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Adilson Elias Xavier

Rio de Janeiro

Setembro de 2013

NUMERUSCLOUD: UM SISTEMA PARA AUTOMAÇÃO DE IMPLANTAÇÃO DE ROTINAS NUMÉRICAS USANDO COMPUTAÇÃO EM NUVEM

João Carlos Purificação dos Santos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Adilson Elias Xavier, D.Sc.

Prof. Sergio Barbosa Villas-Boas, Ph.D.

Prof. Frederico Caetano Jandre de Assis Tavares, D.Sc.

Prof. José Ferreira de Rezende, Ph.D.

RIO DE JANEIRO – RJ, BRASIL
SETEMBRO DE 2013

Santos, João Carlos Purificação dos

NumerusCloud: Um Sistema para Automação de Implantação de Rotinas Numéricas Usando Computação em Nuvem/João Carlos Purificação dos Santos. – Rio de Janeiro: UFRJ/COPPE, 2013.

XIII, 79 p.: il.; 29, 7cm.

Orientador: Adilson Elias Xavier

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2013.

Referências Bibliográficas: p. 75 – 79.

1. Computação científica 2. Computação numérica 3. Computação em nuvem 4. Web Services I. Xavier,Adilson Elias. II.Universidade Federal do Rio de Janeiro, COPPE , Programa de Engenharia de Sistemas e Computação. III. Título

*"Quem passou pela vida em branca nuvem, E em plácido repouso adormeceu; Quem não sentiu o frio da desgraça, Quem passou pela vida e não sofreu; Foi espectro de homem, não foi homem, Só passou pela vida, não viveu."(Francisco Otaviano)*

# Agradecimentos

Agradeço aos meus pais, a minha filha e a minha esposa pelo apoio integral que deram. Agradeço especialmente ao meu pai, que sempre foi o meu maior mentor.

Ao meu orientador Adilson Elias Xavier

Ao co-orientador Sergio Barbosa Villas-Boas, que muito ajudou-me e que orientou-me de fato. Seu nome não consta como orientador oficial, mas apenas como co-orientador, meramente por questões burocráticas.

Ao povo brasileiro, que pagou o meu curso ao longo desses anos.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre (M.Sc.)

# NUMERUSCLOUD: UM SISTEMA PARA AUTOMAÇÃO DE IMPLANTAÇÃO DE ROTINAS NUMÉRICAS USANDO COMPUTAÇÃO EM NUVEM

João Carlos Purificação dos Santos

Setembro/2013

Orientador: Adilson Elias Xavier

Programa: Engenharia de Sistemas e Computação

O uso de execução remota de funções para aplicações científicas cria interessantes possibilidades para arquiteturas de software. Por exemplo: pode-se conceber um software em que uma camada contém a apresentação gráfica e interface e outra camada contém execução numérica. A arquitetura com isolamento de camadas pode produzir diversas consequências muito desejáveis, incluindo melhoria de manutenibilidade do código, proteção da propriedade intelectual, maior flexibilidade para substituição tecnológica do hardware de execução numérica, independência de tipo de cliente para as funções numéricas, e outras.

Nesse trabalho, é proposta uma arquitetura de software - chamada de NumerusCloud - que usa tecnologia de cloud computing para implementar com 3 camadas um sistema que permite automatizar a implantação de rotinas numéricas. A arquitetura proposta oferece importantes vantagens sobre as técnicas conhecidas de execução remota, incluindo isolamento de camada entre o agente de implantação e o autor da rotina numérica. A arquitetura proposta foi implementada, e como exemplo de utilização fez-se a implantação de uma função de clusterização (para a qual aplicam-se restrições de propriedade intelectual), com 2 tipos de clientes na camada de apresentação - um GUI em PC e outro em mobile (Android).

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

NUMERUSCLOUD - AN AUTOMATED METHOD TO DEPLOY NUMERIC PROCEDURES USING CLOUD COMPUTING

João Carlos Purificação dos Santos

September/2013

Advisor: Adilson Elias Xavier

Department: Systems Engineering and Computer Science

The remote execution of functions for scientific applications creates interesting possibilities of software architectures. For example: it is possible to conceive software in a tier containing only graphical presentation and interface, and other tier containing numeric execution. An architecture with tier isolation may produce several very desirable consequences, including the improvement of the maintainability of the code, enhancement of the intellectual property protection, improved flexibility to handle the technological substitution of hardware, the independence of type of client for the numeric functions, and others.

In this work, it is proposed a software architecture - named NumerusCloud - that uses cloud computing technologies to implement with 3 tiers a system that allows the automation of deployment of numeric functions. The proposed architecture offers important advantages over the known techniques of remote execution, including the isolation of the tiers of the deployment agent and the tier of the author of numerical task. The proposed architecture was implemented, and as usage example it was deployed a numeric function to calculate the clustering problem (a function where intellectual restrictions apply), using 2 types of clients in presentation tier - one as a GUI for PC and other for mobile (Android).

# Nomenclatura

- *API - Application Program Interface* - interface de programação de uma aplicação.
- *BIOS* - Basic Input/Output System - Sistema Básico de Entrada e Saída. O BIOS é um programa de computador responsável pelo suporte básico de acesso ao hardware e que inicia o sistema operacional quando o computador é ligado.
- *browser* - navegadores.
- *cloud computing* - computação em nuvem.
- *DAO - Data Access Object* - literalmente Objeto de Acesso de Dados, um padrão de projeto de software.
- *database* - banco de dados.
- *data center* - Centro de Processamento de Dados ou CPD.
- *deployment* - implantação.
- *design pattern* - padrões de projeto (de software).
- *download* - baixar (copiar) arquivos de algum computador para o que se está usando.
- *EJB - Enterprise Java Bean* (literalmente Grão de Café Empresarial) - componente de software na linguagem java para uso em desenvolvimento de sistemas.
- *entry point* - ponto de entrada.
- EUD - *End User Device* - Dispositivo do Usuário Final.
- *facade* - literalmente "fachada", nome de um padrão de projeto de software.

- *filesystem* - sistema de arquivos.
- *firewall* - é um software ou hardware que verifica informações oriundas da Internet ou de uma rede e bloqueia-as ou permite que elas passem por um computador, dependendo das configurações do firewall.
- *framework* - estrutura.
- *front end* - a parte do sistema de software que interage diretamente com o usuário.
- GPU - *Graphics Processing Unit* - Unidade de Processamento Gráfico.
- GUI - *Graphics User Interface* (interface gráfica de usuário).
- *Grid Computing* - literalmente computação em grade. Uma arquitetura de software para computação paralela.
- *IDE (Integrated Development Environment)* - ambiente integrado de desenvolvimento (de software).
- *input* - entrada.
- *JVM - Java Virtual Machine* - máquina virtual java.
- *login* - procedimento de entrada no sistema, que requer autenticação.
- *mobile* - dispositivo móvel, isso é, telefone celular tipo *smart phone* ou tablet.
- *MVC - Model View Controller* - literalmente: modelo visualização controle, um padrão de projeto de software.
- NCC *Numerus Cloud Controller* - Controlador Numerus Cloud
- NTE *Numerical Task Executor* - Executor de Tarefa Numérica
- *open source* - código aberto.
- *output* - saída.
- PC (*Personal Computer*) - computador pessoal.

- *pool* - conjunto.
- *request* - solicitação ou requisição.
- *response* - resposta.
- RFC (*Request for Comments*) - é um documento que descreve os padrões de cada protocolo da Internet previamente a serem considerados um padrão.
- RMI (*Remote Method Invocation*) - invocação de método remoto.
- RPC (*Remote procedure call*) - chamada remota de função ou procedimento.
- *runtime* - período de execução (de um programa).
- *servlet* - sem tradução. Componente de software em java.
- *session id* - identificador da sessão do cliente no Numerus Cloud.
- *stream* - fluxo ou encadeamento.
- *task process id* - identificador da instância da tarefa a ser executada.
- *thin client* - literalmente "cliente magro"; são terminais de computadoresterminais, muitas vezes chamados de "terminais burros", pois eles eram usados para comunicação, porém sem capacidade de processamento interno.
- *thread* - encadeamento, no sentido de uma linha paralela de execução de um programa.
- *timeout* - tempo limite.
- *time-sharing* - tempo compartilhado.
- *upgrade* - atualização (de software).
- *upload* - enviar arquivos para algum computador.
- VPN (*virtual private network*) - rede privada virtual.
- *workstation* - estação de trabalho.

# Sumário

# Lista de Figuras

# Capítulo 1

# Introdução

Em computação, existem diversas tecnologias que possibilitam a execução remota de funções, isto é, um computador comandar a execução de uma função em um computador remoto. Dentre os inúmeros objetivos que existem para uso de computadores, o mais tradicional é o objetivo de computação numérica, também chamada de computação científica. O uso de execução remota de funções para aplicações científicas cria interessantes possibilidades para arquiteturas de software. Por exemplo: pode-se conceber um software em que uma camada contém a apresentação gráfica e interface e outra camada contém a execução numérica. A arquitetura com isolamento de camadas pode produzir consequências muito desejáveis, incluindo melhoria de manutenibilidade do código, proteção da propriedade intelectual, maior flexibilidade para substituição tecnológica do hardware de execução numérica, independência de tipo de cliente para as funções numéricas, e outras.

Uma tendência recente no cenário de tecnologia de informação é o que vem sendo chamado de *cloud computing* (computação em nuvem). Dentre as características da *cloud computing* (computação em nuvem) inclui-se o fato de que se pode alugar recursos computacionais a preços acessíveis de forma escalável, até um limite tão grande que é virtualmente infinito. As tecnologias de software para isso podem ser usadas tanto na Internet pública, com ip verdadeiro, quanto em data center (Centro de Processamento de Dados - CPD) privado, se requerido. Destaca-se o fato de que praticamente todo software e protocolos necessários para a implementação de *cloud computing* (computação em nuvem) são originalmente open source (código aberto) ou possuem versões open source (código aberto) com um bom funcionamento.

Nesse trabalho, é proposta uma arquitetura de software - chamada de NumerusCloud - que usa tecnologia de *cloud computing* (computação em nuvem) para implementar com 3 camadas um sistema que permite automatizar o *deployment* (implantação) de rotinas numéricas. A arquitetura proposta oferece importantes vantagens sobre as técnicas conhecidas de execução remota, incluindo isolamento de camada entre o agente de *deployment* (implantação) e o autor da rotina numérica o que provê proteção à propriedade intelectual. Algumas características de *cloud computing* (computação em nuvem) são implicitamente incluídas na arquitetura proposta, tornando-a mais flexível e útil.

O trabalho também propõe uma camada de software para implementar controle de sessão para sistemas na nuvem, chamado de DOUA (Database Oriented Usecase Authorization). Essa é uma proposição genérica que serve para qualquer software na nuvem que requeira controle de sessão, isso é, autorização para execução de caso de uso. Veja seção 3.8, na página 27.

A arquitetura proposta foi implementada, e como exemplo de utilização fez-se o *deployment* (implantação) de uma função de clusterização (para a qual aplicam-se restrições de propriedade intelectual), com 2 tipos de clientes na camada de apresentação - um GUI (*Graphical User Interface* - Interface Gráfica com o Usuário) em PC e outro em *mobile* (Android).

Rotinas e procedimentos da computação científica são feitos para a solução de problemas ligados à própria ciência e à engenharia, envolvendo a construção de modelos matemáticos e técnicas de solução numérica, isto é, a execução de requisitos funcionais numéricos. Uma vez desenvolvidas as tarefas numéricas, surge uma etapa atualmente que pode ser tão ou mais complexa: tornar o programa ou serviço numérico disponível ao mundo exterior, representado por diversos programas clientes que rodam em diversos dispositivos. Esses programas são: navegadores (*browsers*), aplicativos do tipo *desktop*, softwares embarcados em dispositivos móveis (*tablets* e *smartphones*), ou mesmo programa de um barramento. Ao mesmo tempo, a computação científica requer a disponibilidade de soluções computacionais para alta performance, avançados recursos de infraestrutura de hardware como memória, *clusters* de máquinas e supercomputadores, que são difíceis de configurar, manter e operar. Paralelamente, surgem outras necessidades como a proteção da propriedade intelec-

tual do software numérico gerado, por vezes fruto de anos de desenvolvimento, testes e dedicação, permitir o uso de avançados recursos de harware como memória, pool (conjunto) de máquinas ou multi-processadores, e o estabelecimento de políticas de uso com relação às tarefas e aos tipos de clientes. A *cloud computing* (computação em nuvem) proporciona aos cientistas um novo modelo de utilização da infraestrutura computacional. Recursos como armazenamento bem como aplicações podem ser dinamicamente fornecidos.

O sistema Numerus Cloud - NC- proposto tem como finalidade a automatização do *deployment* (implantação) de procedimentos numéricos utilizando o *cloud computing* (computação em nuvem).

Duas das principais características do *cloud computing* (computação em nuvem) são: poupar ou economizar o processamento e o uso de recursos locais e a transparência para o cliente em relação à localizacão real do(s) computador(es) onde realmente está sendo processado o método numérico. As características da máquina que executa o serviço, bem como os aspectos de hardware e de rede não são levados em consideração, mas sim que "alguém" executará o serviço requisitado pelo cliente. Um autor de métodos numéricos prepara seu procedimento numérico (uma tarefa numérica) que é executado num computador de seu controle. Registra em seguida esta mesma tarefa no Numerus Cloud e define uma política de uso para este procedimento. No computador controlado pelo autor estará residente um programa agente responsável pelo recebimento de requisições da NC e pela execução das tarefas registradas. O uso do procedimento numérico é disponibilizado via Web Services pela NC. O cliente final roda um programa na sua própria máquina ou em qualquer outro dispositivo que chama o procedimento numérico através da NC. O autor pode criar uma política de uso para uma determinada tarefa e para um determinado tipo de cliente. Desta maneira, um tipo de cliente poderia ser, por exemplo, um 'estudante' que executaria a tarefa com limite máximo de elementos de entrada. Um outro caso pensado é a saída gerada ter um número limitado de elementos estabelecido pela política de uso para este tipo de cliente. O cliente não precisa e nem deve conhecer absolutamente nada sobre a máquina onde o procedimento numérico está sendo executado. Isto ficará a cargo exclusivamente do NC.

A solução adotada para a comunicação entre os diferentes clientes e o Numerus

Cloud é o Web Services. Web Service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia, sistemas desenvolvidos em plataformas diferentes tornam-se compatíveis. Os Web Services são componentes que permitem às aplicações enviar e receber dados em formato XML. uma linguagem universal, Os próximos capítulos mostrarão a definição, características e crescimento da *cloud computing* (computação em nuvem), os Web Services e a a história das chamadas remotas até os dias de hoje, o funcionamento e arquitetura do Numerus Cloud e seus principais aspectos. Uma análise comparativa do Numerus Cloud e *cloud computing* (computação em nuvem) com outras técnicas de *deployment* (implantação) e obtenção de alta performance no que tange a computação científica fará a conclusão deste trabalho.

# Capítulo 2

# Revisão Bibliográfica

Neste capítulo serão mostrados os principais conceitos e tecnologias necessárias para o entendimento deste trabalho. Serão apresentadas as definições e a história da *cloud computing* (computação em nuvem), da conexão remota até os Web Services. As tecnologias também serão mostradas juntamente com a perspectiva da arquitetura do Numerus Cloud.

## 2.1 *Cloud Computing* (computação em nuvem) e o Numerus Cloud

O conceito de computação na nuvem (em inglês, *cloud computing*) refere-se à utilização da memória e das capacidades de armazenamento e cálculo de computadores e servidores compartilhados e interligados por meio da Internet ([27]). O armazenamento de dados é feito em serviços que poderão ser acessados de qualquer lugar do mundo, a qualquer hora, não havendo necessidade de instalação de programas ou de armazenar dados. O acesso a programas, serviços e arquivos é remoto, através da Internet - daí a alusão à nuvem. O uso desse modelo (ambiente) é mais viável do que o uso de unidades físicas. Num sistema operacional disponível na Internet, a partir de qualquer computador e em qualquer lugar, pode-se ter acesso a informações, arquivos e programas num sistema único, independente de plataforma. O requisito mínimo é um computador compatível com os recursos disponíveis na Internet. O PC torna-se apenas um chip ligado à Internet, a "grande nuvem" de computadores, sendo necessários somente os dispositivos de entrada (teclado, mouse) e saída

(monitor). Também pode-se qualificar *cloud computing* (computação em nuvem), como o modelo mais recente de sistema distribuído que permite ao usuário acessar uma grande quantidade de aplicações e serviços de qualquer lugar e independente da plataforma, bastando para isso, ter um terminal conectado à nuvem ([32]). A expressão nuvem além de ser usada para representar um acesso remoto aos dados e serviços, remete a ideia de um ambiente desconhecido, em que é possível ver apenas o seu início e o seu fim. Por este motivo, essa nomenclatura foi muito bem usada para este novo ambiente, onde toda a infraestrutura e recursos computacionais ficam invisíveis. O usuário tem acesso apenas a uma simples interface que é disponibilizada na aplicação e no serviço. A nuvem pode ser representada pela Internet, isto é, a infraestrutura de comunicação formada por uma grande quantidade de hardwares, softwares, interfaces, redes, dispositivos tanto para controle, quanto para armazenamento, que em conjunto, permitem a entrega da computação como um serviço. Para tornar o conceito de nuvem possível, é preciso unir todas as aplicações e dados de usuários em grandes centros de armazenamento, chamados de *data centers* (Centro de Processamento de Dados - CPD). Uma vez unificados, a infraestrutura e as aplicações dos usuários são distribuídos na forma de serviços por meio da Internet. Um ponto essencial para a compreensão deste modelo de computação refere-se aos participantes da nuvem, que são divididos em três grupos: provedor de serviço, desenvolvedor e usuário. O provedor fica responsável em gerenciar, disponibilizar e monitorar toda a infraestrutura da nuvem, garantindo a segurança e a qualidade das aplicações. O desenvolvedor provê serviços para o usuário. Finalmente, o usuário consome recursos oferecidos pela nuvem.

Em linhas gerais, a computação em nuvem surge como um novo modelo de serviço preparado para fornecer processamento, infraestrutura e armazenamento de dados através da Internet.

### 2.1.1 Características do Cloud Computing (Computação em nuvem)

Segundo Pedrosa e Nogueira ([32]), as principais características para o funcionamento do cloud computing (computação em nuvem) são:

**Escalonamento** - a computação em nuvem cria a ilusão de recursos computacionais


Figura 2.1: A nuvem

infinitos para uso. Com isso, os usuários imaginam que a nuvem é capaz de fornecer rapidamente recursos em qualquer quantidade a qualquer instante. É esperado que recursos adicionais sejam providos automaticamente quando ocorrer o aumento da demanda, ou ainda no caso da diminuição da demanda. De forma resumida, escalonamento, é a forma com que a nuvem lida com o aumento ou a diminuição da necessidade de recursos ([34],[25],[43], [7]).

**Autoatendimento** - o usuário de serviços da computação em nuvem tem a expectativa de conseguir novos recursos de acordo com sua necessidade e de maneira instantânea. Para atender a esta expectativa, a nuvem computacional deve permitir o acesso em autoatendimento para que o usuário personalize, solicite e use os serviços desejados sem intervenção no ambiente ([26],[23]).

**Medição por uso** - uma vez que o usuário opte por utilizar a quantidade de recursos e serviços que julgar necessário, os serviços devem ter seu preço estabelecido com base no uso de baixa duração. Com isso, as nuvens implementam recursos que garantam um ágil comércio de serviços. Essa medição de uso dos recursos deve ser totalmente automática e de acordo com a forma e o tipo de serviço oferecido.

**Acesso a rede e customização** - Os recursos devem estar disponíveis através da rede e permitir a utilização em plataformas heterogêneas, como computadores,

celulares e outros. A customização refere-se à necessidade de personalização dos recursos da nuvem para cada usuário, desde serviços de infraestrutura até os serviços de software.

### 2.1.2 Tipologia

A computação em nuvem está dividida em quatro principais classes, considerando o nível de abstração do recurso provido e do modelo de serviço do provedor:

**IaaS** - *Infrastructure as a Service* ou Infraestrutura como Serviço: quando se utiliza uma percentagem de um servidor, geralmente com configuração que se adeque à sua necessidade.É a entrega de infraestrutura de computação como um serviço totalmente terceirizado([4] ).

**PaaS** - *Plataform as a Service* ou Plataforma como Serviço: oferece uma plataforma de desenvolvimento onde o software pode ser desenvolvido, testado e implantado, ou seja, o ciclo de vida de um software pode ser operado em um PaaS. (exemplo: Windows Azure).

**SaaS** - *Software as a Service* ou Software como Serviço: é baseado no conceito de alugar software de um determinado provedor em vez de comprá-lo como da maneira convencional (adquirindo um DVD do produto por exemplo). (exemplo: GoogleDocs, MicrosoftSharePoint Online) ([21]).

**DBaas** - *Data Base as a Service* ou Banco de dados como Serviço: quando utiliza a parte de servidores de banco de dados como serviço.

### 2.1.3 Vantagens

A maior vantagem da computação em nuvem é a possibilidade de utilizar softwares sem que estes estejam instalados no computador. Mas há outras vantagens:

Na maioria das vezes o usuário não precisa se preocupar com o sistema operacional e hardware que está usando em seu computador pessoal, podendo acessar seus dados na “nuvem computacional” independentemente disso.

As atualizações dos softwares são feitas de forma automática, sem necessidade de intervenção do usuário.

O trabalho corporativo e o compartilhamento ([33])de arquivos se tornam mais fáceis, uma vez que todas as informações se encontram no mesmo “lugar”, ou seja, na “nuvem computacional”.

Os softwares e os dados podem ser acessados em qualquer lugar, bastando que haja acesso à Internet, não estando mais restritos ao ambiente local de computação, nem dependendo da sincronização de mídias removíveis.

O usuário tem um melhor controle de gastos ao usar aplicativos, pois a maioria dos sistemas de computação em nuvem fornece aplicações gratuitamente e, quando não gratuitas, são pagas somente pelo tempo de utilização dos recursos. Não é necessário pagar por uma licença integral de uso de software.

Diminui a necessidade de manutenção da infraestrutura física de redes locais cliente/servidor, bem como da instalação dos softwares nos computadores corporativos, pois esta fica a cargo do provedor do software em nuvem, bastando que os computadores clientes tenham acesso à Internet.

A infraestrutura necessária para uma solução de cloud computing (computação na nuvem) é bem mais enxuta do que uma solução tradicional de hosting (hospedagem) , consumindo menos energia, refrigeração e espaço físico e consequentemente contribuindo para preservação e uso racional dos recursos naturais.

### 2.1.4 Desvantagens

A maior desvantagem da computação em nuvem, vem fora do propósito desta, que é o acesso à Internet. Caso o acesso seja perdido ou quebrado, todos os sistemas conectados (que dependam de uma conexão) ficarão comprometidos. Por essa razão, a disponibilidade é uma das maiores apreensões do cloud computing (computação em nuvem). Uma possível solução é ter mais de um prestador e, consequentemente, mais de uma nuvem. O que tornaria possível deixar o usuário executar suas aplicações em outra nuvem, enquanto a primeira está fora do ar. Contudo, esta alternativa não é tão simples, pois requer a interoperabilidade entre as nuvens([6]).

Outras desvantagens:

Velocidade de transferência dos dados: caso seja necessário uma grande taxa de transferência, o sistema pode ser comprometido.

Custo: assim como todo tipo de serviço, ele é custeado.

Segurança: maior risco de comprometimento da privacidade do que em armazenamento local.

### 2.1.5 Numerus Cloud Computing

O Numerus Cloud herda todas as vantagens oferecidas pela cloud computing (computação na nuvem), principalmente a vantagem do acesso aos dados e aplicações dos usuários de qualquer localização geográfica pela Web, resultando em flexibilidade e mobilidade aos usuários. Há uma vantagem adicional que o modelo do Numerus Cloud oferece, usando a característica do modelo distribuído sob o qual o Numerus Cloud se baseia e que permite o isolamento das camadas componentes da arquitetura: a proteção do software numérico que é executado numa máquina remota. A segurança com relação aos dados trafegados é um problema enfrentado pela computação em nuvem pois, a informação antes armazenada localmente, irá localizar-se na nuvem em um local físico desconhecido. Em relação à privacidade e consistência das informações, é visto que nuvens públicas possuem uma grande exposição a ataques. Para amenizar esse problema, exige-se uma criptografia dos dados e um sistema de gerenciamento para cópias de segurança. Vale ressaltar que esta proteção reside em cima dos dados pois são eles os únicos que trafegam como requisição e resposta dos

serviços. Mas para o Numerus Cloud, a segurança com relação ao acesso ao software do autor da rotina numérica existe de forma automática, garantida pela arquitetura de isolamento proposta e desenvolvida.

### 2.1.6 História

**Anos 1950s:**

O conceito de cloud computing (computação na nuvem) remonta à década de 1950, quando os computadores *mainframe* (computador de grande porte, dedicado normalmente ao processamento de um volume grande de informações) começaram a ser usados em universidades e empresas, acessíveis via *thin clients* ("cliente magro") - computadores terminais, muitas vezes chamados de "terminais burros", pois eles eram usados para comunicação, porém sem capacidade de processamento interno. Para fazer uso mais eficiente dos caros *mainframes*, permitiu-se que vários usuários compartilhassem o acesso físico ao computador a partir de vários terminais, bem como compartilhassem o tempo de CPU. Isso eliminou os períodos de inatividade do *mainframe* permitindo um maior retorno sobre o investimento no *mainframe*. A prática de compartilhar o tempo de CPU em um *mainframe* ficou conhecida na indústria como *time-sharing* (sistema de tempo compartilhado).

**Anos 1960s 1990s:**

John McCarthy opinou na década de 1960 que a "computação poderia um dia ser organizada como uma utilidade pública" ([18]) . Quase todas as características modernas de cloud computing (computação na nuvem) (escalonamento, recursos providos de acordo com a demanda e ilusão de recursos computacionais infinitos para uso, serviço como um serviço utilitário, serviços on line e a comparação com uma indústria de fornecimento de eletricidade) foram exploradas no livro de Douglas Parkhill de 1966, *The Challenge of the Computer Utility*. A tendência era o retorno a 1950 quando o especialista em computação Herb Grosch postulou que o mundo inteiro iria operar em terminais burros alimentados por cerca de 15 grandes *data centers* ([15]). Grosch enunciou o que mais tarde ficou conhecida como a lei de Grosch: "O poder computacional de um processador é proporcional ao quadrado de seu preço", ou seja, pagando duas vezes mais, pode-se obter o quádruplo da perfomance. Esta observação encaixou-se muito bem, na tecnologia do *mainframe*, e fez

com que a maioria das organizações viesse a comprar a maior máquina que pudesse pagar.

**Anos 1990s:**

Nos anos 1990, as empresas de telecomunicações começaram a oferecer serviços de VPN (*Virtual Private Network* - Rede Privada Virtual) com qualidade de serviço razoável e com baixo custo. Como os computadores se tornaram mais predominantes, cientistas e técnicos da computação melhoraram a infraestrutura existente fazendo com que mais usuários tivessem acesso à computação de larga-escala através do *time-sharing*.

**Desde 2000:**

Após a bolha dos ".com", a Amazon teve um papel fundamental em todo o desenvolvimento da cloud computing (computação na nuvem), modernizando seu *data center* (que operava na época com apenas 10% de sua capacidade) resultando em melhoria dos serviços internos. Em 2006 a Amazon lançou o Amazon Web Services(AWS) que fornecia cloud computing (computação na nuvem) para clientes externos. No início de 2008, o Eucalyptus se tornou o primeiro software *open source* (código aberto) compatível com AWS. Em Março de 2011, a IBM anunciou o framework "IBM SmartCloud" ([8],[13],[24]).

### 2.1.7 Crescimento de popularidade

As redes de alta capacidade, computadores e dispositivos de armazenamento de baixo custo, o uso de máquinas virtuais (virtualização), arquitetura orientada a serviço (o Web Service é um pequeno exemplo desta arquitetura) e a *utility computing* (computação sob demanda) levaram a um crescimento da cloud computing (computação na nuvem)([1], [20]) .

## 2.2 Web Services

A solução proposta para a comunicação entre os diferentes clientes e o Numerus Cloud Controller é o Web Services. Web Service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecno-

logia, é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Sendo assim, a tecnologia deve garantir que todos os fornecedores possam suportar as mesmas especificações dos Web Services. Dessa forma, um serviço escrito em uma linguagem pode implementar a especificação para um Web Service da mesma forma que um escrito em uma linguagem diferente. Os Web Services são componentes que permitem às aplicações enviar e receber dados em um formato qualquer, usualmente no formato XML (*EXtensible Markup Language*). Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato XML([39]). Para as empresas, os Web Services podem trazer agilidade para os processos e eficiência na comunicação entre cadeias de produção ou de logística. Toda e qualquer comunicação entre sistemas passa a ser dinâmica e principalmente segura, pois não há intervenção humana. Essencialmente, o Web Service faz com que os recursos da aplicação do software estejam disponíveis sobre a rede de uma forma normalizada. Outras tecnologias fazem a mesma coisa, como por exemplo, os *browsers* (navegadores) da Internet acessam às páginas Web disponíveis usando por norma as tecnologias da Internet, HTTP e HTML. No entanto, estas tecnologias não são bem sucedidas na comunicação e integração de aplicações. Utilizando a tecnologia Web Service, uma aplicação pode invocar outra para efetuar tarefas simples ou complexas mesmo que as duas aplicações estejam em diferentes sistemas e escritas em linguagens diferentes. Em outras palavras, os Web Services fazem com que os seus recursos estejam disponíveis para qualquer aplicação cliente.

### 2.2.1 História

Até chegar à solução dos Web Services, houve o desenvolvimento e amadurecimento de diversas soluções. A chamada de procedimentos remotos (RPC, *Remote Procudure Call*) é uma dessas soluções. Serão mostrados uma pequena história e os fatos cronológicos que pontuaram o acesso remoto às aplicações.

#### Uma história da RPC

**Definição:** Chamada remota de procedimento (RPC, acrônimo de *Remote Procedure Call*) é uma tecnologia de comunicação entre processos que permite a um pro-

grama de computador chamar um procedimento em outro espaço de endereçamento (geralmente em outro computador, conectado por uma rede). O programador não se preocupa com detalhes de implementação dessa interação remota: do ponto de vista do código, a chamada se assemelha a chamadas de procedimentos locais ([2]). Quando o software em questão usa os princípios da orientação a objeto , RPC é chamada de *remote invocation* ou *remote method invocation* (RMI - invocação de método remoto). Portando, os Web Services foram a escolha natural para a comunicação no Numerus Cloud pois atende a proposta de baixo acomplamento entre as camadas componentes da arquitetura e permitindo que qualquer programa cliente acesse o Numerus Cloud.

**História:** A idéia de RPC data de 1976, quando foi descrito no RFC 707 ([41]) (RFC ou *Request for Comments* é um documento que descreve os padrões de cada protocolo da Internet). O nome *Remote Procedure Call* é creditado à Bruce Jay Nelson ([5] ), cientista ligado à computação. Durante o seu PhD, Nelson trabalhou na Xerox PARC onde desenvolveu o conceito do RPC. Um dos primeiros usos comerciais da tecnologia foi feita pela Xerox no "Courier", de 1981. A primeira implementação popular para Unix foi o Sun RPC (atualmente chamado *ONC RPC Open Network Computing*), usado como base do *Network File System* e que ainda é usada em diversas plataformas. Como mais uma informação, em meados da década de 80, a Sun Microsystems usa como sistema operacional de suas *workstations* (estações de trabalho) o BSD Unix estendido com RPC. Outra implementação pioneira em Unix foi o *Network Computing System* (NCS) da Apollo Computer, que posteriormente foi usada como fundação do DCE/RPC no Distributed Computing Environment (DCE). Uma década depois a Microsoft adotou o DCE/RPC como base para a sua própria implementação de RPC, MSRPC. O DCOM foi implementada com base nesse sistema ([11]).

(Observação: DCOM - acrônimo para *Distributed Component Object Model* - é uma tecnologia proprietária da Microsoft para criação de componentes de software distribuídos em computadores interligados em rede. O DCOM é uma extensão do COM (também da Microsoft) para a comunicação entre objetos em sistemas distribuídos. A tecnologia foi substituída, na plataforma de desenvolvimento .NET, pela API .NET Remoting e empacotada no WCF. O DCOM pode ser utilizado

na construção de aplicações em três camadas, de forma a centralizar as regras de negócio e processos, obter escalabilidade e facilitar a manutenção).

Na década de 1990, o ILU da Xerox PARC e o CORBA ofereciam outro paradigma de RPC baseado em objetos distribuídos, com mecanismos de herança.

Desde de seu início proposto por B.J. Nelson e nos 15 anos seguintes, várias evoluções ocorreram no sistema RPC básico levando ao desenvolvimento de sistemas bem melhores como o NCS que oferecia aos programadores mais funcionalidade e maior simplicidade. O CORBA - *Common Object Request Broker Architecture* do grupo *Object Management Group* e o DCOM da Microsoft são exemplos do resultado deste processo evolutivo ([19]). Com a introcução do *Java Developer's Kit release 1.1* (JDK 1.1), uma terceira alternativa para a criação de aplicações distribuídas surgiu. O sistema *Java Remote Method Invocation* (RMI) possui muitas características de outros sistemas RPC onde um objeto rodando em uma JVM (*Java Virtual Machine*) faz uma chamada de método de um outro objeto rodando em uma outra JVM e, talvez, em uma outra máquina física. (De forma análoga, atualmente utiliza-se XML como linguagem de descrição de interface e HTTP como protocolo de rede para formar Web Services, cujas implementações incluem SOAP e XML-RPC).

O XML marca o início da tecnologia dos Web Services. O passo seguinte foi o desenvolvimento de um protocolo padronizado baseado em XML que possibilitasse a troca de mensagens ([36]).

A IBM e a Microsoft trabalharam na elaboração de uma forma que descrevesse uma conexão em Web Services. A especificação desenvolvida por essas duas empresas foi fundida resultando na WSDL (*Web Services Description Language*). IBM, Microsoft e Ariba iniciaram os trabalhos sobre uma solução que permitisse a descoberta de serviços Web disponíveis. Em 2000, foi anunciada a versão 1.0 do UDDI (*Universal Description, Discovery e Integration*).

### 2.2.2 Padrão

O W3C (*World Wide Web Consortium*) e o OASIS (*Organization for the Advancement of Structured Information Standards*) são as instituições responsáveis pela padronização dos Web Services. Empresas como IBM e Microsoft, duas das maiores do setor de tecnologia, apoiam o desenvolvimento deste padrão. Segundo o W3C ,

um Web Service define-se como: um sistema de software projetado para suportar a interoperabilidade entre máquinas sobre rede. Tem uma relação descritiva num formato *machine-processable*, especificamente WSDL. Outros sistemas interagem com o Web Service usando as mensagens SOAP, tipicamente sobre HTTP com XML na junção com outros padrões da Web.
Existem cinco padrões fundamentais de WebServices:

**XML (*EXtensible Markup Language*):** é usado como um formato geral para descrever modelos, formatos e tipos de dados.

**HTTP (*Hypertext Transfer Protocol*) (*HTTPS - HyperText Transfer Protocol Secure*):** é um protocolo de baixo nível usado pela internet, onde é possível transferir os dados para os WebServices pelas redes.

**WSDL (*Web Services Description Language* - Linguagem de Definição de WebServices):** é usado para definir as interfaces dos serviços, ou seja, define a sua assinatura (nome dos métodos e parâmetros de entrada e saída) e seus detalhes de ligação e deploy (protocolo e localização)([40]).

**SOAP (*Simple Object Access Protocol* - Protocolo Simples de Acesso a Objetos):** é um padrão que define o protocolo de WebServices, enquanto o HTTP é um protocolo de baixo nível, SOAP é o formato específico para trocar dados de WebServices.

**UDDI (*Universal Description, Discovery e Integration* - Descrição Universal, Descoberta e Integração):** é um padrão para o gerenciamento de WebServices, ou seja, registrar e localizar serviços. Desempenha um papel auxiliar que não é exigido.

As bases para a construção de um Web Service são os padrões XML e SOAP ([10]). O transporte dos dados é realizado normalmente via protocolo HTTP ou HTTPS para conexões seguras (o padrão não determina o protocolo de transporte).

Os dados são transferidos no formato XML, encapsulados pelo protocolo SOAP. Adiante serão vistas mais definições sobre as tecnologias que caracterizam os Web Services, vantagens e desvantagens e o porque do uso da tecnologia dos Web Services para o Numerus Cloud.

Os Web Services são identificados por um URI (*Uniform Resource Identifier*), descritos e definidos usando XML.

### 2.2.3 O porque do uso da tecnologia dos Web Services no Numerus Cloud

Um dos motivos que tornam os Web Services atrativos é o fato deste modelo ser baseado em tecnologias que são padrões, em particular XML e HTTP. Os Web Services são utilizados para disponibilizar serviços interativos na Web, podendo ser acessados por outras aplicações usando, por exemplo, o protocolo SOAP.
O Numerus Cloud não faz distinção de nenhum cliente e propõe o isolamento entre as suas camadas componentes. O Web Service permite o baixo acoplamento entre cliente e servidor e é acessado por qualquer aplicação. O Web Service trabalha em cima do protocolo de transporte HTTP que é largamente usado na cloud computing (computação na nuvem) e permite a conexão entre os clientes da nuvem. Para um cliente acessar os serviços do Numerus Cloud , basta possuir acesso ao arquivo WSDL que descreve os serviços, os parâmetros de entrada e as respostas. Os Web Services tornam-se, com o protocolo HTTP, uma evolução do RPC para uso na nuvem.
O RMI carrega as mesmas premissas do RPC, mas possui a restrição de conectar apenas programas em Java, razão pela qual não pode ser usado pelo Numeus Cloud.

## 2.3 Soluções de Implantação

Nessa seção são descritos métodos de implantação de rotinas numéricas.

### 2.3.1 Executável Monolítico

Um executável monolítico é um programa num arquivo único executável. Implantar um programa como esse, em teoria, é fácil, pois basta copiar o programa para o

cliente e executar o único arquivo existente.

O processo de implantação é trabalhoso, pois no caso da primeira implantação ou da atualização do aplicativo, é preciso copiar fisicamente o arquivo para todas as pessoas que vão usar o programa (podem ser numerosas pessoas).

O processo é também problemático, pois cada máquina de cliente tem características próprias (versão do sistema operacional, BIOS, CPU, GPU, memória, disco, e outras). A diversidade de máquinas pode causar problemas de compatibilidade. Além disso, há o fato de que se entrega fisicamente o módulo compilado para a pessoa que vai usar a rotina, e dessa forma se deixa todo o controle sobre a execução para esta última. Isso é equivalente a dizer que não há forma implícita de se prevenir de ataques que modifiquem a rotina implantada (por exemplo com objetivo de violar a propriedade intelectual).

### 2.3.2 Executável com Rotinas em Bibliotecas de Ligação Dinâmica

Esse caso consiste em um programa executável que tem dependência de bibliotecas com ligação dinâmica (*.dll no sistema Windows ou *.so no sistema Linux). Da mesma forma como no caso do executável monolítico, implantar um programa como esse, em teoria, é fácil, pois basta copiar os arquivos do programa para o cliente e executar o programa principal.

Como no caso anterior, o processo de implantação é trabalhoso, pois no caso da primeira implantação ou da atualização do aplicativo é preciso copiar fisicamente os arquivos para todas as pessoas que vão usar o programa (podem ser numerosas pessoas).

Como no caso anterior, o processo tem os mesmos problemas de compatibilidade de máquina. O problema de fragilidade ao ataque de propriedade intelectual é também o mesmo.

A diferença básica entre essa solução a anterior é que pode-se atualizar apenas a biblioteca dinâmica sem atualizar os demais arquivos do sistema.

### 2.3.3 Executável Chamando Rotina Remota

Esse é o caso mais sofisticado em que a execução da rotina numérica ocorre fisicamente num computador diferente do computador do cliente. Trata-se portanto de um sistema com 2 camadas - uma camada de cliente e uma camada para execução remota. Nesse caso, surgem várias modificações que podem, em casos específicos, ser consideradas como vantagens.

- Como a rotina é executada no servidor, não há problema de compatibilidade com as várias máquinas de clientes diferentes.
- O cliente não tem acesso direto a rotina numérica, e com isso, se obtém uma camada adicional para proteção da propriedade intelectual.
- É imediato e garantidamente padronizado o processo de atualização da rotina. Por se atualizar a rotina no servidor central, é garantido que todos os clientes farão uso da última versão da rotina implantada.

Há também desvantagens com essa arquitetura

- A centralização do processamento numérico numa mesma máquina servidora tende a criar problemas sérios de escalabilidade.
- No momento de atualização da rotina numérica, é necessário se fazer *upload* (envio) do novo pacote compilado e reinicialização da aplicação que contém a rotina numérica. Isso causa interrupção temporária da aplicação do servidor.
- O autor da rotina numérica precisa entregar o arquivo executável para a pessoa que controla o computador servidor. Pode haver constrangimento de segurança do autor em realizar essa entrega, pois quem controla o computador servidor pode violar a propriedade intelectual do autor da rotina numérica.
- O formato técnico do arquivo da rotina numérica precisa ser compatível com o do servidor de execução remota (se for um servidor de aplicação em Java, um formato popular é o *.war). O empacotamento da rotina numérica nesse formato não costuma ser bem aceito para uma pessoa com perfil de autor de rotina numérica. A necessidade de se fazer num mesmo pacote ambas as

tarefas (empacotar para servidor de aplicação e a execução da rotina numérica ela própria) é uma dificuldade de implantação.

# Capítulo 3

# A proposta principal: o Numerus Cloud

A motivação para esse trabalho deveu-se a observação de que a implantação de procedimentos numéricos é em geral muito trabalhosa ou problemática. Isso pode ser confirmado por literatura ( [28], [37], [12]) e também por experiência pessoal.

## 3.1 Objetivos e Requisitos

A motivação do NumerusCloud é propor uma arquitetura de software para automatização da implantação de rotinas numéricas. A arquitetura proposta deve ser capaz de entregar as vantagens da solução descrita em 2.3.3, sem as desvantagens associadas.

O Numerus Cloud tem como finalidade a automatização do *deployment* (implantação) de procedimentos numéricos utilizando o *cloud computing* (computação em nuvem). Usando as características da *cloud computing* (computação em nuvem) de virtualização do hardware e software de suporte envolvidos, a complexidade relacionada ao *deployment* (implantação) de uma aplicação e à infraestrutura é bastante simplificada , fazendo com que não haja a necessidade de um conhecimento profundo sobre as características técnicas por parte do cliente ou usuário. Além disso, o Numerus Cloud, permite um fácil acesso aos seus serviços oferecidos através de uma forma padronizada que é o Web Services.

## 3.2 A arquitetura do Numerus Cloud

Mas é a arquitetura do Numerus Cloud que permite um forte isolamento entre o autor do programa numérico e o cliente que requisita este programa e que faz com que o *deployment* (implantação) do programa numérico seja bastante rápido e simples.

**A arquitetura proposta contém 3 personagens com papéis bem definidos:**
**O cliente (EUD - *End User Device* - Dispositivo do Usuário Final)** que faz a requisição do método numérico a partir de qualquer dispositivo.
**O NCC (*Numerus Cloud Controller* - Controlador do Numerus Cloud)** que aje como camada controladora recebendo todas as requisições dos clientes, gerenciado-as e repassando-as para as máquinas administradas pelos autores das tarefas numéricas.
**O agente (NTE - *Numeric Task Executor* - Executor da Tarefa Numérica)** residente na máquina onde será executada a tarefa numérica.

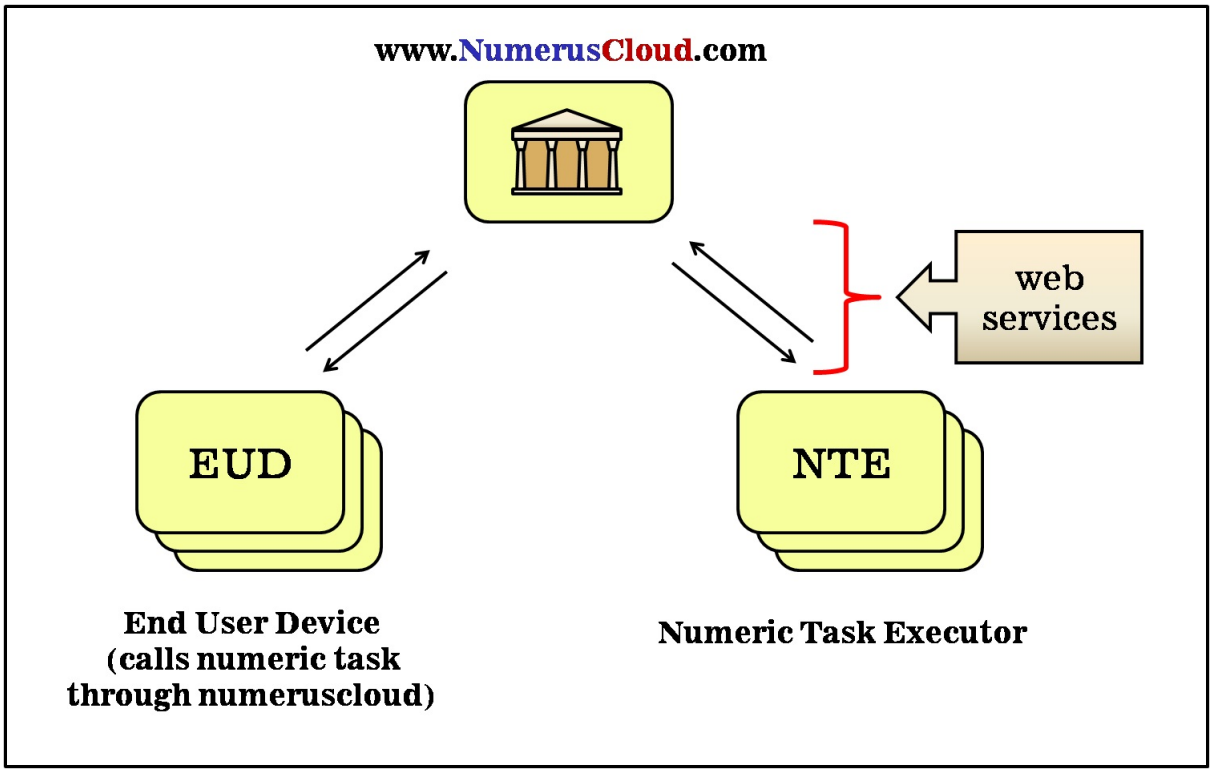


Figura 3.1: Numerus Cloud

## 3.3 O funcionamento

Um autor de métodos numéricos prepara seu procedimento numérico (tarefa numérica) que é executada num computador de seu controle. Para disponibilizá-la para uso, o autor registra em seguida esta mesma tarefa e outras informações como caminhos de diretórios de entrada e saída de dados, nome do autor, versão do programa, o formato de entrada e caminho do programa executável no Numerus Cloud (Numerus Cloud Controller). Caso queira, define e registra uma política de uso para este procedimento, que é função da própria tarefa e do tipo de cliente que quer usá-la. No computador controlado pelo autor estará residente um programa agente (NTE). Este agente será instalado pelo autor ou pelo administrador da máquina e é este agente, o terceiro personagem da arquitetura NC, que chamará a tarefa, funcionando como porta de entrada na máquina do autor. Portanto, o agente será o responsável pelo recebimento de requisições provenientes da NC e pela execução das tarefas registradas e relacionadas às requisições.

A tarefa numérica pode ser qualquer procedimento numérico que recebe como entrada dados ou bytes sob um determinado formato fornecendo dados lógicos como resposta.

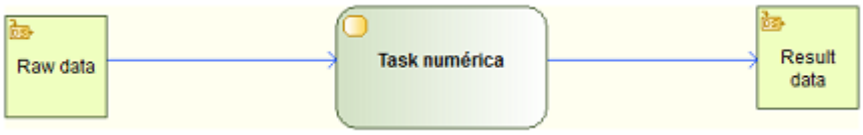


Figura 3.2: tarefa numérica

O uso do procedimento numérico é disponibilizado via Web Services pela NC. O cliente final roda um programa na sua própria máquina ou em qualquer outro dispositivo que chama o procedimento numérico através do NC. O cliente não precisa e nem deve conhecer absolutamente nada sobre a máquina onde o procedimento numérico está sendo executado. O cliente enxerga apenas o Numerus Cloud e dele requisita o serviço.

Duas das principais características do *cloud computing* (computação em nuvem) são: poupar ou economizar o processamento e o uso de recursos locais e a transparência para o cliente em relação à localização real do(s) computador(es) onde

realmente está sendo processado o método numérico. As características da máquina que faz o serviço, bem como os aspectos de hardware e de rede (infraestrutura) não são levados em consideração, mas sim que "alguém" executará o serviço requisitado pelo cliente.

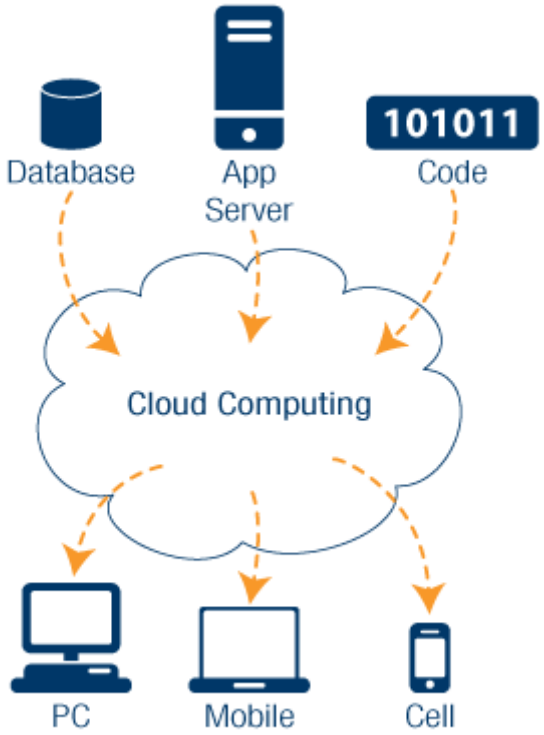


Figura 3.3: Isolamento na nuvem

O cliente que requisita o serviço não sabe quem realmente o executará e tão pouco o servidor do serviço sabe quem é o usuário cliente que requisitou tal serviço.

## 3.4 Política de uso

O autor pode criar uma política de uso que varia em função da tarefa e do tipo de cliente. Desta maneira, um tipo de cliente poderia ser, por exemplo, um 'estudante' que apenas executaria a tarefa a partir de um numero máximo de elementos de entrada (*input*) estabelecido pela politica de uso para este tipo de cliente e registrado no Numerus Cloud. A política de uso poderia ser um script contendo restrições e informações e seria sempre executado pelo Numerus Cloud. Este script controlaria "o que e como" cada cliente poderia chamar o serviço de acordo com a tarefa desejada.

## 3.5 Casos de uso

A análise do funcionamento do Numerus Cloud é sempre feita a partir da noção de caso de uso e seguindo esta proposta, qualquer serviço requisitado pelo cliente estará sob a forma de um caso de uso. Sendo assim, existirão casos de uso relacionados à execução de uma tarefa de negócio (a tarefa numérica) e casos de uso relacionados com as operações de inclusão, alteração, exclusão e consulta de uma tarefa.

**A execução completa de uma tarefa envolve os seguintes casos de uso:**
*Login* ou autenticação no NC.
A chamada da tarefa e a criação de um *task process id*, um identificador único de uma instância da tarefa selecionada.
O upload de arquivos com dados de entrada.
A execução da tarefa.
O download do resultado.

Mas existem outros casos de usos que não estão relacionados diretamente com a execução da tarefa numérica. Por exemplo, um cliente poderá consultar a partir de uma palavra chave, a lista das tarefas relacionadas a esta palavra. Ou mesmo, um cliente poderá pedir pela NC qual o formato exigido para os parâmetros de entrada de dados que estarão nos arquivos a serem enviados para uma determinada tarefa.

## 3.6 Autenticação (login)

Um cliente necessita, na resolução de seu sistema, que uma determinada tarefa numérica seja executada. Na visão do cliente, a tarefa está localizada na NC, isto é, a tarefa pertence à NC e é a NC que a executará. Primeiramente, o cliente fará o *login* (autenticação) na NC e, para tanto, fornecerá seu nome de usuário (*username*) e senha (*password*). O *login* (autenticação) é o caso de uso inicial que permite a entrada do usuário na NC. Após o *login* (autenticação), a NC autoriza o usuário a realizar as chamadas aos serviços que tem permissão. O usuário recebe como retorno do *login* (autenticação) uma *session key* (*session id* ou identificador de sessão) que é uma chave usada por ele para outras requisições, como o upload de novos arquivos de

dados, a chamada da execução da tarefa e o download do resultado. É importante notar que os formatos sob os quais os arquivos de entrada (*input*) e o resultado (*output*) estão dispostos são fornecidos pelo autor da tarefa. O cliente fornecerá dados de entrada sempre através de upload de arquivos.

## 3.7 Identificador de Sessão e Identificador de Instância de Tarefa

Para qualquer serviço requisitado, o cliente deve fornecer o *session id* (identificador de sessão) obtido inicialmente. É através desse *session id* (identificador de sessão) que o NC controla a sessão com o cliente. O *session id* (identificador de sessão) possui um *timeout* (tempo limite) até a chamada da execução da tarefa. Após a chamada, o *timeout* (tempo limite) recomeçará a valer após o término da tarefa numérica. Quando o cliente informa a tarefa que deseja, o NC cria um *task process id* (identificador de instância de tarefa) referente à instância da tarefa que será executada, guarda o *task process id* (identificador de instância de tarefa) na sessão e retorna este id para o cliente. Neste ponto, os serviços chamados terão como argumentos básicos o *session id* (identificador de sessão) e o *task process id* (identificador de instância de tarefa).

Na sequência, o cliente pode executar tantos uploads de arquivo quanto quiser. A origem deste upload é um *stream* (fluxo) de bytes qualquer: arquivo em disco, dados de um GUI (*Graphical User Interface* - Interface Gráfica com o Usuário), um *network stream* (fluxo de bytes pela rede), etc. Os arquivos que são enviados chegam até a NC que os envia para o agente NTE responsável. O agente finalmente copia os arquivos para a máquina administrada pelo autor da tarefa. Depois da conclusão do upload do arquivo de entrada, o cliente chama a execução da tarefa. O NC recebe a requisição de execução da tarefa escolhida, verifica qual é o agente responsável pela execução da tarefa e, uma vez localizado, submete a tarefa para o agente (o upload do arquivo de entrada e a execução da tarefa são assíncronos). O cliente usa o *task process id* (identificador de instância de tarefa) para acompanhar o processo e executar o download do resultado.

O agente residente em cada máquina é genérico. Recebe as requisições vindas do NC

e executa a tarefa numérica. O agente nada conhece sobre qualquer característica ligada ao negócio que a tarefa numérica executa. A requisição vinda da NC já carrega consigo todos os atributos necessários para a execução da tarefa, como o caminho do programa executável, o diretório de entrada de dados, o diretório de saída e também o *task process id* (identificador de instância de tarefa) criado pela NC e que fora retornado para o cliente. Tais atributos (com exceção do *task process id* (identificador de instância de tarefa)) foram estabelecidos quando do registro na NC da tarefa numérica pelo autor.

## 3.8 DOUA, uma Camada de Software para Controle de Acesso para Cloud

O kit de desenvolvimento de software para *cloud computing* (computação em nuvem) que foi utilizado nesse trabalho é baseado nos produtos de software listados abaixo:

1. linguagem de programação Java
2. MySQL
3. Netbeans
4. Tomcat (servidor web baseado em Java)
5. axis2 (biblioteca para Web Services por cima do Tomcat)
6. plugin axis2 para Netbeans [29]

O uso desses produtos de software conduz à geração de um serviço na nuvem, baseado na tecnologia Web Services, com os dados trafegando em formato XML. O serviço assim criado é desprovido de controle de acesso, isto é, a qualquer momento um cliente pode acessar a url do serviço passando os parâmetros de entrada, e o serviço proverá as saídas correspondentes. Os serviços implementam os casos de uso do sistema. Serviços na nuvem assim definidos são equivalentes a itens de uma API, tal como um sistema operacional.

Por vários motivos, é necessário que haja controle no acesso desses serviços, dependente do "ator" (também chamado de "tipo de usuário"). Para a inclusão

desse controle , foi proposto uma camada reutilizável de software e que foi batizada de DOUA (*Database Oriented Usecase Authorization*). A proposição original do DOUA foi feita em [38]. Essa camada cria o conceito de sessão, que é criada após o usuário ser autenticado por senha. O NumerusCloud teve o acesso dos seus casos de uso controlado pela camada DOUA. É relativamente fácil reutilizar o DOUA como camada a controlar acesso de casos de uso para um software genérico na nuvem, usando os produtos de software listados.

A ideia básica do DOUA é que um sistema de software contém atores e casos de uso. Cada usuário do sistema deve ser de um dos tipos (atores) definidos. Deve haver uma descrição das permissões dos casos de uso por ator em algum lugar, de forma que o sistema saiba para cada caso de uso, se o usuário tem permissão de execução ou não. É de especial importância que essa descrição NÃO seja implementada com codificação (e.g. Java), mas sim por dados que estejam persistidos em algum tipo de banco de dados ou arquivo. Isso porque um sistema pode requerer por questão de manutenção a alteração da descrição das permissões de caso de uso por ator. Usando-se o DOUA, isso é feito apenas por se alterar os dados, sem necessidade de se recompilar o programa, o que é muito mais simples.

A camada DOUA autoriza ou não um usuário a executar determinado serviço do Numerus Cloud. Quando o usuário faz o *login* ou a autenticação no sistema, são guardadas em sessão as informações deste usuário, inclusive seu id responsável pela sua identificação no controle de acesso. O objetivo do *session id* (identificador de sessão) é identificar a sessão do usuário. Porém, o que se deseja da sessão com o usuário, é o seu id ou usuário id guardado. Através deste id ou usuário id, o Numerus Cloud consultará o sistema de controle de acesso na obtenção dos serviços ou recursos que o usuário em questão tem autorização de uso. Originalmente pensou-se em estabelecer as entidades ligadas ao controle de acesso numa base de dados relacional. Porém, a implementação do controle de acesso foi toda feita a partir de interfaces , o que permitiu o uso de qualquer sistema de persistência. No exemplo do Numerus Cloud implementado, foi usado como estrutura de dados, arquivos XML. A nomenclatura DOUA tem origem no controle de acesso baseado num banco de dados relacional, mas, apesar de outras estruturas usarem o conceito, a sigla permaneceu.

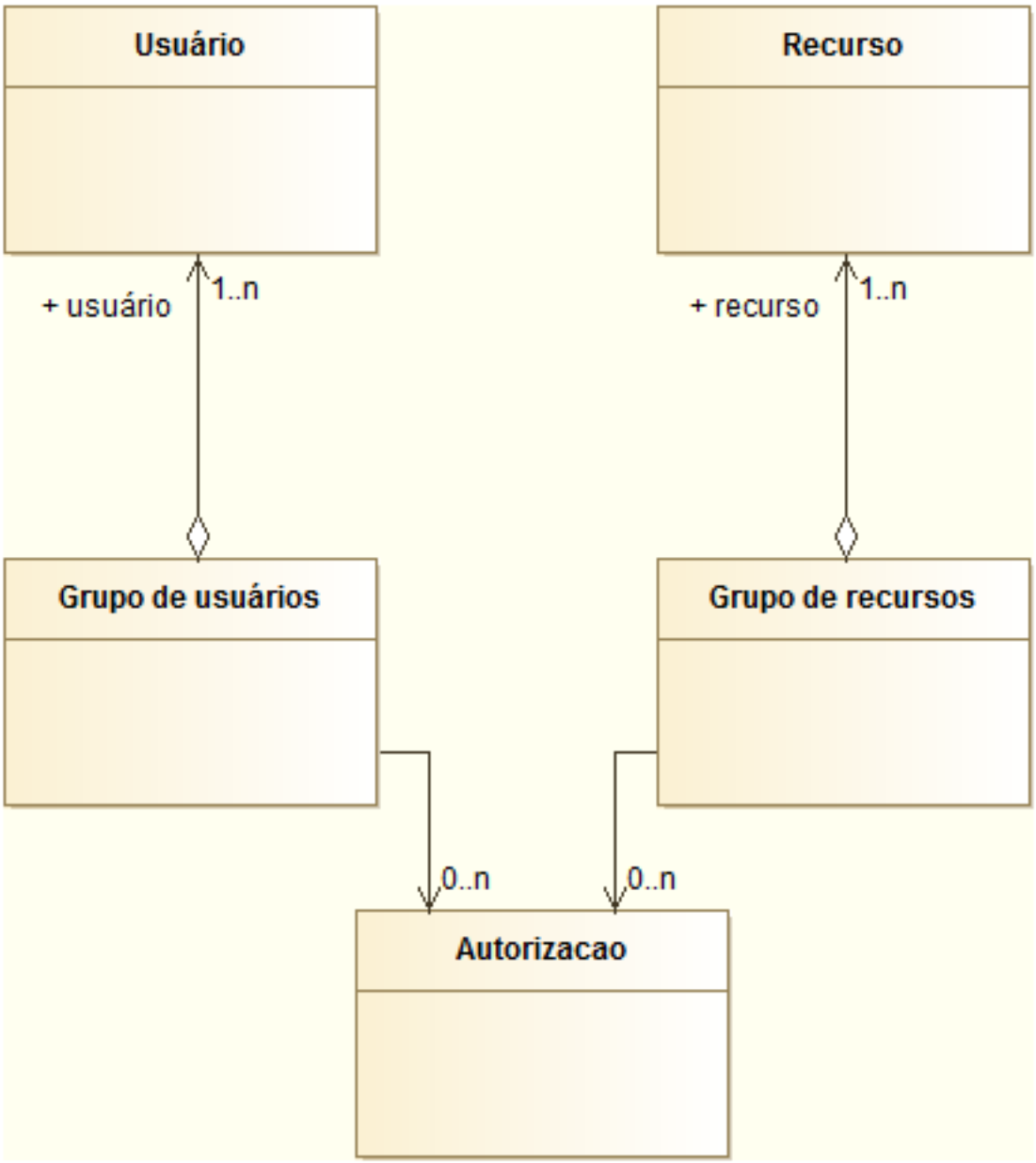


Figura 3.4: Controle de acesso

O controle de acesso possui basicamente cinco entidades principais. Um grupo de usuários contém um ou mais usuários. Um grupo de recursos contém um ou mais recursos. Os recursos podem ser os métodos ou serviços. Por fim, uma determinada autorização contempla um grupo de usuários e um grupo de recursos. Desta maneira, seja uma autorização qualquer A1. Se A1 possuir o grupo de usuários GU1 e possuir o grupo de recursos GR1, todos os usuários de GU1 irão acessar todos os recursos de GR1.

O controle de acesso é uma camada desacoplada e pode ser usada por qualquer programa, inclusive por um Web Services. O módulo DOUA foi acoplado ao sistema Numerus Cloud. A camada Controller do Numerus Cloud usa o DOUA e não sabe como foi feita a implementação do DOUA. No nosso caso, foi feita uma implementação DOUA usando o XML como estrutura dos dados conforme o diagrama apresentado acima.

## 3.9 Comunicação com Web Services e JSON

Até aqui tem-se a clássica formação MVC (*model-view-controller* - modelo-visão-controle) com o NC assumindo o papel de *controller*, o cliente o papel de *view* e o agente juntamente com o programa numérico responsáveis pelas regras de negócio (camada *model*). Mas esta arquitetura possui uma característica comum quanto à comunicação entre as camadas. As comunicações entre o cliente e o NC e entre o NC e a máquina controlada pelo autor são feitas através de Web Services (como uma ressalva, estabelecemos que o formato de dados usado na troca de informações entre a NC e o agente da máquina cliente foi o JSON - *Javascript Object Notation*. Este formato foi escolhido pela maior quantidade de informações de conteúdo por tamanho em bytes). Web Services são agentes integradores que realizam o compartilhamento de dados entre um indeterminável número de aplicações. Sua principal característica é a capacidade de comunicação entre programas que rodam em sistemas e plataformas heterogêneos, permitindo assim um forte desacoplamento entre os programas. A escolha de Web Services como a forma de comunicação entre as camadas do Numerus Cloud foi portanto natural, pois tem como protocolo de comunicação o HTTP e a linguagem XML para estruturação e armazenamento de dados.

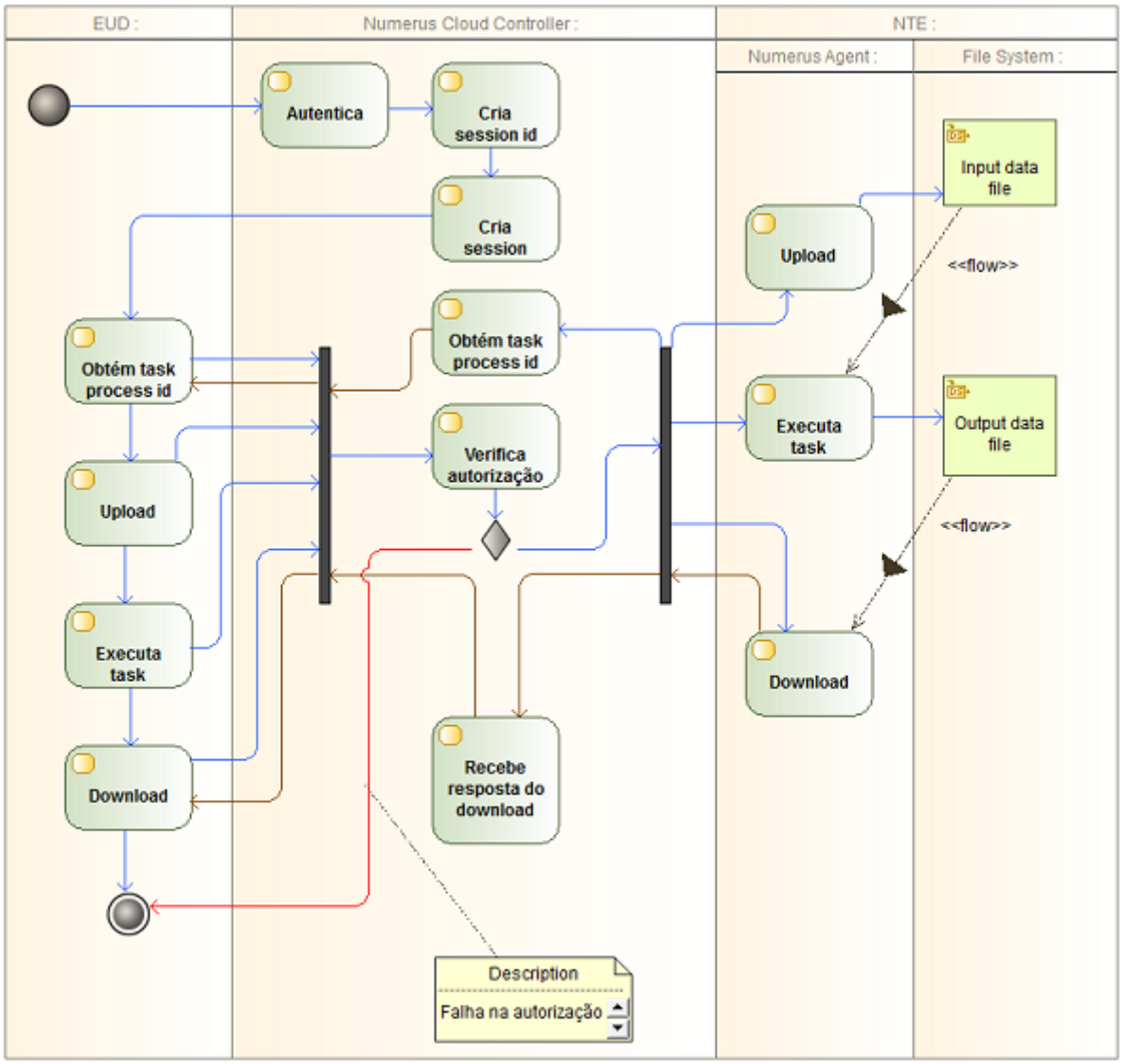


Figura 3.5: Autenticação e autorização

O protocolo HTTP é amplamente divulgado e conhecido. A linguagem XML para realizar a estruturação e armazenamento das informações é universal e bastante aceita sendo que, praticamente, todas as linguagens de programação possuem bibliotecas para o tratamento e *parsing*(análise sintática) dos dados sob o formato XML. Uma crítica ou desvantagem comentada a respeito do XML é a baixa densidade de informações oferecida, causada pela imensa quantidade de tags (marcações) da própria linguagem. Alternativas para otimizar ou aumentar esta densidade é o uso de novos formatos de dados, como o JSON ,que é um formato mais leve para a troca de informações. O JSON é derivado da linguagem JavaScript e é usado para a criação de objetos como estruturas de dados simples e arrays. Apesar de estar relacionado com o JavaScript, o JSON é independente de linguagem com *parsers* (analisadores sintáticos) disponíveis para a maioria das linguagens.

## 3.10 Observação sobre o Upload de Arquivo

Os parâmetros de entrada serão fornecidos através de upload de arquivos. Propomos esta maneira pois assim os dados são enviados de uma forma mais densa e genérica. Os arquivos podem ser tipo texto com protocolos de dados específicos de acordo com a indicação dos autores, arquivos binários, ou mesmo imagens contendo a informação que seria analisada pela tarefa numérica. Dessa maneira, existe uma liberdade maior quanto à entrada dos dados a serem consumidos pela tarefa. E cada vez que o serviço relacionado ao caso de uso "upload de arquivos" é executado, além do *session id* (identificador de sessão), é passado o *task process id* (identificador de instância de tarefa) que identifica a instância da tarefa a ser executada.

## 3.11 Papéis e Responsabilidades

**Um sumário das responsabilidades de cada parte do sistema da NC:**

**O Cliente EUD - *End User Device*** O cliente requisita a tarefa da NC. Deve primeiramente se autenticar no Numerus Cloud. A execução completa de uma tarefa é composta pela execução de alguns serviços relacionados a casos de uso. São eles:

- *Login* (autenticação onde são fornecidos o nome do usuário e a senha (*username* e *password*).
- Consulta das tarefas existentes (com filtros).
- Seleção da tarefa que será chamada.
- Upload de dados de entrada através de arquivos.
- Execução da tarefa.
- Download do resultado.

O cliente recebe um *session id* (identificador de sessão) após o *login* (autenticação) e, com este *session id* (identificador de sessão) ele chama os demais serviços. Após o *login* (autenticação) do cliente, dependendo da política de uso, ele terá um escopo de atuação maior quanto à execução da tarefa numérica. Dependendo do perfil do cliente, a política de uso dará mais ou menos direitos a este cliente. A política de uso é estabelecida pelo autor da tarefa e leva em consideração o tipo do cliente e a tarefa em questão.

**O NCC - *Numerus Cloud Controller*** O NCC ocupa o papel central na arquitetura. É a parte que representa a *cloud computing* (computação em nuvem) essencialmente porque todo processo executado no NC é totalmente transparente para clientes e autores dos programas numéricos. O NC controller tem como responsabilidades:

- Receber os pedidos de registros das tarefas numéricas e de suas propriedades de execução (diretório para upload dos arquivos com dados de entrada, caminho do arquivo executável da tarefa e diretório de saída (a partir do qual será feito o download do resultado).É papel do autor da tarefa o seu registro junto à Numerus Cloud.
- Gerenciar as tarefas registradas dos diversos autores.
- Receber as requisições vindas dos clientes.
- Criar e controlar as sessões dos clientes (usuários logados ou autenticados).

- Criar *task process id* (identificador de instância de tarefa) e chamar as tarefas requisitadas.
- Receber os arquivos enviados pelo cliente e repassando-os para os agentes responsáveis.
- Receber os resultados (saídas) gerados pela execução das tarefas e retorná-los para o cliente.
- Gerenciar os status de upload dos arquivos com dados de entrada, execução da tarefa e download do resultado.
- Executar a política de uso estabelecida pelo autor da tarefa numérica, baseado no cliente e na tarefa.

O NC detém num dado momento, um registro contendo o *session id* (identificador de sessão) que representa o cliente, o *task process id* (identificador de instância de tarefa) que representa a instância da tarefa a ser executada. De posse dessas informações o NC é capaz de localizar o agente responsável pela máquina onde a tarefa numérica será efetivamente executada. Além disso, o NC deve saber quando uma tarefa está completada e retornar o resultado para o cliente. O cliente quando chama o serviço que atende o caso de uso 'chamar tarefa', fica esperando o NC retornar com o resultado.

**O Agente NTE (*Numerical Task Executor*)** É uma aplicação que roda na máquina do autor da tarefa. O agente recebe a requisição de execução da tarefa vinda do NC e a executa na máquina do autor da tarefa. É importante observar que a requisição contém todas as informações necessárias para a execução da tarefa que são:

- O caminho do programa numérico para a execução da tarefa.
- O diretório origem onde se localizam os arquivos que foram enviados do cliente para o NC e do NC para o agente quando da execução do caso de uso 'upload de arquivos'.
- O diretório destino onde o resultado será enviado.

- O modo de execução da tarefa. Uma tarefa poderá ser executada exclusivamente ou não, dependendo de como foi configurada pelo autor.

Os papéis executados pelo agente são:

- Receber as diversas requisições que chegam da NCC.
- Copiar arquivos de upload do cliente para o diretório origem do qual a tarefa fará a leitura dos parâmetros de entrada.
- Executar a tarefa identificada pelo caminho do programa executável informado pelo autor da tarefa quando do registro da tarefa na NCC.
- Controlar as *threads* que executam as diversas instâncias das tarefas do autor.
- Controlar os processos exclusivos e normais.
- Obter o resultado do diretório destino informado pelo autor da tarefa quando do registro da tarefa na NCC.
- Enviar o resultado para o NCC.

## 3.12 Diagramas relacionados aos casos de uso

Baseado na arquitetura apresentada e nos papéis atribuídos aos atores da dinâmica do Numerus Cloud, foi estabelecida uma série de casos de uso que controlam o Numerus Cloud. Serão apresentados os diagramas de sequência relativos aos casos de uso do cliente quando da execução de um serviço no Numerus Cloud.

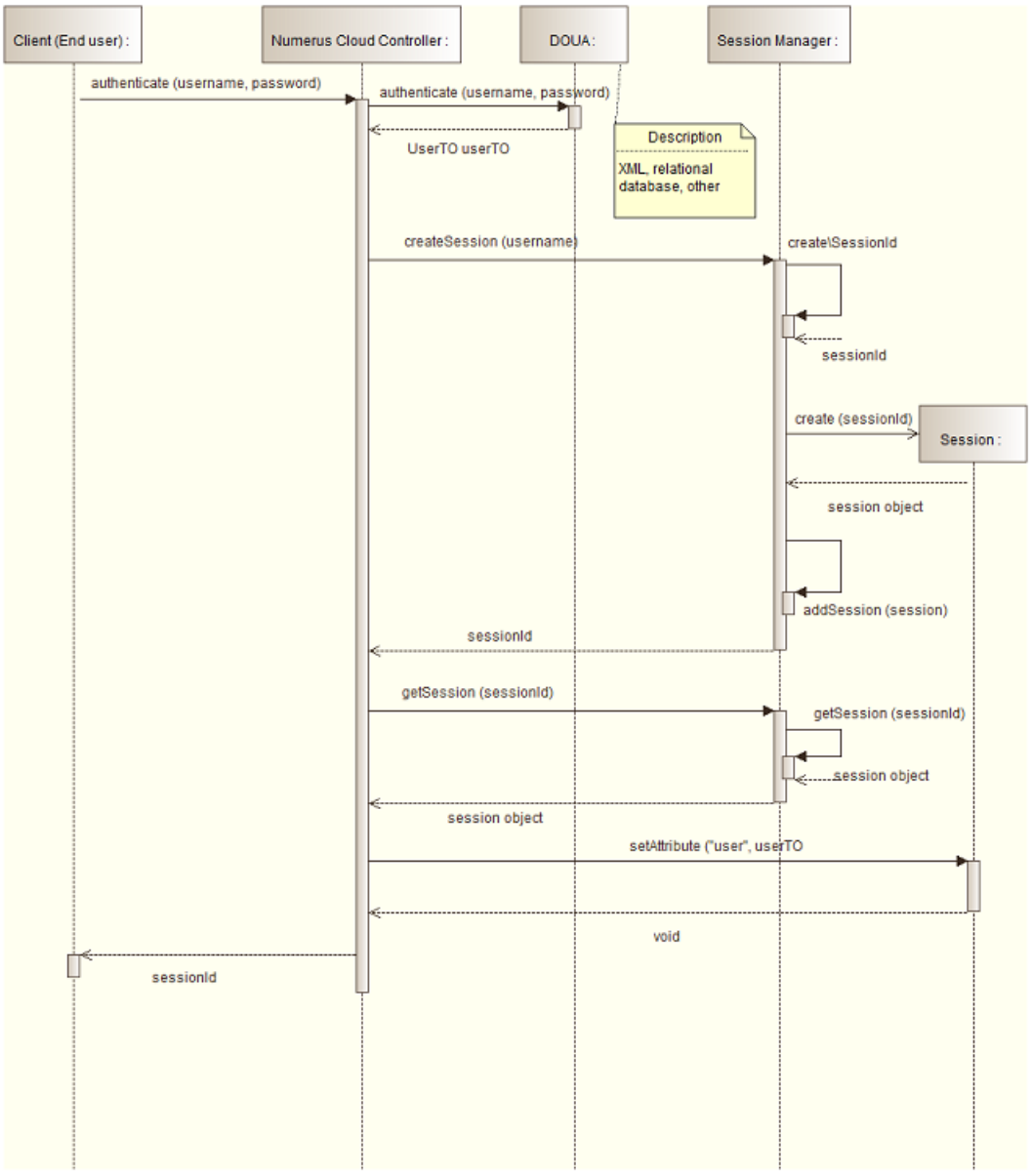


Figura 3.6: *Login* no Numerus Cloud

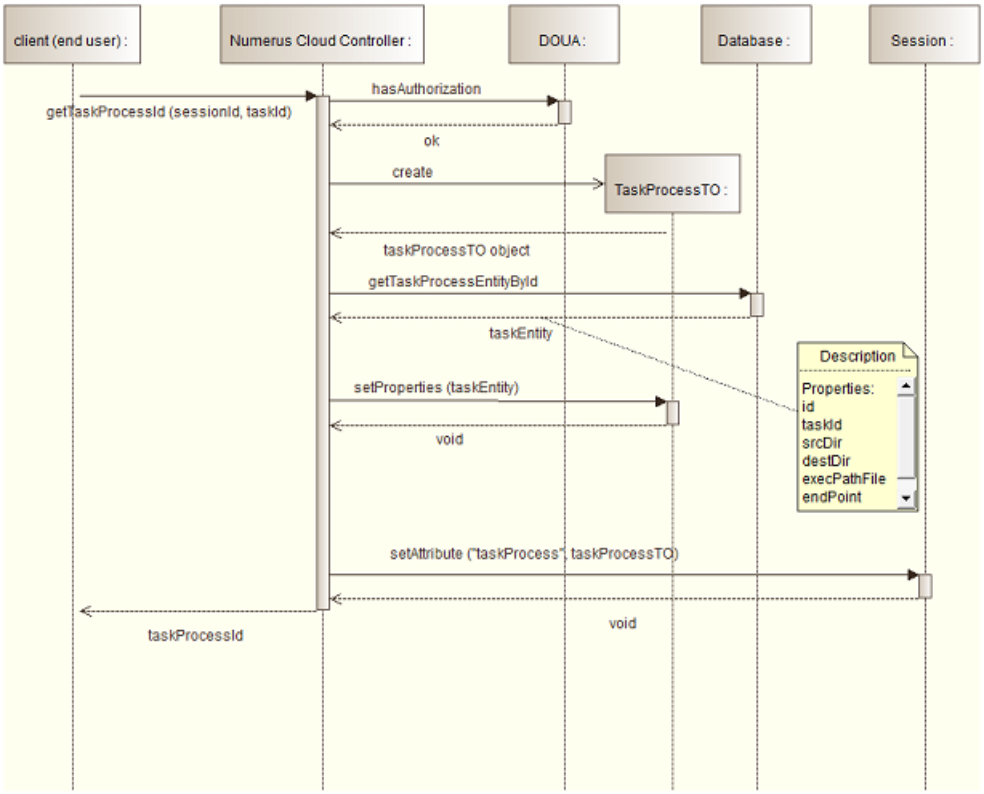


Figura 3.7: Obtenção de um Task Process Id (identificador de instância de tarefa)

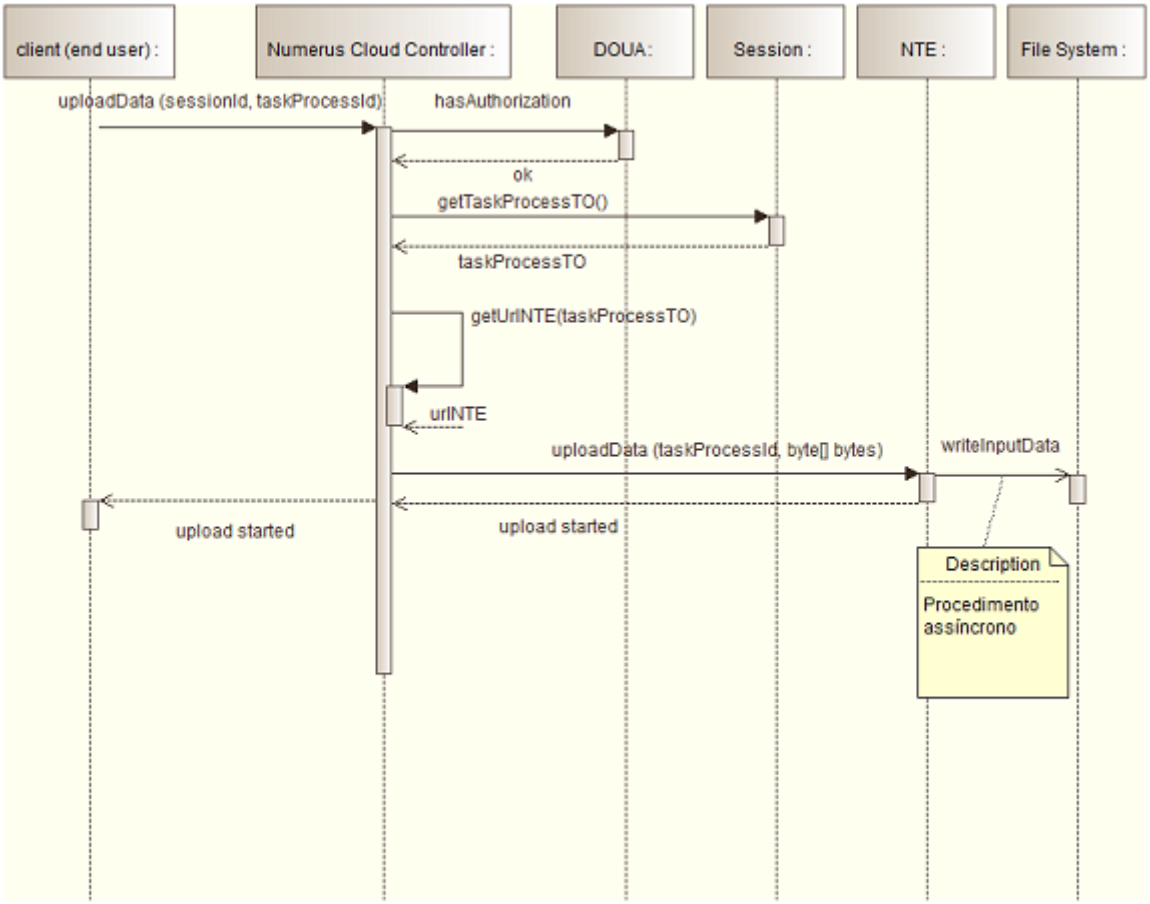


Figura 3.8: Upload - envio dos dados da requisição

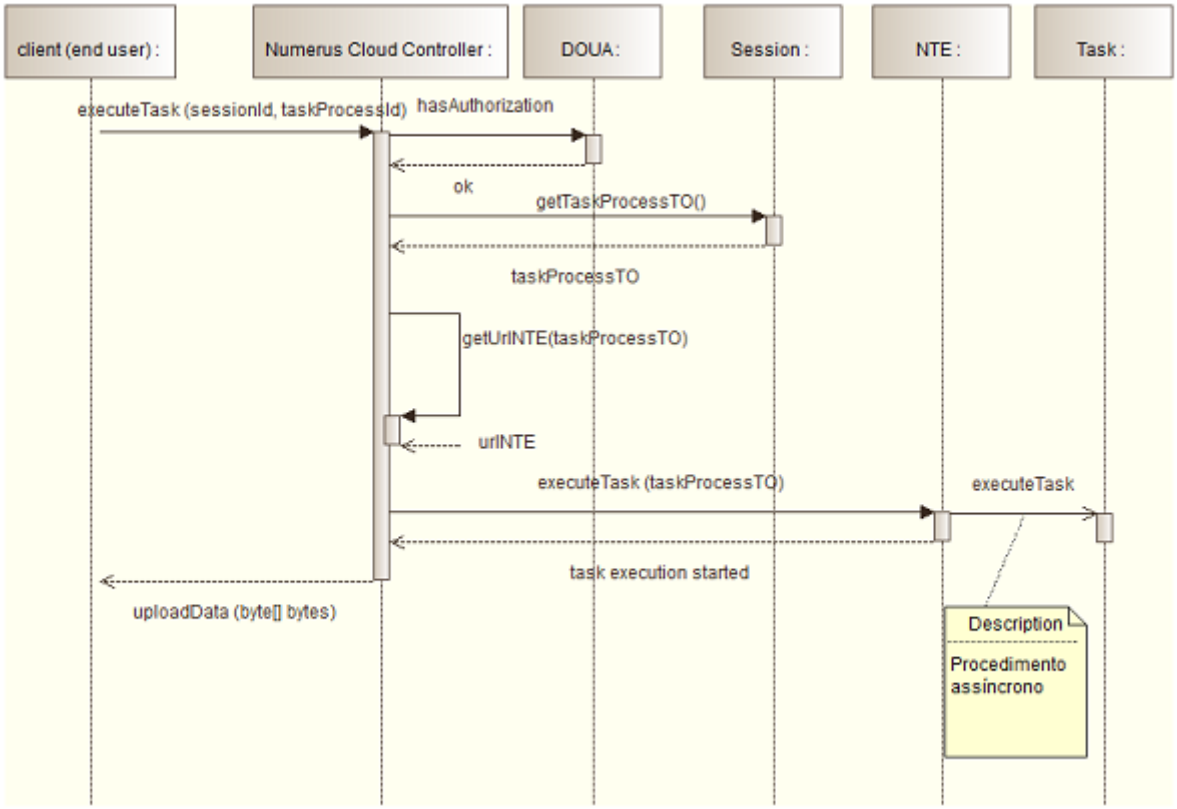


Figura 3.9: Execução da tarefa

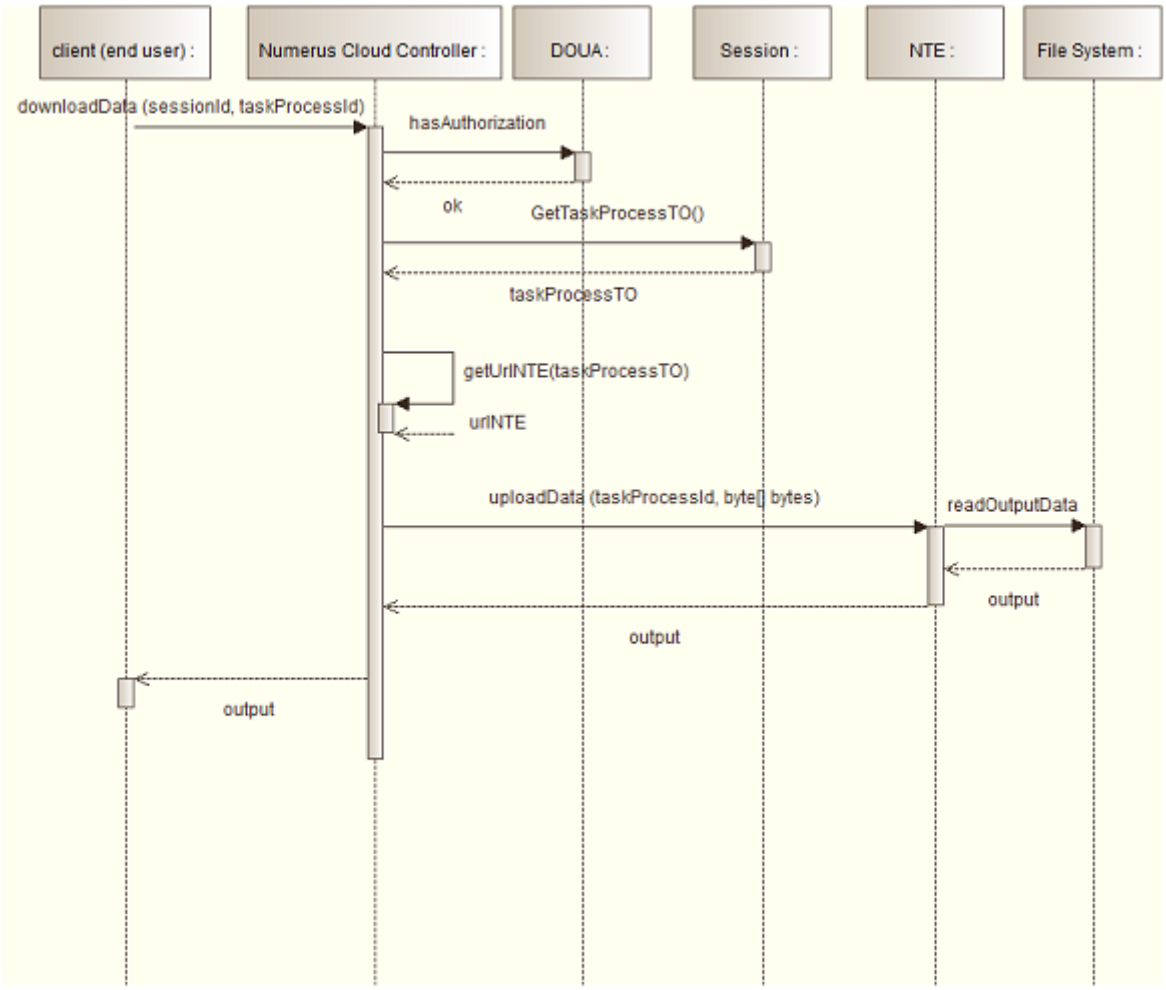


Figura 3.10: Download - recebimento da resposta

# Capítulo 4

# Análise comparativa do Numerus Cloud

## 4.1 Servidores Web, Servidor de Aplicação e Containers Web:

Servidores Web são softwares especializados em prover conteúdo via HTTP. Um exemplo importante de servidor web é o software Apache Server. Os servidores Web têm recursos como cache de imagens e normalmente são extensíveis para poder acoplar algum tipo de mecanismo de geração dinâmica de conteúdo (PHP, Java, Coldfusion, etc). Um servidor de aplicações (*Application Server*), é um servidor que disponibiliza um ambiente para a instalação e execução de certas aplicações, centralizando e dispensando a instalação nos computadores clientes. Os servidores de aplicação também são conhecidos por *middleware*. O objetivo do servidor de aplicações é disponibilizar uma plataforma que separe do desenvolvedor de software algumas das complexidades de um sistema computacional. No desenvolvimento de aplicações comerciais, por exemplo, o foco dos desenvolvedores deve ser a resolução de problemas relacionados ao negócio da empresa, e não de questões de infraestrutura da aplicação. O servidor de aplicações responde a algumas questões comuns a todas as aplicações, como segurança, garantia de disponibilidade, balanceamento de carga e tratamento de exceções. Devido a popularização da plataforma Java, o termo servidor de aplicação é frequentemente uma referência a "servidor de aplicação JEE",

isto é, os servidores de aplicação são softwares cujo funcionamento é regido pela especificação *Java Enterprise Edition* [30]. O servidor WebSphere Application Server da IBM e o WebLogic Server da Oracle são dois dos mais conhecidos servidores JEE comerciais. Alguns servidores de software livre também são muito utilizados, como Glassfish, JBoss, JOnAS, Apache Geronimo e Apache Tomcat([16], [22], [9], [31]). Este último apesar de ser um servidor de aplicações JEE, não é servidor de EJBs (EJB ou *Enterprise JavaBeans* são componentes de negócio distribuídos e transacionais da plataforma JEE que roda em um servidor de aplicação). O Tomcat é um exemplo de *container* (contenedor) de software livre, onde os módulos Web podem ser publicados, e, por essa razão, é dito que o Tomcat é um *container* (contenedor) Web. A linguagem de programação destes softwares é Java. Os módulos Web são em geral implementados através de servlets (classe Java usada para estender as funcionalidades de um servidor) e JSP (*Java Server Pages* é uma tecnologia que ajuda os desenvolvedores de software a criarem páginas Web geradas dinamicamente) e a lógica de negócio através de EJBs. *Containers* (contenedores) Web não são servidores Web e vice-versa. Um servidor Web pode delegar o processamento dinâmico a um *container* (contenedor) Web (como é o caso do Apache Server que pode delegar ao Tomcat - [16] ).

## 4.2 Deployment (implantação)

As técnicas padronizadas de *deployment* (implantação ou disponibilização de uma aplicação em um servidor de aplicações) dos serviços relacionados aos Web Services geralmente envolvem uma compilação dos arquivos fontes, a geração de pacotes para *deployment* (implantação)(por exemplo, na especificação JEE, os pacotes para *deployment* (implantação) de uma aplicação são representados por aquivos compactados no formato *zip* contendo as classes a serem publicadas - *ZIP* é um formato de compactação de arquivos muito difundido pela Internet), o *upload* (envio) desses arquivos (pacotes) e a descompactação desses arquivos nos diretórios específicos de cada servidor de aplicação. O servidor estabelece o ambiente onde as aplicações são publicadas ( [3], [37]). Os Web Services são aplicações que residem nos servidores de aplicação ou nos servidores Web. Atualmente , existem ferramentas ou aplicativos

específicos para facilitar o *deployment* (implantação) em tempo de desenvolvimento e testes, como as ferramentas de integração contínua (Jenkins ou o Hudson). Porém, essas técnicas ainda exigem um nível maior de conhecimento de TI, gerando uma dependência e um obstáculo para a publicação de novas versões do software de programação numérica. Vale observar que o que se deseja é a independência por parte do autor da tarefa numérica das tarefas administrativas de TI ligadas às configurações das aplicações de suporte e ao *deployment* (implantação) dos novos serviços. O Numerus Cloud usa também o Web Services para o envio das requisições e recebimento das respostas dos serviços disponibilizados. Porém, os serviços são sempre os mesmos e são usados apenas como um canal por onde passam as requisições e as respostas a essas requisições. Não há mistura da lógica ou *core business* do autor da tarefa numérica com a parte administrativa dos Web Services. O Numerus Cloud desconhece completamente o que trafega pela nuvem. O *deployment* (implantação) de uma nova versão do programa numérico é feito simplesmente com a inclusão de um novo arquivo executável na máquina controlada pelo autor do programa. A proposta de uma independência com relação a softwares que não tem relação com o programa numérico é conseguida com o processo de *deployment* (implantação) do Numerus Cloud, diferentemente do processo padrão de *deployment* (implantação). A proposta de proteção do software do autor da tarefa numérica também é atingida pelo Numerus Cloud graças a independência com relação ao funcionamento da tarefa. O Numerus Cloud não sabe absolutamente nada sobre o conteúdo da tarefa, estabelecendo um funcionamento para toda e qualquer tarefa numérica.

## 4.3 Segurança

O usuário pode ficar relutante na criação de sistemas onde os dados residem fora de seu controle. Muitas empresas, devido à governança, risco e regulamentos, possuem regras rígidas sobre a manipulação de dados estratégicos e confidenciais. As maiores preocupações quanto à segurança são:

Abuso do uso da *cloud computing* (computação em nuvem).

Perda ou vazamento de dados.

Sequestro de contas, serviços ou tráfego.

Uma primeira abordagem para minimizar estas questões é o uso de nuvens privadas. As nuvens privadas guardam toda a infraestrutura localmente, sob a proteção do *firewall* da empresa ou instituição e sob o controle direto do grupo de TI. Neste caso, a máquina onde roda o Numerus Cloud fica na rede que contém a máquina controlada pelo autor do programa numérico. O funcionamento do Numerus Cloud permanece o mesmo.

## 4.4 Conexão e performance

Como na questão de segurança, uma possível solução para problemas ligados à conexão e performance é o uso de nuvens híbridas, isto é, conciliar a nuvem particular e a nuvem pública. Mas o conceito é o mesmo. Neste caso, a própria nuvem é controlada internamente à empresa ou instituição. Nenhum pacote sai da rede, e, existe uma máquina onde reside o Numerus Cloud. O funcionamento é o mesmo do Numerus Cloud numa nuvem pública. Com esta configuração, o *upload* (envio) dos dados que serão consumidos pela tarefa numérica é bem mais rápido. Paralelamente, a segurança dos dados de requisição é mantida, pois não há tráfego fora da rede. Com relação à performance , algoritmos de compressão e posterior descompressão de binários podem ser usados para tornar o *upload* (envio de dados) e o *download* (baixa do resultado)(dois serviços básicos do Numerus Cloud) mais rápidos diminuindo a quantidade de bytes das informações trafegadas. Compressão de números do tipo float é uma técnica que pode ser aplicada.

## 4.5 Grid Computing e Cloud Computing

*Grid computing* (computação em grade), é um modelo computacional capaz de alcançar uma alta taxa de processamento dividindo as tarefas entre diversas máquinas, podendo ser em rede local ou rede de longa distância, que formam uma máquina virtual. Esses processos podem ser executados no momento em que as máquinas não estão sendo utilizadas pelo usuário, assim evitando o desperdício de processamento da máquina utilizada. Mesmo reconhecendo que o uso bastante difundido das tecnologias de Grid na computação cientifica é comprovado pelo imenso número de projetos existentes, algumas questões ainda fazem com que o acesso a esta tec-

nologia não seja tão fácil quanto é retratado. Algumas questões são burocráticas: sendo o Grid baseado no compartilhamento e no caso de serem públicos, no compartilhamento de máquinas pela Web, os grupos de pesquisa que pretendem usar o Grid devem submeter uma proposta descrevendo o tipo de pesquisa a ser realizado. Este procedimento leva a uma espécie de competição no uso de grids científicas e projetos científicos de menor expressão não conseguem acesso às grids. Outras questões são técnicas e mais importantes: na maioria dos casos de grids científicas há a necessidade de ambientes preestabelecidos e pacotes prontos onde as aplicações da grid serão executadas, e algumas vezes, ferramentas específicas e APIs tem de ser usadas podendo haver limitações no próprio sistema operacional de quem hospedará tais serviços (nas máquinas individuais que compõem a grid). Exemplos práticos envolvem o uso de softwares específicos que podem não estar disponíveis no ambiente de *runtime* (tempo de execução) onde será executada a aplicação. Portanto, situações em que a burocracia é um problema menor, o problema técnico pode constituir um obstáculo fundamental para a computação cientifica no caso do uso de grids. Procedimentos baseados em técnicas de virtualização, neste sentido, é a alternativa. A *cloud computing* (computação em nuvem) , que é a técnica mais emergente de fornecimento de serviços de TI , pode resolver os problemas citados. Através das técnicas de virtualização, a *cloud computing* (computação em nuvem) oferece aos usuários finais uma variedade de serviços que abrangem todas as camadas da computação, desde o hardware até o nível de aplicação, estabelecendo uma cobrança do tipo *pay-per-use* (pagamento por uso). Outra característica importante da qual os cientistas podem se beneficiar é a capacidade de dimensionar para mais ou para menos a infraestrutura usada de acordo com os requisitos da aplicação e orçamento dos usuários. Através do aluguel da infraestrutura, os cientistas acessam imediatamente os recursos requeridos sem nenhuma necessidade de planejamento e podem liberar esses mesmos recursos quando não mais necessário. A conclusão é que a *cloud computing* (computação em nuvem) fornece um mecanismo flexível de entrega de serviços de TI nos níveis que vão do hardware até o software e que faz com que o leque de opções disponíveis para os cientistas seja grande o suficiente para que cubra todas as necessidades específicas de suas pesquisas.

## 4.6 Web Services e o RMI

Existem algumas tecnologias para conectar sistemas remotamente, dentre elas o Web Services que é a tecnologia usada pelo Numerus Cloud. Uma alternativa ao Web Services é o RMI (*Remote Method Invocation* - Invocação de Método Remoto). O RMI é uma interface de programação que permite a execução de chamadas remotas no estilo RPC em aplicações desenvolvidas em Java. Através da utilização da arquitetura RMI, é possível que um objeto ativo em uma JVM (*Java Virtual Machine* - Máquina Virtual Java) possa interagir com objetos de outras JVMs, independentemente da localização dessas máquinas virtuais. O RMI é uma solução que resolve o problema de acesso remoto quando o cliente e o servidor são programas feitos em Java. O Web Services é independente de linguagem de programação. Uma vez criado o Web Services, basta o cliente acessar o arquivo WSDL que descreve os serviços, parâmetros e respostas aos serviços dos Web Services. Dessa maneira, é possível a criação de um programa cliente escrito em qualquer linguagem, para acessar o Web Services. Observando a arquitetura do Numerus Cloud, existe o acesso remoto entre o Numerus Cloud Controller e o Numerus Task Executor residente na máquina do autor da tarefa numérica. Sendo ambos programas escritos em Java, o RMI poderia ter sido utilizado. Porém, a complexidade seria maior e haveria a necessidade de manter as JVMs do cliente e do servidor em sincronismo. Não é possível uma atualização (*upgrade*) no cliente sem ter uma atualização (*upgrade*) no servidor. Ao invés de criar um servidor RMI, foi criado um servlet (que roda no Tomcat) compondo uma solução simples e fina no sentido de consumo de recursos.

# Capítulo 5

# Resultado

Para mostrar o funcionamento da proposta do Numerus Cloud, foram implementados programas relativos às camadas existentes na arquitetura. As camadas NC-Controller e NTE são compostas por duas aplicações e executadas num servidor de aplicações. Cada uma dessas aplicações, por sua vez, é composta por classes que foram desenvolvidas dentro do paradigma *Design Patterns* (padrões de projeto - [42]) . Um *Design Pattern*, descreve uma solução geral reutilizável para um problema recorrente no desenvolvimento de sistemas de software orientados a objetos. Não é um código final, é uma descrição ou modelo de como resolver o problema do qual trata, que pode ser usada em muitas situações diferentes. Os *Design Patterns* normalmente definem as relações e interações entre as classes ou objetos, sem especificar os detalhes das classes ou objetos envolvidos, ou seja, estão num nível de generalidade mais alto ( [35]).

**Principais Design Patterns utilizados ( [17], [14]): Abstract Factory, Façade, Singleton e DAO:** O DAO (*Data Access Object* - Objeto de Acesso a Dados) é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados. Numa aplicação que utilize a arquitetura MVC (*Model View Controller*), todas as funcionalidades de bancos de dados, tais como obter as conexões, mapear objetos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes de DAO.

O *Façade* (termo em francês que significa fachada em portugûes ou *facade* em

inglês) é um objeto que fornece uma interface simplificada para um conjunto maior de código, como uma biblioteca de classes.

*Abstract Factory* permite a criação de famílias de objetos relacionados ou dependentes, através de uma única interface e sem que a classe concreta seja especificada.

Foram desenvolvidos dois programas clientes para plataformas diferentes. Um programa é um aplicativo do tipo *desktop* e pode ser executado em qualquer máquina *desktop* e o outro programa roda num dispositivo móvel (*smartphone* ou *tablet*) com sistema operacional Android.

**O programa numérico:** Como programa numérico, foi usado o método de clusterização XCM (*Xavier Clustering Method* [44]).O programa numérico está numa máquina remota e isolada, e é acionado a partir de uma nuvem pública. O acesso a esta máquina é totalmente restrito. Apenas o arquivo executável é colocado em algum diretório da máquina e que será chamado pelo Numerus Cloud. Não há nenhuma referência em termos de código sobre o conteúdo, sobre o que faz ou como se comporta o programa executável. Não há uma linha de código relativo às regras de negócio do programa numérico nos arquivos fontes do Numerus Cloud Controller, do NTE e dos programas clientes. Portanto, é caracterizado o isolamento total entre o programa numérico e as camadas do Numerus Cloud.

## 5.1 Implementação

Para o desenvolvimento dos programas do NC Controller, NTE e os programas clientes foi usado a linguagem Java (Oracle), na versão 1.7 com a ferramenta IDE NetBeans(Oracle)(IDE, do inglês *Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo).

O programa cliente que é executado na máquina *desktop* foi feito em Java Swing (uma API para desenvolvimento de interfaces gráficas GUI - *Graphical User Interface* - para programas em Java).

O programa cliente para dispositivos móveis foi feito usando a linguagem Java em cima do sistema operacional Android.

As camadas NC Controller e NTE são duas aplicações que rodam no *container* (contenedor) Web Tomcat. A versão do Tomcat é a 7.0. Essas aplicações são do tipo JEE.(*Java Platform, Enterprise Edition* ou Java EE - em português Plataforma Java, Edição Empresarial - é uma plataforma de programação para servidores na linguagem de programação Java. A plataforma fornece uma API e um ambiente de tempo de execução para o desenvolvimento e execução de softwares corporativos, incluindo serviços de rede e Web, e outras aplicações de rede de larga escala, multicamadas, escaláveis, confiáveis e seguras).

No exemplo, foi usado usada uma nuvem pública, em máquinas virtuais de total controle e restrição. O sistema operacional onde rodam, o Tomcat, o NC Controller e o NTE é o Linux Ubuntu.

### 5.1.1 NC Controller

O NC Controller contém várias classes, e dentre elas, existe a classe *entry point* (ponto de entrada) "NumerusCloudWS" contendo os métodos da interface Web Services que expõe os serviços ao mundo exterior. O Web Services do NC foi feito em cima da tecnologia Axis2 da Apache. O Apache Axis2 é um *framework* de código aberto, baseado na linguagem Java e no padrão XML, utilizado para construção de Web Services no padrão SOAP. Através do Axis os desenvolvedores podem criar aplicações distribuídas.

Abaixo, segue um trecho da classe *entry point* do Numerus Cloud Controller.

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package numeruscloud;
6
7  import java.util.ArrayList;
8  import java.util.Date;
9  import java.util.List;
10 import numeruscloud.task.TaskProcessTO;
11
12 /**
13  *
14  * @author Joao Carlos
15  */
16 public class NumerusCloudWS
17 {
18     private static SessionManager sessionManager;
```

```
19
20      private SessionManager getSessionManager ()
21      {
22          if (sessionManager == null)
23          {
24              sessionManager = SessionManager.getInstance();
25          }
26          return sessionManager;
27      }
28
29      private Session getSession (String sessionId)
30      {
31          return getSessionManager().getSession (sessionId);
32      }
33
34      private synchronized static String createSessionId (String
            username,
35      String password)
36      {
37          return System.currentTimeMillis() + username + password;
38      }
39
40      private TaskProcessTO getProcessTOById (String taskProcessId,
41      List<TaskProcessTO> processes)
42      {
43          TaskProcessTO processTOFound = null;
44          for (TaskProcessTO processTO : processes)
45          {
46              if (processTO.getId().equals(taskProcessId))
47              {
48                  processTOFound = processTO;
49                  break;
50              }
51          }
52          return processTOFound;
53      }
54
55      public String authenticate (String username, String password)
56      {
57          NumerusCloudFacadeBean numerusCloudBean =
58  new NumerusCloudFacadeBean ();
59  UserTO userTO = numerusCloudBean.authenticate (username, password)
        ;
60          String sessionId = null;
61          if (userTO != null)
62          {
63              sessionId = createSessionId (username, password);
64              this.getSessionManager().createSession (sessionId);
65          }
66          return sessionId;
67      }
68
69      public String createTaskProcessId (String sessionId, String
            taskId)
70      {
71          NumerusCloudFacadeBean numerusCloudBean = new
                NumerusCloudFacadeBean ();
72          TaskProcessTO taskProcessTO = numerusCloudBean.
                createTaskProcessTO (taskId);
73
74          // save taskProcessTO into session:
75          Session session = this.getSession (sessionId);
76          List<TaskProcessTO> myprocesses = null;
```

```
77          if (!session.existAttribute ("myprocesses"))
78          {
79              myprocesses = new ArrayList<TaskProcessTO>();
80              session.setAttribute ("myprocesses", myprocesses);
81          }
82          else
83          {
84              myprocesses = (List<TaskProcessTO>)this.getSession
85        (sessionId).getAttribute ("myprocesses");
86          }
87          myprocesses.add (taskProcessTO);
88
89          return taskProcessTO.getId();
90      }
91
92      public void uploadDataNamed (String sessionId,
93      String taskProcessId, String name, byte[] data)
94      {
95          System.out.println ("sessionId " + sessionId);
96          System.out.println ("taskProcessId " + taskProcessId);
97          System.out.println ("name " + name);
98          System.out.println ("data " + data);
99
100         // find the process:
101         Session session = this.getSession (sessionId);
102         System.out.println ("session " + session);
103         List<TaskProcessTO> processes =
104   (List<TaskProcessTO>)session.getAttribute ("myprocesses");
105         System.out.println ("processes " + processes);
106         TaskProcessTO processTO =
107   this.getProcessTOById (taskProcessId, processes);
108         System.out.println ("processTO " + processTO);
109
110         // get the srcDir from the process object:
111         NumerusCloudFacadeBean numerusBean
112   = new NumerusCloudFacadeBean ();
113         numerusBean.uploadDataNamed (processTO, name, data);
114     }
115
116     public void executeTaskDataNamed (String sessionId,
117     String taskProcessId, String dataName)
118     {
119         // find the process:
120         Session session = this.getSession (sessionId);
121         List<TaskProcessTO> processes
122   = (List<TaskProcessTO>)session.getAttribute ("myprocesses");
123         TaskProcessTO processTO
124   = this.getProcessTOById (taskProcessId, processes);
125
126         // get the srcDir from the process object:
127         NumerusCloudFacadeBean numerusBean
128   = new NumerusCloudFacadeBean ();
129         numerusBean.executeTaskProcessDataNamed (processTO, dataName
              );
130     }
131
132     public byte[] downloadDataNamed (String sessionId,
133     String taskProcessId, String dataName)
134     {
135         // find the process:
136         Session session = this.getSession (sessionId);
137         List<TaskProcessTO> processes = (List<TaskProcessTO>)
138   session.getAttribute ("myprocesses");
```

```
139         TaskProcessTO processTO
140   = this.getProcessTOById (taskProcessId, processes);
141
142         NumerusCloudFacadeBean numerusBean
143   = new NumerusCloudFacadeBean ();
144         return numerusBean.downloadDataNamed (processTO, dataName);
145     }
146
147 }
```

---

**Comunicação Numerus Cloud Controller com o NTE:** O NTE possui como *entry point* uma classe servlet Java. Um servlet é um "programa" Java utilizado para estender as funcionalidades da camada servidora, através do modelo de programação solicitação e resposta (*request/response*). O seu uso mais comum é na criação de conteúdo Web dinamicamente, permitindo assim a extensão de funcionalidades dos servidores Web. Para isso, a Java Servlet API, que é o protocolo pelo qual a classe Java pode responder às requisições que chegam ao servidor, define um conjunto de classes específicas para o protocolo HTTP.

O NC Controller se comunica com a camada NTE através do envio de requisições para a servlet do NTE. Segue abaixo a classe do NC Controller responsável pelo envio de requisições e recebimento das respostas do NTE:

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package numeruscloud;
6
7  import java.io.IOException;
8  import java.io.InputStream;
9  import java.io.ObjectInputStream;
10 import java.io.ObjectOutputStream;
11 import java.io.OutputStream;
12 import java.net.HttpURLConnection;
13 import java.net.MalformedURLException;
14 import java.net.URL;
15 import java.net.URLConnection;
16 import java.util.ArrayList;
17 import java.util.HashMap;
18 import java.util.List;
19 import java.util.Map;
20 import java.util.logging.Level;
21 import java.util.logging.Logger;
22 import numeruscloud.task.TaskProcessTO;
23 import org.json.simple.JSONObject;
24
25 /**
26  *
27  * @author Joao Carlos
28  */
29 public class NumerusAgentDAOImpl implements NumerusAgentDAO
30 {
```

```
31      private static Logger log = Logger.getLogger (
            NumerusAgentDAOImpl.class.getName());
32
33      private NumerusAgentDAOImpl (){}
34      public static NumerusAgentDAOImpl create ()
35      {
36          return new NumerusAgentDAOImpl ();
37      }
38
39      public JSONObject sendDataToNumericAgent (String endpoint,
            Object data)
40      {
41          ObjectOutputStream oos = null;
42          OutputStream out = null;
43          try
44          {
45              JSONObject json = (JSONObject)  data;
46              String wsURL = endpoint;
47              URL url = new URL(wsURL);
48              URLConnection connection = url.openConnection();
49              HttpURLConnection httpConn = (HttpURLConnection)
                    connection;
50              httpConn.setDoInput(true);
51              httpConn.setDoOutput(true);
52              httpConn.setRequestMethod("POST"); // ???
53              out = httpConn.getOutputStream();
54              oos = new  ObjectOutputStream (out);
55              oos.writeObject(json);
56              oos.flush();
57              InputStream is = httpConn.getInputStream();
58              ObjectInputStream ois = new  ObjectInputStream (is);
59              Object obj = ois.readObject();
60              return (JSONObject)obj;
61          }
62          catch (MalformedURLException ex)
63          {
64              ex.printStackTrace();
65              log.log (Level.SEVERE, null, ex);
66              throw new RuntimeException (ex);
67          }
68          catch (IOException ex)
69          {
70              ex.printStackTrace();
71              log.log (Level.SEVERE, null, ex);
72              throw new RuntimeException (ex);
73          }
74          catch (Exception ex)
75          {
76              ex.printStackTrace();
77              log.log (Level.SEVERE, null, ex);
78              throw new RuntimeException (ex);
79          }
80          finally
81          {
82              try
83              {
84                  oos.close();
85                  out.close();
86              }
87              catch (IOException ex)
88              {
89                  log.log (Level.SEVERE, null, ex);
90                  throw new RuntimeException (ex);
```

```
91              }
92          }
93      }
94
95      @Override
96      public void uploadDataNamed
97      (TaskProcessTO processTO, String dataName, byte[] data)
98      {
99          log.info ("DAO layer : >>> uploadDataNamed");
100         try
101         {
102             String taskProcessId = processTO.getId();
103             String srcDir = processTO.getSrcDir();
104             String endpoint = processTO.getEndpoint();
105             // create the input JSON object:
106             JSONObject jsonIn = new JSONObject();
107             jsonIn.put ("ncservice", "uploadDataNamed");
108
109             JSONObject jsonInData = new JSONObject();
110             jsonIn.put ("data", jsonInData);
111             jsonInData.put ("taskProcessId", taskProcessId);
112             jsonInData.put ("srcDir", srcDir);
113             jsonInData.put ("dataIn", data);
114             jsonInData.put ("dataName", dataName);
115
116             JSONObject jsonOut =
117       this.sendDataToNumericAgent (endpoint, jsonIn);
118
119             log.info ("jsonOut " + jsonOut);
120         }
121         catch (RuntimeException ex)
122         {
123             ex.printStackTrace();
124             Logger.getLogger (NumerusAgentDAOImpl.class.getName()).
125       log(Level.SEVERE, null, ex);
126             throw ex;
127         }
128     }
129
130     @Override
131     public void executeTaskProcessDataNamed
132     (TaskProcessTO processTO, String dataName)
133     {
134         log.info ("DAO layer : >>> executeTaskProcessDataNamed");
135         try
136         {
137             String taskProcessId = processTO.getId();
138             String execPathFilename = processTO.getExecPathFile();
139             String srcDir = processTO.getSrcDir();
140             String destDir = processTO.getDestDir();
141             String endpoint = processTO.getEndpoint();
142             // create the input JSON object:
143             JSONObject jsonIn = new JSONObject();
144             jsonIn.put ("ncservice", "executeTaskProcessDataNamed");
145
146             JSONObject jsonInData = new JSONObject();
147             jsonInData.put("taskProcessId", taskProcessId);
148             jsonInData.put ("execPathFilename", execPathFilename);
149             jsonInData.put ("srcDir", srcDir);
150             jsonInData.put ("destDir", destDir);
151             jsonInData.put ("dataName", dataName);
152
153             jsonIn.put ("data", jsonInData);
```

```
154
155             JSONObject jsonOut =
156       this.sendDataToNumericAgent (endpoint, jsonIn);
157
158             log.info ("jsonOut " + jsonOut);
159         }
160         catch (RuntimeException ex)
161         {
162             ex.printStackTrace();
163             Logger.getLogger (NumerusAgentDAOImpl.class.getName()).
164       log(Level.SEVERE, null, ex);
165             throw ex;
166         }
167     }
168
169     @Override
170     public byte[] downloadDataNamed
171     (TaskProcessTO processTO, String dataName)
172     {
173         log.info ("DAO layer : >>> downloadData");
174
175         String taskProcessId = processTO.getId();
176         String destDir = processTO.getDestDir();
177         String endpoint = processTO.getEndpoint();
178         // call byte[] NumerusAgentWS.downloadData ();
179
180         // create the JSON object:
181         JSONObject jsonIn = new JSONObject();
182         jsonIn.put ("ncservice", "downloadDataNamed");
183
184         JSONObject jsonInData = new JSONObject();
185         jsonIn.put ("data", jsonInData);
186         jsonInData.put ("taskProcessId", taskProcessId);
187         jsonInData.put ("destDir", destDir);
188         jsonInData.put ("dataName", dataName);
189
190         JSONObject jsonOut =
191   this.sendDataToNumericAgent (endpoint, jsonIn);
192         log.info ("jsonOut " + jsonOut);
193
194         Object obj = jsonOut.get ("data");
195         if (obj.getClass().equals (byte[].class))
196         {
197             byte[] data = (byte []) obj;
198             return data;
199         }
200         else //(Exception.class.isAssignableFrom (obj.getClass()))
201         {
202             Exception ex = (Exception)obj;
203             throw new RuntimeException (ex);
204         }
205     }
206
207 }
```

### 5.1.2 NTE

Como já mencionado, o NTE é um conjunto de classes localizadas na máquina controlada pelo autor do programa numérico e que tem como um *entry point* um

servlet. Este servlet recebe as requisições do NC Controller e se comunica com o *file system* (sistema de arquivos) da máquina. O servlet é mostrado a seguir com o método chamado pelo servidor para processamento das requisições:

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package na;
6
7
8  import java.io.IOException;
9  import java.io.InputStream;
10 import java.io.ObjectInputStream;
11 import java.io.ObjectOutputStream;
12 import java.io.OutputStream;
13 import java.net.HttpURLConnection;
14 import java.net.URL;
15 import java.net.URLConnection;
16 import java.util.logging.Logger;
17 import javax.servlet.ServletException;
18 import javax.servlet.annotation.WebServlet;
19 import javax.servlet.http.HttpServlet;
20 import javax.servlet.http.HttpServletRequest;
21 import javax.servlet.http.HttpServletResponse;
22 import org.json.simple.JSONObject;
23
24 /**
25  *
26  * @author Joao Carlos
27  */
28 @WebServlet(name = "NAServlet", urlPatterns = {"/NAServlet"})
29 public class NumerusAgentServlet extends HttpServlet
30 {
31     private static Logger log
32     = Logger.getLogger (NumerusAgentServlet.class.getName());
33
34     private JSONObject uploadDataNamed (JSONObject jsonIn)
35     {
36         JSONObject jsonDataIn = (JSONObject) jsonIn.get ("data");
37
38         String taskProcessId = (String) jsonDataIn.get ("
               taskProcessId");
39         String srcDir = (String) jsonDataIn.get ("srcDir");
40         byte[] data = (byte[]) jsonDataIn.get ("dataIn");
41         String dataName = (String) jsonDataIn.get ("dataName");
42
43         NumerusAgentFacadeBean naFacadeBean =
44   new NumerusAgentFacadeBean ();
45         naFacadeBean.uploadDataNamed
46   (taskProcessId, srcDir, dataName, data);
47
48         JSONObject jsonOut = new JSONObject ();
49         jsonOut.put ("response", "ok uploadData");
50         return jsonOut;
51     }
52
53     private JSONObject executeTaskProcessDataNamed (JSONObject
           jsonIn)
54     {
55         JSONObject jsonDataIn = (JSONObject) jsonIn.get ("data");
56
```

```
57          String taskProcessId = (String) jsonDataIn.get ("
                taskProcessId");
58          String execPathFilename =
59    (String) jsonDataIn.get ("execPathFilename");
60          String srcDir = (String) jsonDataIn.get ("srcDir");
61          String destDir = (String) jsonDataIn.get ("destDir");
62          String dataName = (String) jsonDataIn.get ("dataName");
63
64          log.info ("executeTaskProcessDataNamed");
65          NumerusAgentFacadeBean naFacadeBean = new
                NumerusAgentFacadeBean ();
66          naFacadeBean.executeTaskDataNamed
67    (taskProcessId, execPathFilename, srcDir, destDir, dataName);
68
69          JSONObject jsonOut = new JSONObject ();
70          jsonOut.put ("response", "ok executeTaskProcess");
71          return jsonOut;
72      }
73
74      private JSONObject downloadDataNamed
75      (JSONObject jsonIn)
76      {
77          JSONObject jsonDataIn =
78    (JSONObject) jsonIn.get ("data");
79          String taskProcessId =
80    (String) jsonDataIn.get ("taskProcessId");
81          String destDir = (String) jsonDataIn.get ("destDir");
82          String dataName = (String) jsonDataIn.get ("dataName");
83
84          NumerusAgentFacadeBean naFacadeBean
85    = new NumerusAgentFacadeBean ();
86          byte[] dataOut = naFacadeBean.downloadDataNamed
87    (taskProcessId, destDir, dataName);
88
89          JSONObject jsonOut = new JSONObject ();
90          jsonOut.put ("response", "ok downloadData");
91          jsonOut.put ("data", dataOut);
92          return jsonOut;
93      }
94
95      /**
96       * Processes requests for both HTTP <code>GET</code> and <code>
              POST</code> methods.
97       * @param request servlet request
98       * @param response servlet response
99       * @throws ServletException if a servlet-specific error occurs
100      * @throws IOException if an I/O error occurs
101      */
102     protected void processRequest (HttpServletRequest request,
103     HttpServletResponse response) throws ServletException,
            IOException
104     {
105         InputStream is = request.getInputStream();
106         ObjectInputStream ois = new ObjectInputStream (is);
107         OutputStream out = response.getOutputStream();
108         ObjectOutputStream oos = new ObjectOutputStream (out);
109         try
110         {
111             Object obj = ois.readObject();
112             log.info ("obj read : >>> " + obj);
113
114             // get the service:
115             JSONObject jsonIn = (JSONObject) obj;
```

```
116             String serviceName = (String) jsonIn.get ("ncservice");
117             log.info ("serviceName : >>> " + serviceName);
118 
119             JSONObject jsonOut = null;
120 
121             if (serviceName.equals ("uploadData"))
122             {
123                 jsonOut = this.uploadData (jsonIn);
124             }
125             else if (serviceName.equals ("uploadDataNamed"))
126             {
127                 jsonOut = this.uploadDataNamed (jsonIn);
128             }
129             else if (serviceName.equals ("executeTaskProcess"))
130             {
131                 jsonOut = this.executeTaskProcess (jsonIn);
132             }
133             else if (serviceName.equals ("
                    executeTaskProcessDataNamed"))
134             {
135                 jsonOut = this.executeTaskProcessDataNamed (jsonIn);
136             }
137             else if (serviceName.equals ("downloadData"))
138             {
139                 jsonOut = this.downloadData (jsonIn);
140             }
141             else if (serviceName.equals ("downloadDataNamed"))
142             {
143                 jsonOut = this.downloadDataNamed (jsonIn);
144             }
145 
146             oos.writeObject (jsonOut);
147             oos.flush();
148         }
149         catch (Exception ex)
150         {
151             ex.printStackTrace();
152         }
153         finally
154         {
155             oos.close();
156         }
157     }
158 
159     /**
160      * Handles the HTTP <code>GET</code> method.
161      * @param request servlet request
162      * @param response servlet response
163      * @throws ServletException if a servlet-specific error occurs
164      * @throws IOException if an I/O error occurs
165      */
166     @Override
167     protected void doGet
168     (HttpServletRequest request, HttpServletResponse response)
169     throws ServletException, IOException {
170         processRequest(request, response);
171     }
172 
173     /**
174      * Handles the HTTP <code>POST</code> method.
175      * @param request servlet request
176      * @param response servlet response
177      * @throws ServletException if a servlet-specific error occurs
```

```
178      * @throws IOException if an I/O error occurs
179      */
180     @Override
181     protected void doPost
182     (HttpServletRequest request, HttpServletResponse response)
183     throws ServletException, IOException {
184         processRequest(request, response);
185     }
186 
187     /**
188      * Returns a short description of the servlet.
189      * @return a String containing servlet description
190      */
191     @Override
192     public String getServletInfo() {
193         return "Short description";
194     }// </editor-fold>
195 }
```

---

*uploadDataNamed(), executeTaskProcessDataNamed()e downloadDataNamed()* são métodos que fazem acesso ao *file system* (sistema de arquivos) e promovem o *upload* (envio) dos dados, a execução da tarefa numérica e o download (baixa) da resposta respectivamente. Os métodos *uploadDataNamed() e executeTaskProcessDataNamed()* chamam *threads* que tornam o procedimento assíncrono.

### 5.1.3 O programa cliente que roda na máquina desktop

Até este ponto, nada foi dito à respeito dos programas clientes e nem dos programas numéricos. Isso porque o Numerus Cloud Controller e o NTE são sempre os mesmos e funcionam sempre no sentido genérico. Os programas dessas camadas desconhecem o conteúdo lógico da programação numérica. O programa cliente escrito em Java Swing deve apenas saber o formato dos dados de entrada e de saída especificados pelo autor do programa numérico e saber o id da tarefa numérica que está sendo requisitada. E mais nada. A seguir, será mostrada a parte do código do programa em Java Swing responsável pelo acesso ao Numerus Cloud Controller, isto é, a parte que promove o acesso ao Web Services do Numerus Cloud.

```
1  package ncclient3;
2  
3  import java.io.BufferedReader;
4  import java.io.InputStream;
5  import java.io.InputStreamReader;
6  import java.net.URL;
7  import java.net.URLConnection;
8  import java.util.ArrayList;
9  import java.util.List;
10 import java.util.Map;
11 import javax.xml.namespace.QName;
12 import javax.xml.ws.BindingProvider;
```

```
13 import javax.xml.ws.WebServiceClient;
14 import nccommon.to.StatusTO;
15 import nccommon.util.UtilBeans;
16
17 /**
18  * This class represents the client interface with the Numerus Cloud
        Server.
19  * It contains all web service client stuff to exchange data with NC
        Server.
20  *
21  * @author JoÃ$o Carlos
22  */
23 public class NCClient3Business {
24
25     private NumerusCloudWSPortType port;
26     private String sessionId;
27     private String taskProcessId;
28
29     /**
30      * Open connection to http://numeruscloud.com/clouddns.php
31      * @return Numerus Cloud host and port
32      */
33     private static String getNumerusCloudDNS()
34     {
35         String host_port = null;
36         try
37         {
38             URL url = new URL ("http://numeruscloud.com/clouddns.php
                    ");
39             URLConnection conn = url.openConnection();
40             InputStream is = conn.getInputStream();
41             BufferedReader reader = new BufferedReader
42       (new InputStreamReader (is));
43             host_port = reader.readLine();
44             reader.close();
45         }
46         catch (Exception ex)
47         {
48             ex.printStackTrace();
49         }
50         return host_port;
51     }
52     private static String getEndpointAddressProperty()
53     {
54         String endpointAddressProperty = "http://" +
55   getNumerusCloudDNS().trim()
56   + "/axis2/services/NumerusCloudWS?wsdl";
57         return endpointAddressProperty;
58     }
59
60     public NCClient3Business()
61     {
62         WebServiceClient ann
63   = NumerusCloudWS.class.getAnnotation (WebServiceClient.class);
64         URL wsdlURL = NCClient3Business.class.getResource
65   ("./wsdl/numeruscloud.wsdl");
66         NumerusCloudWS serviceObj =
67   new NumerusCloudWS (wsdlURL,
68   new QName (ann.targetNamespace(), ann.name()));
69
70         NumerusCloudWSPortType service
71   = serviceObj.getNumerusCloudWSSOAP11PortHttp();
72         BindingProvider bp = ((BindingProvider) service);
```

```
73          Map context = bp.getRequestContext();
74          context.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
75  getEndpointAddressProperty ());
76
77          this.port = service;
78      }
79
80      private List<NCStatusListener> ncListeners = new ArrayList<>();
81
82      public void addNCStatusListener(NCStatusListener
            ncStatusListener) {
83          ncListeners.add(ncStatusListener);
84      }
85
86      public void removeNCStatusListener
87      (NCStatusListener ncStatusListener) {
88          ncListeners.remove(ncStatusListener);
89      }
90
91      public List<NCStatusListener> getNCStatusListeners() {
92          return ncListeners;
93      }
94
95      private void callChangeStatusUploadFromListeners
96      (StatusTO statusTO) {
97          for (NCStatusListener ncListener : ncListeners) {
98              ncListener.changeStatusUpload(statusTO);
99          }
100     }
101
102     private void callChangeStatusExecuteTaskFromListeners
103     (StatusTO statusTO) {
104         for (NCStatusListener ncListener : ncListeners) {
105             ncListener.changeStatusExecuteTask(statusTO);
106         }
107     }
108
109     private void callChangeStatusDownloadFromListeners
110     (StatusTO statusTO) {
111         for (NCStatusListener ncListener : ncListeners) {
112             ncListener.changeStatusDownload(statusTO);
113         }
114     }
115
116     private void callChangeStatusLoginFromListeners
117     (StatusTO statusTO) {
118         for (NCStatusListener ncListener : ncListeners) {
119             ncListener.changeStatusAuthenticate(statusTO);
120         }
121     }
122
123     private void callChangeStatusTaskCreatingFromListeners
124     (StatusTO statusTO) {
125         for (NCStatusListener ncListener : ncListeners) {
126             ncListener.changeStatusTaskCreating(statusTO);
127         }
128     }
129
130     public String login(String loginName, String password) {
131         StatusTO statusTO = new StatusTO();
132         try {
133             statusTO.setCode(StatusTO.ErrorCode.DOING);
134             statusTO.setMessage("Login : ");
```

```
135             callChangeStatusLoginFromListeners(statusTO);
136
137             this.sessionId = port.authenticate(loginName, password);
138
139             statusTO.setCode(StatusTO.ErrorCode.OK);
140             statusTO.setMessage("Session key from login : "
141       + this.sessionId);
142
143         } catch (Exception ex) {
144             statusTO.setCode(StatusTO.ErrorCode.GENERIC_ERROR);
145             statusTO.setMessage("Error login : " + ex.getMessage());
146         } finally {
147             callChangeStatusLoginFromListeners(statusTO);
148         }
149         return this.sessionId;
150     }
151
152     public String createTaskProcessId(String sessionId, String
        taskId) {
153         StatusTO statusTO = new StatusTO();
154         try {
155             statusTO.setCode(StatusTO.ErrorCode.DOING);
156             statusTO.setMessage("Task process id creating :");
157             callChangeStatusTaskCreatingFromListeners(statusTO);
158
159             this.setTaskProcessId(port.createTaskProcessId
160       (sessionId, taskId));
161             statusTO.setCode(StatusTO.ErrorCode.OK);
162             statusTO.setMessage("Task process id created :" +
163       this.getTaskProcessId());
164
165         } catch (Exception ex) {
166             ex.printStackTrace();
167             statusTO.setCode(StatusTO.ErrorCode.GENERIC_ERROR);
168             statusTO.setMessage("Error task process id creating : "
                +
169       ex.getMessage());
170
171         } finally {
172             // call changeStatusUpload method from numerus cloud
                listeners:
173             callChangeStatusTaskCreatingFromListeners(statusTO);
174         }
175         return this.getTaskProcessId();
176     }
177
178     public void upload(final String dataName, final byte[] data) {
179         final NumerusCloudWSPortType port = this.port;
180         Runnable running = new Runnable() {
181             @Override
182             public void run() {
183                 try {
184                     port.uploadDataNamed(sessionId,
185       getTaskProcessId(), dataName, data);
186                     boolean finished = false;
187                     do {
188                         String jsonStatusString =
189       port.getStatusOfUploadDataNamed(getTaskProcessId(), dataName);
190
191                         StatusTO statusTO =
192       UtilBeans.convertStatusJSONIntoStatusTO
193       (jsonStatusString);
194
```

```
195                         // call changeStatusUpload method from
                                numerus cloud listeners:
196                         callChangeStatusUploadFromListeners(statusTO
                                );
197                         Thread.sleep(5000);
198                         finished = statusTO.getCode().equals
199       (StatusTO.ErrorCode.OK);
200                     } while (!finished);
201                 } catch (Exception ex) {
202                     StatusTO statusTO = new StatusTO();
203                     statusTO.setCode(StatusTO.ErrorCode.
                            GENERIC_ERROR);
204                     statusTO.setMessage(ex.getMessage());
205                     statusTO.setTaskProcessId(getTaskProcessId());
206                     statusTO.setDataName(dataName);
207                     // call changeStatusUpload method from numerus
                            cloud listeners:
208                     callChangeStatusUploadFromListeners(statusTO);
209                 }
210             }
211         };
212         new Thread(running).start();
213     }
214
215     public void executeTask(final String dataName) {
216         final NumerusCloudWSPortType port = this.port;
217         Runnable running = new Runnable() {
218             @Override
219             public void run() {
220                 System.out.println("run task again ...");
221                 try {
222                     port.executeTaskDataNamed
223       (sessionId, getTaskProcessId(), dataName);
224                     boolean finished = false;
225                     do {
226                         String jsonStatusString =
227       port.getStatusOfExecuteTaskProcessDataNamed
228       (getTaskProcessId(), dataName);
229
230                         StatusTO statusTO =
231       UtilBeans.convertStatusJSONIntoStatusTO
232       (jsonStatusString);
233
234                         // call changeStatusExecuteTask method from
                                numerus cloud listeners:
235                         callChangeStatusExecuteTaskFromListeners(
                                statusTO);
236                         Thread.sleep(3000);
237                         finished =
238       statusTO.getCode().equals(StatusTO.ErrorCode.OK);
239
240                         if(
241       (statusTO.getCode().equals
242       (StatusTO.ErrorCode.TIMEOUT))
243                                 ||
244                         (statusTO.getCode().
245       equals(StatusTO.ErrorCode.GENERIC_ERROR))
246                         )
247                         {
248                             break;
249                         }
250                     } while (!finished);
251                 } catch (Exception ex) {
```

```
252                         StatusTO statusTO = new StatusTO();
253                         statusTO.setCode(StatusTO.ErrorCode.
                                GENERIC_ERROR);
254                         statusTO.setMessage(ex.getMessage());
255                         statusTO.setDataName(dataName);
256                         // call changeStatusExecuteTask method from
                                numerus cloud listeners:
257                         callChangeStatusExecuteTaskFromListeners(
                                statusTO);
258                     }
259                 }
260             };
261             new Thread(running).start();
262         }
263
264         public byte[] download(String dataName) {
265             return port.downloadDataNamed(sessionId,
266       getTaskProcessId(), dataName);
267         }
268
269         /**
270          * @return the taskProcessId
271          */
272         public String getTaskProcessId() {
273             return taskProcessId;
274         }
275
276         /**
277          * @param taskProcessId the taskProcessId to set
278          */
279         public void setTaskProcessId(String taskProcessId) {
280             this.taskProcessId = taskProcessId;
281         }
282 }
```

O constructor *public NCClient3Business()* instancia o objeto *private Numerus-CloudWSPortType port*. Este objeto é a abstração do Web Services e será usado na chamada de qualquer serviço do Web Services. É mostrado também o método de Login dentro do qual há a chamada do método *authenticate* do objeto *port*.

### 5.1.4 O programa em Android

Abaixo, a classe pertencente ao programa cliente feito em Android e que é responsável pelo acesso ao Numerus Cloud Controller:

```
1  package com.cluster;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import org.ksoap2.SoapEnvelope;
7  import org.ksoap2.serialization.SoapObject;
8  import org.ksoap2.serialization.SoapPrimitive;
9  import org.ksoap2.serialization.SoapSerializationEnvelope;
10 import org.ksoap2.transport.HttpTransportSE;
11
```

```
12 import android.app.Activity;
13 import android.content.Context;
14 import android.content.Intent;
15 import android.graphics.Canvas;
16 import android.graphics.Color;
17 import android.graphics.Paint;
18 import android.os.Bundle;
19 import android.util.Log;
20 import android.view.Menu;
21 import android.view.SurfaceHolder;
22 import android.view.SurfaceView;
23 import android.view.Window;
24 import android.widget.FrameLayout;
25
26 public class DisplayCluster extends Activity {
27   Paint paint;
28   Grid grid;
29   DisplayView displayView;
30   int cOrder;
31   List<Point> centroids = new ArrayList<Point>();
32   List<Integer> colors = new ArrayList<Integer>();
33
34
35
36     private static final String SOAP_ACTION
37     = "http://clusterserver/clusterize";
38     private static final String METHOD_NAME = "clusterize";
39     private static final String NAMESPACE = "http://clusterserver";
40     private static final String URL
41     = "http://192.168.0.101:8080/axis2/services/MyClusterServer?wsdl
          ";
42
43   @Override
44   protected void onCreate(Bundle savedInstanceState) {
45     super.onCreate(savedInstanceState);
46     requestWindowFeature(Window.FEATURE_NO_TITLE);
47
48     colors.add(Color.BLUE);
49     colors.add(Color.CYAN);
50     colors.add(Color.DKGRAY);
51     colors.add(Color.GRAY);
52     colors.add(Color.GREEN);
53     colors.add(Color.MAGENTA);
54
55     Intent data = getIntent();
56     grid = (Grid)data.getSerializableExtra("GRID");
57     final int order = (int)data.getIntExtra("ORDER", 2);
58     cOrder = order;
59
60     FrameLayout displayClusterView = new FrameLayout(this);
61
62     displayView = new DisplayView(this);
63
64     displayClusterView.addView(displayView);
65
66     setContentView(displayClusterView);
67
68           Thread networkThread = new Thread() {
69     @Override
70     public void run() {
71     try
72     {
73        SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
```

```
74          request.addProperty("arg0", order);
75          request.addProperty("arg1", grid.toSoap());
76          SoapSerializationEnvelope envelope =
77          new SoapSerializationEnvelope(SoapEnvelope.VER11);
78          envelope.setOutputSoapObject(request);
79
80          HttpTransportSE ht = new HttpTransportSE(URL);
81          ht.call(SOAP_ACTION, envelope);
82          final  SoapPrimitive response
83          = (SoapPrimitive)envelope.getResponse();
84          final String str = response.toString();
85          centroidFromSoap(str);
86
87          runOnUiThread (new Runnable(){
88          public void run() {
89          displayView.start();
90          }
91          });
92      }
93      catch (final Exception e) {
94        Log.e("ERROR", e.toString());
95      }
96        }
97              };
98        networkThread.start();
99    }
100 }
```

### 5.1.5 O programa numérico

O programa numérico é o programa que usa o método de clusterização XCM (*Xavier Clustering Method*). O programa recebe como entrada de dados, pontos que são chamados de observações. Estes são pontos que estão em qualquer dimensão. No exemplo, foi usada dimensão 2, isto é, pontos no $\mathbb{R}^2$.

#### Dois clientes: Swing e Android

O sistema proposto NumerusCloud permite que se use múltiplos clientes diretamente sobre as chamadas da *cloud computing*. Essa forma de se trabalhar é chamada de SOA-MC (*Service Oriented Architecture - Multiple Client* - Arquitetura Orientada a Serviços para Múltiplos Clientes).

Os dois programas clientes, o *front end* (aplicação *desktop*) feito em Java Swing e o programa em Android, mostraram resultados conforme o esperado. Nos exemplos dos clientes, observações ou pontos foram pintados na tela via mouse ou *touch screen*, no caso do programa em Android. Dessa forma, um conjunto de pontos no $\mathbb{R}^2$ foi coletado e enviado ao NumerusCLoud para a clusterização, isto é, o cálculo dos

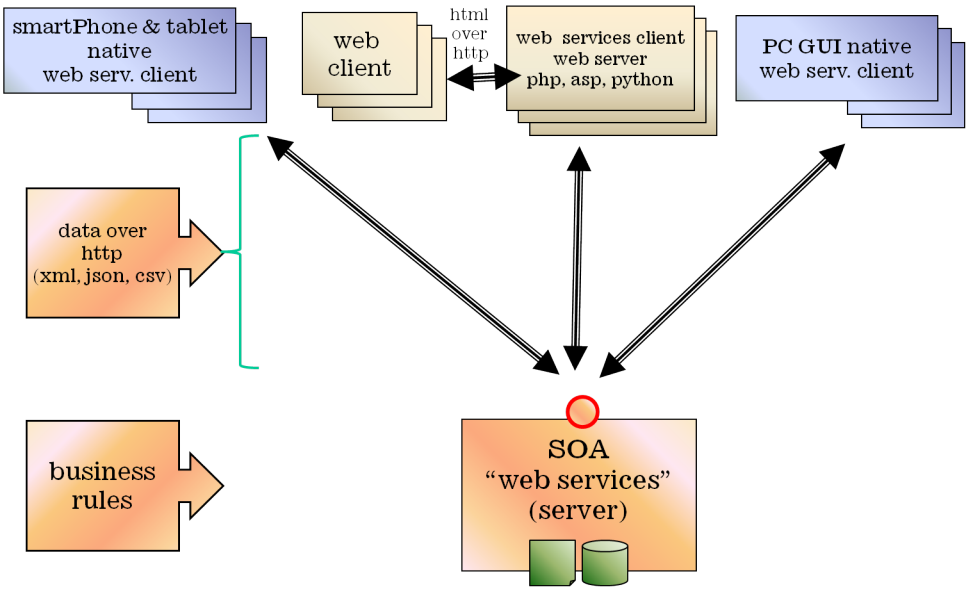


Figura 5.1: Arquitetura SOA-MC (Service Oriented Architecture - Multiple Client)

centroides. Os resultados foram obtidos com sucesso, conforme o esperado. Os *clusters* foram calculados pelo programa numérico e, no cliente, foram pintados na tela. Os resultados são apresentados a seguir:

### Conclusão dos resultados

Os resultados mostrados nos 2 clientes comprovaram que, para o Numerus Cloud, os clientes podem ser heterogêneos quanto ao ambiente e plataforma. Clientes diferentes acessaram um mesmo serviço, mostrando resultados com sucesso. Para que os clientes possam acessar os serviços disponibilizados pelo Numerus Cloud, é necessário que o arquivo WSDL que descreve os métodos, parâmetros de entrada e respostas dos serviços, esteja disponível e possa ser acessado. A partir desse arquivo WSDL, qualquer programa cliente escrito em qualquer linguagem pode chamar os serviços do Numerus Cloud.

Pela arquitetura proposta do Numerus Cloud, os clientes apenas enxergam a camada Numerus Cloud Controller. Desconhecem a existência do programa NTE e ,mais ainda, desconhecem totalmente a localização da rotina numérica.

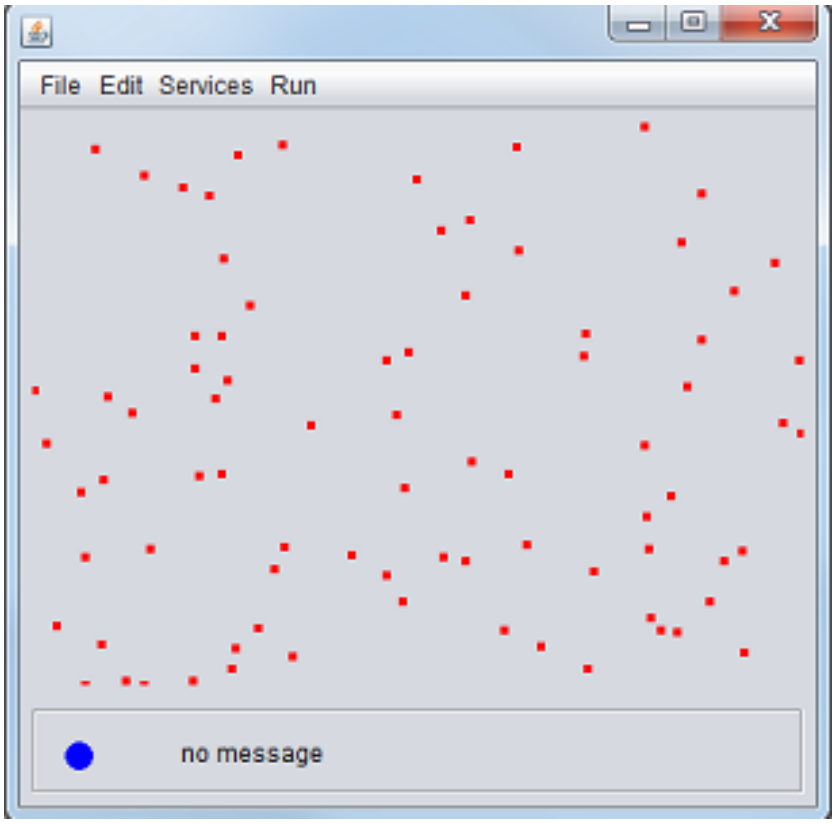


Figura 5.2: Observações no Swing

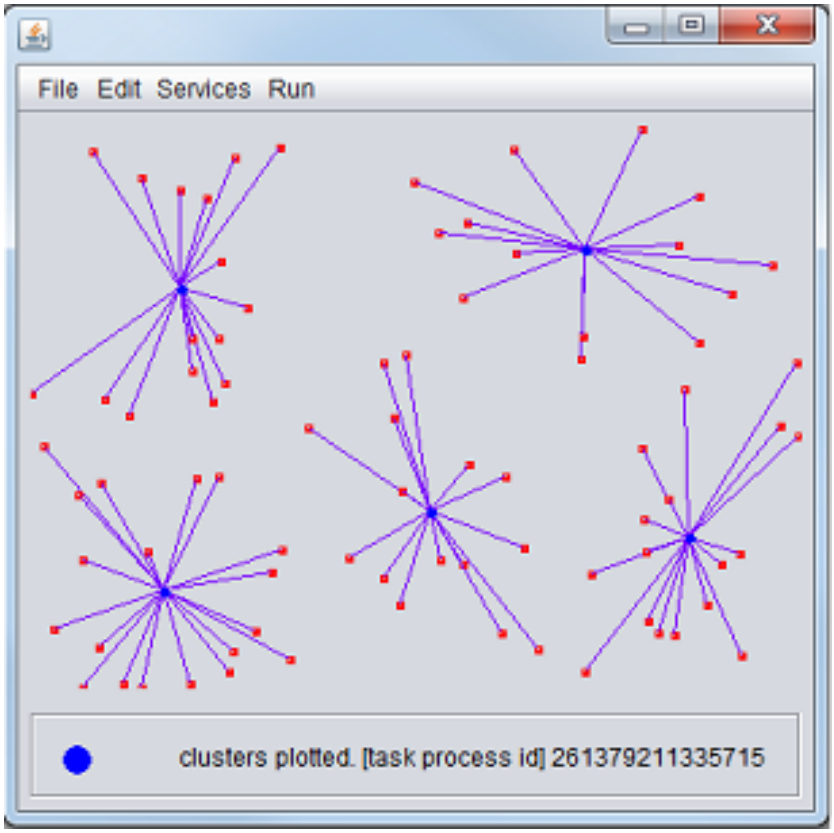


Figura 5.3: Observações e Clusters no Swing

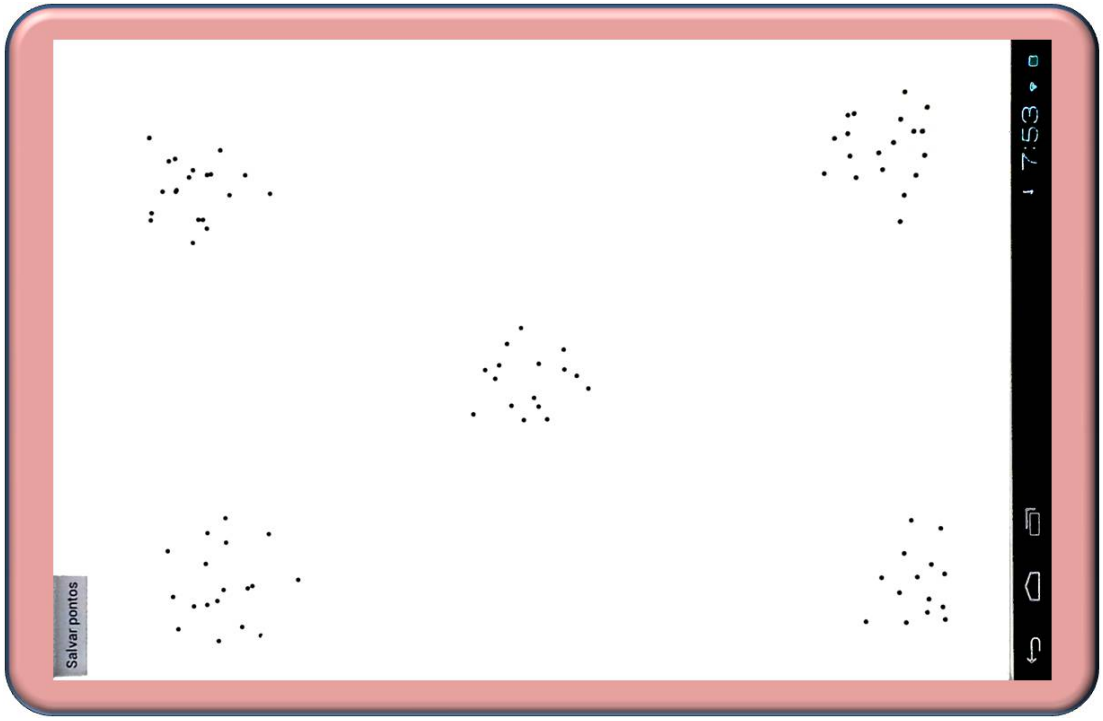


Figura 5.4: Observações no Android

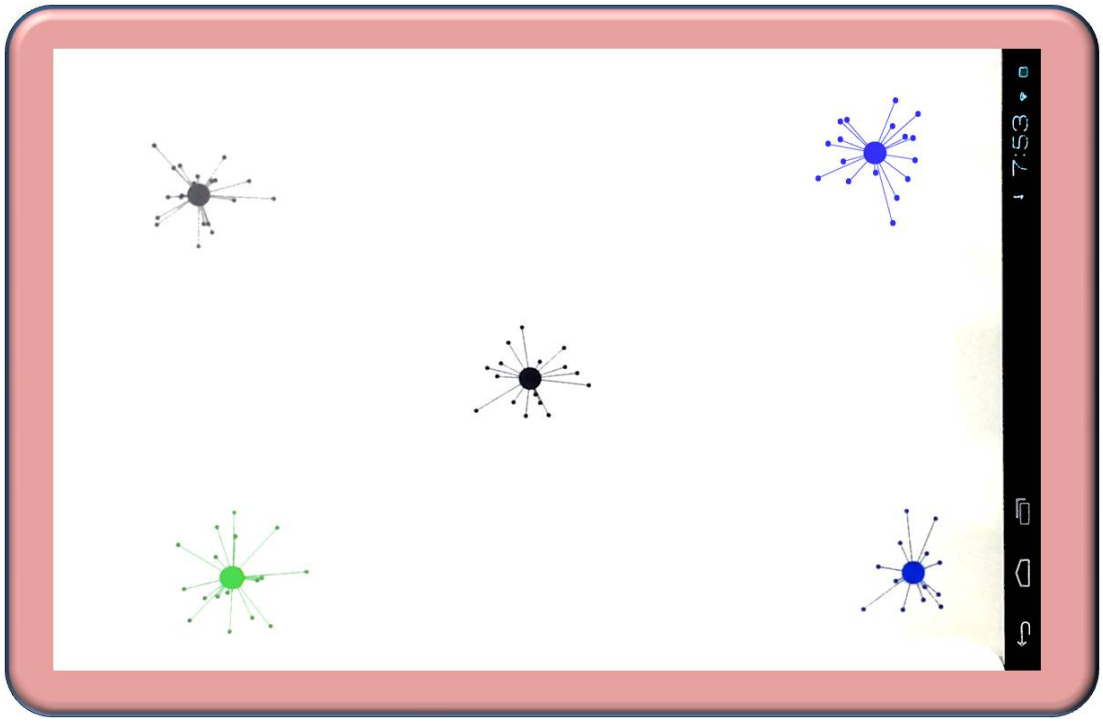

Figura 5.5: Observações e clusters no Android

## 5.2 Avaliação do Cumprimento dos Objetivos

Nesta seção, será comentada como a arquitetura Numerus Cloud proposta é capaz de solucionar as desvantagens da solução descrita em 2.3.3.

- A centralização do processamento numérico numa mesma máquina servidora tende a criar problemas sérios de escalabilidade.

  Com a arquitetura Numerus Cloud, a execução da rotina numérica ocorre na máquina especificada para essa finalidade (*NTE - Numeric Task Executor*), e não no servidor. Isso por si só faz aliviar muito o consumo de recursos computacionais do servidor. Ainda assim, pode haver engargalamento (problemas de escalabilidade) tanto no servidor quanto no NTE.

  Os problemas de engargalamento no servidor podem ser resolvidos com uma técnica que vamos chamar de "*cloud DNS*". Essa técnica consiste em instanciar um conjunto de computadores para a tarefa de servidor, e fazer os clientes acessarem em rodízio cada um deles. Nesse caso, faz-se um serviço muito simples e de alta disponilibilidade que apenas retorna o número IP do servidor que realizará a tarefa do servidor Numerus Cloud. Dessa forma, divide-se a carga dos clientes do Numerus Cloud por quantos servidores forem necessários até que cada servidor trabalhe sempre com carga abaixo daquela que é o seu limite máximo.

  Se essa mesma técnica fosse aplicada na arquitetura descrita em 2.3.3, surgiria o problema de implantar a rotina numérica em cada servidor onde é executada, e com isso se perderia justamente uma das grandes vantagens que se deseja obter.

  Os problemas de engargalamento do NTE são de outra natureza, pois há que se levar em conta o peso (consumo de recursos computacionais) da rotina numérica que se pretende executar. Caso o engargalamento da NTE ocorra pelo fato de que o peso de uma única execução numérica seja muito grande, a solução é fazer *upgrade* de hardware na NTE. Caso o engargalamento da NTE ocorra pelo fato de haver muitas requisições simultâneas do mesmo serviço numérico, pode-se solucionar esse engargalamento muitiplicando-se

as máquinas NTE, e programar a Numerus Cloud para acessar as NTE em rodízio.

- No momento de atualização da rotina numérica, é necessário se fazer *upload* do novo pacote compilado e reinicialização da aplicação que contém a rotina numérica. Isso causa interrupção temporária da aplicação do servidor.

  Com o uso da arquitetura Numerus Cloud, a atualização da rotina numérica ocorre meramente por se substituir o executável na máquina (*NTE - Numeric Task Executor*). Isso pode ser feito sem sequer se notificar nem o servidor Numerus Cloud nem o cliente final.

- O autor da rotina numérica precisa entregar o arquivo executável para a pessoa que controla o computador servidor. Pode haver constrangimento de segurança do autor em realizar essa entrega, pois quem controla o computador servidor pode violar a propriedade intelectual do autor da rotina numérica.

  Com o uso da arquitetura Numerus Cloud, a máquina NTE pode ficar em local ou sob condição controlada exclusivamente pelo autor da rotina numérica, o que lhe permite executar o serviço de atualização da rotina numérica no momento e da forma como mais lhe convier.

  Não se requer do NTE acesso direto nem remoto ao seu sistema de arquivos, nem conta de usuário. É suficiente que se instale na NTE o software agente da Numerus Cloud, que fará interface de dados com o servidor Numerus Cloud.

- O formato técnico do arquivo da rotina numérica precisa ser compatível com o do servidor de execução remota (se for um servidor de aplicação em Java, um formato popular é o *.war). O empacotamento da rotina numérica nesse formato não costuma ser bem aceito para uma pessoa com perfil de autor de rotina numérica. A necessidade de se fazer num mesmo pacote ambas as tarefas (empacotar para servidor de aplicação e a execução da rotina numérica ela própria) é uma dificuldade de implantação.

  A escola de formação de profissionais que desenvolvem software com finalidade de processamento científico e em ponto flutuante (isto é, rotinas numéricas), tende a enfatizar habilidades muito específicas em desenvolvimento de soft-

ware. A escola de formação de profissionais que desenvolvem software com finalidade de implantar servidores Web ou Web Services tende a enfatizar habilidades muito diferentes daquelas observadas para processamento científico.

A solução descrita em 2.3.3 requer que seja feita a implantação a partir de um arquivo que contém dentro de si, ao mesmo tempo, a execução da rotina numérica e também o empacotamento necessário para a execução num servidor de aplicação. Essa necessidade não ocorre no caso da arquitetura Numerus Cloud proposta, pois o agente Numeurs Cloud que é executado na NTE realiza a tarefa de interface de dados com o servidor Numerus Cloud, de forma que a rotina numérica trata a entrada e saída de dados pelo sistema de aquivos do NTE, de forma muito usual para processamento científico.

Caso se deseje medir a quantidade de melhoria de facilidade de implantação provido pela arquitetura proposta, seria necessário que se fizesse um experimento como descrito a seguir. Várias implantações de rotina numérica devem ser realizadas para cada uma das arquiteturas (a solução descrita em 2.3.3 e a Numerus Cloud proposta). O tempo que se vai gastar desde o recebimento dos requisitos até a implantação final deve ser medido para cada caso, e uma estatística deve ser feita.

Um experimento como esse é de grande complexidade, e por isso foi considerado fora do escopo dessa tese.

Por experiência de estimativa de esforço de desenvolvimento de software para cada um desses segmentos (rotinas numéricas e servidores Web Services) permite-se especular que o resultado de um experimento como esse é bastante previsível, e indicará que a arquitetura proposta de fato facilita a implantação de rotinas numéricas.

# Capítulo 6

# Conclusão

Nesse trabalho, foi proposta uma arquitetura de software - chamada de NumerusCloud - que usa tecnologia de *cloud computing* (computação em nuvem) para implementar com 3 camadas um sistema que permite automatizar o *deployment* (implantação) de rotinas numéricas. A arquitetura proposta oferece importantes vantagens sobre as técnicas conhecidas de execução remota, incluindo isolamento de camada entre o agente de *deployment* (implantação) e o autor da rotina numérica, o que provê proteção à propriedade intelectual pois o programa numérico reside numa máquina totalmente controlada pelo autor e cuja localização é desconhecida pelo próprio sistema Numerus Cloud. Algumas características de *cloud computing* (computação em nuvem) são implicitamente incluídas na arquitetura proposta, tornando-a mais flexível e útil. Essas características permitem a virtualização de recursos de hardware e software, e, consequentemente, criam um atrativo para o cliente do serviço numérico, que não se preocupa com questões mais complexas relacionadas com TI e que não estão associadas com a lógica do negócio da aplicação. O Web Services , sendo a ferramenta usada para o acesso remoto aos serviços do Numerus Cloud, permitiu que programas executados em clientes heterogêneos quanto à plataforma e linguagem pudessem acessar os serviços como demonstrado na implementação e teste de 2 tipos de clientes: um programa GUI em um PC e outro programa em um dispositivo móvel Android. Os clientes do Numerus Cloud tem portanto duas vantagens: por um lado, a herança das características da *cloud computing* (computação em nuvem), que esconde a complexidade em termos de software e hardware, permitindo o uso de recursos mais poderosos visando uma melhor perfor-

mance; do outro lado, o baixo acoplamento entre os programas clientes e o Numerus Cloud permitido pelo Web Services, torna o acesso disponível para todos os tipos de clientes.

E em relação ao autor do programa numérico, o isolamento, além de permitir um *deployment* (implantação) automático do programa e uma proteção à propriedade intelectual (caso seja necessária), oferece ainda uma vantagem adicional: o autor pode escrever seu programa em qualquer linguagem, uma vez que a arquitetura foi moldada para que as requisições aos serviços numéricos passassem primeiramente pelo componente NTE (que se comunica também via Web Services com o Numerus Cloud). O NTE tem a capacidade de chamar qualquer programa executável ou *script* feito em qualquer linguagem.

O trabalho também propôs uma camada de software para implementar controle de acesso para sistemas para nuvem, chamado de DOUA (*Database Oriented Usecase Authorization*). Essa é uma proposição genérica que serve para qualquer software na nuvem que requeira controle de acesso, isto é, autorização para execução de caso de uso. O sistema NumerusCloud é ele próprio feito com arquitetura de 3 camadas. O DOUA é uma (sub) camada da camada do sistema NumerusCloud que é executada na nuvem.

## 6.1 Trabalhos futuros

**Política de uso:** O modelo do Numerus Cloud oferece suporte para implementação de política de uso colocada pelo autor do programa numérico. Apesar de não ter sido feita uma implementação, o Numerus Cloud permite a inclusão das regras de negócio que tornam possível a verificação de uma política de uso. A política de uso seria dependente do programa numérico e do tipo de cliente. Um *script* relacionado com a política de uso poderia ser executado na camada do Numerus Cloud Controller.

**Serviços prioritários:** O Numerus Cloud, através da camada NTE, está preparado para a colocação de prioridades na execução de certos certos programas numéricos. Uma ideia seria estabelecer prioridades dependentes do programa numérico e do tipo de cliente, analogamente à política de uso.

# Referências Bibliográficas


[1] The Economist 2009-10-15. Cloud computing: Clash of the clouds, http://www.economist.com/node/14637206, acessado em 17 de setembro de 2013. 2009.

[2] V. S. Sunderam A. T. Krantz. Client server computing on message passing systems: Experiences with pvm-rpc. *Euro-Par'97 Parallel Processing Lecture Notes in Computer Science Volume 1300, 1997, pp 110-117*, 1997.

[3] Vijay Karamcheti Alexander Totok. Infrastructure for automatic dynamic deployment of j2ee applications in distributed environments. *Component Deployment Lecture Notes in Computer Science Volume 3798, 2005, pp 17-32*, 2005.

[4] Harm; Tong Qiang Guo; Liu Guo Ning Amies, Alex; Sluiman. Infrastructure as a service cloud concepts. *Developing and Hosting Applications on the Cloud. IBM Press. ISBN 978-0-13-306684-5*, 2012.

[5] Bruce Nelson Andrew Birrell. Remote procedure call. *Software System Award citation. Association for Computing Machinery.*, 1994.

[6] A. S. Barbosa, F. P; Charão. Computing e cloud computing uma relação de diferenças, semelhanças, aplicabilidade e possibilidades de cooperação entre os dois paradigmas. *Seminários de Arquiterura de Sistemas de Alto Desempenho, Santa Maria*, 2009.

[7] André B.Bondi. Characteristics of scalability and their impact on performance. *ACM WOSP 00 Proceedings of the 2nd International Workshop on Software and Performance, New York, NY.*, 2000.

[8] Carl Brooks. Amazon's early efforts at cloud computing partly accidental. *IT Knowledge Exchange. Tech Target*, 2010.

[9] OW2 Bull. Jonas : Java open application server. *http://jonas.objectweb.org/*, 2013.

[10] Ethan Cerami. Web services essentials distributed applications with xml-rpc, soap, uddi and wsdl. *O'Reilly Media Released: February 2002*, 2002.

[11] John R. Corbin. Future directions of rpc programming. *The Art of Distributed Applications Sun Technical Reference Library 1991, pp 245-256*, 1991.

[12] Microsoft Corporation. Understanding application deployment and upgrade. *Microsoft BizTalk Server 2006*, 2006.

[13] G Kousiouris K Oberle T Voith M Boniface E Oliveros T Cucinotta S Berger D Kyriazis, A Menychtas. A real-time service oriented infrastructure. *International Conference on Real-Time and Embedded Systems (RTES 2010), Singapore*, 2010.

[14] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. Design patterns: elements of reusable object-oriented software. *Addison-Wesley, Boston (1995)*, 1995.

[15] Ryan; Falvey. Regulation of the cloud in india. *Journal of Internet Law 15 (4)*, 2011.

[16] Apache Software Foundation. Apache tomcat. *http://tomcat.apache.org/*, 2013.

[17] Martin Fowler. Analysis patterns reusable objects models). *Addison-Wesley Professional, 1997 - Computers*, 1997.

[18] Simson Garfinkel. The cloud imperative. technology review (mit). *National Institute of Standards and Technology.*, 2011.

[19] Object Management Group. The common object request broker: Architecture and specifications. *The Common Object Request Broker: Architecture and Specifications, 2.0 (draft) ed, May 1995*, 1995.

[20] Galen Gruman. What cloud computing really means. *InfoWorld, http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031, acessado em 17 de setembro de 2013*, 2008.

[21] Mohammad Hamdaqa. A reference model for developing cloud applications. *International Conference on Cloud Computing and Services Science*, 2011.

[22] Red Hat. J2ee application deployment. *JBoss Application Server*, 2013.

[23] Y. Guo He, Sijin; L. Guo. Real time elastic cloud management for limited resources. *Proceedings of 2011 IEEE 4th International Conference on Cloud Computing (Cloud 2011): 622 629. doi:10.1109/CLOUD.2011.47. ISBN 978-0-7695-4460-1.*, 2011.

[24] IBM. Launch of ibm smarter computing, http://www.ibm.com/smarter-computing/, acessado em 17 de setembro de 2013. 2011.

[25] Luis Rodero-Merino Álvaro Polo Juan J. Hierro Juan Cáceres, Luis M. Vaquero. Service scalability over the cloud). *Handbook of Cloud Computing 2010, pp 357-377*, 2010.

[26] Ming; M. Humphrey Mao. A performance study on the vm startup time in the cloud. *Proceedings of 2012 IEEE 5th International Conference on Cloud Computing (Cloud2012): 423. doi:10.1109/CLOUD.2012.103. ISBN 978-1-4673-2892-0.*, 2012.

[27] Alta van der Merwe Mariana Carroll, Paula Kotzé. Securing virtual and cloud environments. *Cloud Computing and Services Science, Service Science: Research and Innovations in the Service Economy. Springer Science+Business Media.*, 2012.

[28] Rob Harrop Jan Machacek Michael Wessler, Erin Mulder. J2ee application deployment. *Oracle Application Server 10g 2004, pp 137-157*, 2004.

[29] Netbeans. Axis2 plugin for netbeans. *http://deadlock.netbeans.org/hudson/job/nbms-and- javadoc/lastStableBuild/artifact/nbbuild/nbms/updates.xml.gz; acessado em 17 de setembro de 2013.*

[30] Oracle. Java platform, enterprise edition (java ee) technical documentation. *http://docs.oracle.com/javaee/*, 2013.

[31] OW2. Ow2 consortium. *http://www.ow2.org/*, 2013.

[32] T. Pedrosa, P. H. C; Nogueira. Computação em nuvem. *http://www.ic.unicamp.br/˜ducatte/mo401/1s2011/T2/Artigos/G04-095352-120531-t2.pdf; acessado em 17 de setembro de 2013*, 2005.

[33] Timothy Grance Peter Mell. The nist definition of cloud computing. *National Institute of Standards and Technology.*, 2011.

[34] Nirupama Bhat Mundukur R. S. M. Lakshmi Patibandla, Santhi Sri Kurra. A study on scalability of services and privacy issues in cloud computing). *Distributed Computing and Internet Technology Lecture Notes in Computer Science Volume 7154, 2012, pp 212-230*, 2012.

[35] Jakub Rudzki. How design patterns affect application performance a case of a multi-tier j2ee application). *Scientific Engineering of Distributed Java Applications Lecture Notes in Computer Science Volume 3409, 2005, pp 12-23*, 2013.

[36] David Sklar. Developing lightweight web services with xml-rpc. *Essential PHP Tools: Modules, Extensions, and Accelerators 2004, pp 171-186*, 2004.

[37] Jean-Bernard Stefani Takoua Abdellatif, Jakub Kornas. J2ee packaging, deployment and reconfiguration using a general component model. *Component Deployment Lecture Notes in Computer Science Volume 3798, 2005, pp 134-148*, 2005.

[38] S. B. Villas-Boas. Doua (database oriented usecase authorization). *http://www.sbvb.com.br/proposals/DOUA-sbvb.com.br.pdf; acessado em 17 de setembro de 2013*, 2013.

[39] w3c. Web services glossary, http://www.w3.org/tr/ws-gloss/, acessado em 17 de setembro de 2013. *W3C*, 2004.

[40] w3c. Web services description language (wsdl) version 2.0 part 2: Adjuncts, http://www.w3.org/tr/wsdl20-adjuncts/, acessado em 17 de setembro de 2013. *W3C*, 2007.

[41] Anand M. White. A high-level framework for network-based resource sharing. *RFC 707. Augmentation Research Center*, 1975.

[42] Jonathan Kaplan William Crawford. J2ee design patterns. *O Reilly Media, Inc*, 2003.

[43] Liang Q. Bertino Wu, J. Improving scalability of software cloud for composite web services. *Cloud, IEEE International Conference on Cloud Computing, Honolulu, Hawaii, 143-146.*, 2009.

[44] A. E. Xavier. The hyperbolic smoothing clustering method. *Pattern Recognition*, 43 (3):731–737, 2010.