UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

CEZAR FUHR

IMPLEMENTAÇÃO DE WEB SERVICES RUBY ON RAILS E JRUBY NA AMAZON: UM ESTUDO EXPERIMENTAL SOBRE A INTEGRAÇÃO DE APLICAÇÕES EM SISTEMAS DISTRIBUÍDOS

MONOGRAFIA DE ESPECIALIZAÇÃO

MEDIANEIRA
2012

CEZAR VILMAR FUHR

IMPLEMENTAÇÃO DE WEB SERVICES RUBY ON RAILS E JRUBY NA AMAZON: UM ESTUDO EXPERIMENTAL SOBRE A INTEGRAÇÃO DE APLICAÇÕES EM SISTEMAS DISTRIBUÍDOS

Monografia apresentada como requisito parcial à obtenção do título de Especialista na Pós Graduação em Engenharia de Software, da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Medianeira.

Orientador: Prof. Fernando Schütz, Me.

MEDIANEIRA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Especialização em Engenharia de Software

TERMO DE APROVAÇÃO

Implementação de Web services Ruby on Rails e JRuby na Amazon: Um estudo experimental sobre a integração de aplicações em sistemas distribuídos

Por

Cezar Vilmar Fuhr

Esta monografia foi apresentada às 09h30min h do dia 17 de março de 2012, como requisito parcial para a obtenção do título de Especialista no curso de Especialização em Engenharia de Software, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O acadêmico será argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. M. Sc. Fernando Schütz
UTFPR – Campus Medianeira
(orientador)

_____
Prof. M.Sc. Juliano Rodrigo Lamb
UTFPR – Campus Medianeira

_____
Prof. Dr. Vilson Dalle Mole
UTFPR – Campus Medianeira

Dedico este trabalho á amigos, namorada, família, e todas as pessoas que notaram a minha escassez de tempo no período da elaboração do trabalho.

AGRADECIMENTOS

À Deus pelo dom da vida, pela fé e perseverança para vencer os obstáculos.

Aos meus pais, pela orientação, dedicação e incentivo nessa fase do curso de pós-graduação e durante toda minha vida.

Aos colegas de trabalho que de alguma forma contribuíram com opiniões, na troca de idéias sobre o trabalho.

Ao meu orientador e professor Fernando Schütz, que me orientou, pela sua disponibilidade, interesse e receptividade com que me recebeu e pela prestabilidade com que me ajudou.

Agradeço aos pesquisadores e professores do curso de Engenharia de Software, professores da UTFPR, Câmpus Medianeira.

Enfim, sou grato a todos que contribuíram de forma direta ou indireta para realização desta monografia.

“Lembre-se, hoje é o dia de amanhã que

tanto lhe preocupava ontem.”.

(Dale Carnegie)

RESUMO

Fuhr, Cezar Vilmar. Implementação de protocolo RPC em sistemas distribuídos. 2012. 66 p. Monografia (Especialização em Engenharia de Software) Universidade Tecnológica Federal do Paraná, Medianeira, 2012.

O presente trabalho tem o intuito de apresentar uma análise de tendência mundial das empresas de tecnologia, e mostra que as grandes empresas como Amazon, Google e Microsoft que estão investindo bastante em ambientes de computação em nuvem. É mostrado como é fácil criar servidores na Amazon que oferece alta disponibilidade e capacidade de aumentar os recursos de forma escalonável, dessa forma, empresas desenvolvedoras de software podem utilizar esses ambientes para desenvolverem seus produtos, ou ainda, empresas em geral podem diminuir seus custos migrando servidores para a nuvem. O trabalho ainda faz uma abordagem sobre software como serviço, os softwares especialistas, sistemas distribuídos e as principais formas de comunicação usando Web service, demonstrando que existem várias formas de comunicação entre Web services . Por fim, é desenvolvido um Web service especialista que implementa servidor RPC e é responsável pelo armazenamento e recuperação de arquivos XML, um Web service que gera gráficos através de dados XML e um aplicação desktop que se comunica com os dois sistemas distribuídos.

Palavras-chave: Plataformas Cloud, Sistemas distribuídos, Comunicação entre Web service, RPC.

ABSTRACT

Fuhr, Cezar Vilmar. Implementação de protocolo RPC em sistemas distribuídos. 2012. 66 p. Monografia (Especialização em Engenharia de Software) Universidade Tecnológica Federal do Paraná, Medianeira, 2012.

This paper aims to present a trend analysis of global technology companies, and showsthat big companies like Amazon, Google and Microsoft that are investing heavily in cloud computing environments. It is shown how easy it is to create servers in the Amazon that offers high availability and increase the resources in a scalable, thus software development companies can use these environments to develop their products, or evenbusiness in general can lower your moving costs servers to the cloud. The work alsomakes an approach to software as a service, software specialists, distributed systems and the main forms of communication using Web service, showing that there are various forms of communication between Web services. Finally, we developed a Web servicethat implements RPC server expert and is responsible for storing and retrieving XML files, a Web service that generates graphs using XML data and a desktop applicationthat communicates with both distributed systems.

Keywords: Cloud platforms, distributed systems, communication between Web services, RPC.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Por meio de sites de tecnologia, fóruns e revistas especializadas, percebe-se que empresas estão migrando seus produtos e serviços para a nuvem, á popularização de Data Centers e o preço mais acessível fazem com que isso aconteça rapidamente. Grandes empresas como Google, Amazon e Microsoft têm apostado nesse mercado e ao que tudo indica esta será uma “nova fronteira na era digital”.

No mercado atual de tecnologia, as empresas têm cerca de 3% de suas estruturas de TI na nuvem, e em alguns anos, os CIOS acreditam que esse número pode chegar a mais de 40%. Aliando aos muitos serviços que podem usufruir desse novo mercado, como: aplicações corporativas, tecnologias colaborativas, inteligência artificial, virtualização de sistemas operacionais e comunicação de dados, esse mercado terá impulsão para ter uma forte ascensão nos próximos anos (CONVERGENCIA DIGITAL, 2011).

Já as empresas brasileiras, 81% delas pretendem aumentar o orçamento de TI para o segmento de computação em nuvem já em 2012, maior inclusive que a média mundial que é de 55%. A expectativa do setor de TI é gerar novos negócios, ou ainda para diminuir os altos custos de infraestrutura, já que na maioria dos casos, na nuvem, o valor é pago pelo processamento utilizado (ADMINISTRADORES, 2011).

Segundo Taurion (2010), a empresa Google está trabalhando para se consolidar nesse mercado, por exemplo, permite que desenvolvedores testem e construam suas aplicações em um ambiente simulado com os principais sistemas operacionais, suportando linguagens de programação como o Python e o Java. Além disso, é possível escalar de forma automática a demanda de recursos, pois toda a demanda de hardware é feita em cima da própria plataforma da empresa. A alternativa do Google é a melhor para aplicações de pequeno porte, desenvolvidas por equipes pequenas, no estilo “Agile Development” (TAURION, 2010).

A empresa Amazon, que após perceber que poderia lucrar com sua infraestrutura de TI montada para suprir a demanda de vendas de suas lojas de varejo, desenvolveu uma plataforma onde é possível literalmente alugar um servidor remoto, sendo possível instalar vários sistemas operacionais e praticamente

qualquer software. A empresa fornece a administração e a manutenção da infraestrutura de armazenamento, deixando o cliente livre para desenvolver seu produto. A plataforma foi projetada para tolerar falhas e corrigir o sistema rapidamente, fornece grande flexibilidade para desenvolvedores e escalabilidade de demanda de hardware, o produto foi desenvolvido pela própria Amazon (CHAGANTI, 2011).

Ainda no quesito computação em nuvem, outra gigante que aparece forte em investimento é a Microsoft. A empresa domina boa parte das tecnologias e *software* para computadores *desktop* e agora investe para consolidar também no ambiente Web. A solução da empresa difere um tanto quanto as outras anteriores, porém também conta com muitas das características dos concorrentes como alta escalabilidade e alta disponibilidade na nuvem, fornecendo também suporte á ferramentas de desenvolvimento para que desenvolvedores possam criar produtos dentro de seu produto Cloud (GHEDIN, 2011).

Empresas de desenvolvimento de software mais atentas, já estão de olho nessas novas oportunidades de negócios, com isso espera-se que novos produtos, comecem a aparecer para essas plataformas já em pouco tempo. Aplicações web por si só já atrativas, somando com empresas desenvolvedoras que não terão a necessidade de se preocupar com a disponibilidade e escalabilidade de recursos, a computação em nuvem terá todos os ingredientes para dar certo.

Resolvidos os problemas de infraestrutura, que podem ser alugados, é necessário observar os vários tipos de comunicação entre sistemas web, pois quando ser trata de serviços para a nuvem, muitos deles são integrados a outros serviços. A chamada de procedimento remoto é umas das formas de comunicação em sistemas distribuídos e será discutida amplamente neste trabalho.

1.1 OBJETIVO GERAL

Desenvolver aplicações Web Services na nuvem como um estudo experimental para a implementação da tecnologia RPC em sistemas distribuídos.

1.2 OBJETIVOS ESPECÍFICOS

- Desenvolver o estudo bibliográfico
- Modelar e projetar os sistemas que servirão de estudos experimentais
- Testar e validar os modelos através da implementação de comunicação RPC nas linguagens Ruby/JRuby e Delphi.

1.3 JUSTIFICATIVA

A tecnologia da informação é melhorada constantemente. Antigamente os *softwares* rodavam centralizados em grandes *mainframes*, isso evoluiu para que fosse possível utilizar os softwares máquinas conectados em rede. Após isso, as próprias máquinas dos usuários evoluíram e nesse ponto foi possível rodar os *softwares* localmente, disseminando assim o uso de computadores domésticos. Hoje a tendência da tecnologia é que estes mesmos *softwares*, que agora são chamados de serviços, possam ser executados na *internet* (TANENBAUM e STEEN, 2007).

Grandes empresas dentre elas Amazon, Google e Microsoft, estão investindo pesado para proporcionar ambientes de desenvolvimento e plataformas escalonáveis. Essas empresas pretendem absorver parte do mercado e assim tornar-se referenciais no mercado de *Cloud Computing*. Embora não sejam os mesmos produtos, tanto Amazon, quanto Google e Microsoft apostam em serviços rodando na web, capacidade de escalabilidade de recursos e facilidades de desenvolvimento, deixando toda a infraestrutura pronta para os desenvolvedores, que por sua vez precisam somente focar no desenvolvimento do software.

As pequenas e médias empresas de desenvolvimento de software também devem aproveitar essa onda tecnológica. Se por um lado são pequenas para desenvolver sua própria infraestrutura, no outro, podem utilizar-se das plataformas das grandes empresas para produzir seus produtos. Exemplo disso seria a utilização do Windows Azure e dentro dela, utilizando o software de desenvolvimento da própria Microsoft, o Visual Studio.

As pequenas e médias empresas não somente as desenvolvedoras, podem ainda migrar aplicativos e softwares que já possuem para a nuvem, eliminando assim o custo de manutenção de servidores locais, como: Energia elétrica, falhas de hardware e pessoal capacitado para manter essas estruturas de TI.

Outra vantagem de serviços rodando na *web* em relação aos *softwares* instalados na máquina do usuário, é o fato das máquinas estarem expostos a vírus e erros de registro do sistema. A máquina do usuário ainda pode conter recursos limitados para o aplicativo funcionar corretamente, sem contar que em praticamente toda empresa, os computadores são protegidos por *antivírus*, *firewall* e bloqueio de portas. Todas essas barreiras dificultam para que *software* possa funcionar adequadamente, diferentemente de Web services.

Os Web services trabalham em cima de protocolos da *internet*, esses protocolos são liberados para acesso em praticamente todas as organizações. Além disso, para manter um serviço *web* é necessário ter um servidor e como o processamento fica por conta dos servidores e não dá máquina do usuário, esses servidores já são preparados para ter alta disponibilidade.

Os *softwares* na *web* podem ainda tornar serviços altamente especializados, e com alta disponibilidade e escalabilidade. Fornecendo APIs para a integração com outros servidores, e com isso vender processamento para terceiros, através de serviços interconectados. Dessa forma uma empresa faz um software e dispõe API para comunicação, a outra empresa faz outro software que pode ser integrado á primeira. Ambas as empresas ganham, a segunda por não precisar desenvolver o serviço e a primeira que terá lucro pela venda de processamento.

2 REFERENCIAL TEÓRICO

O presente capítulo apresenta os resultados de pesquisas bibliográficas efetuadas, com o intuito de estudar os conteúdos inerentes aos objetivos deste trabalho.

2.1 COMPUTAÇÃO EM NUVEM

Para Taurion (2009), O termo Computação em Nuvem ainda é relativamente novo e bastante amplo. Isso acontece por que grandes empresas de tecnologia tem diferentes definições sobre *Cloud Computing*. A utilização de recursos computacionais de máquinas remotas ou aplicações que são executadas diretamente da *web* são exemplos de computação em nuvem. Essa tecnologia é uma revolução natural da convergência de várias tecnologias e conceitos para a *internet* (TAURION, 2009).

Essa onda tecnológica, já estão sendo acompanhadas pelos executivos da Tecnologia da Informação, a ideia de se utilizar recursos ociosos de computadores independentes é bastante atrativa. Pelo lado do cliente também existe vantagens, pois não é necessário preocupar-se com investimentos em hardware, infraestrutura e disponibilidade (TAURION, 2009).

Uma das grandes vantagens da utilização de computação em nuvem é vantagem de se pagar somente o que é utilizado, ou seja, o pagamento é feito sob demanda, o desenvolvedor não precisa investir em compra, manutenção de servidores e *links* de *internet*. O foco será somente o produto, reduzindo drasticamente o tempo de desenvolvimento e o término do prazo de entrega.

2.2 PLATAFORMAS WEB

A plataforma *web* para os usuários é muito simples, praticamente todos os computadores já vem preparados para utilizar a *internet*, inclusive alguns

navegadores já são integrados ao próprio sistemas operacional. Para Taurion (2009), a utilização dos serviços em massa são consequência de serem fáceis de usar e muito convenientes. Acesso a *e-mails*, fotos, arquivos de texto em qualquer lugar, de qualquer computador, faz a diferença (TAURION, 2009).

Entre as grandes empresas que oferecem serviços de computação em nuvem podemos destacar a Microsoft, a Google e a Amazon. A Microsoft desenvolveu o Windows Azure, a Google disponibiliza suas APIs e sua infraestrutura, já a Amazon dispõe de máquinas virtuais e formas de armazenamento de dados.

2.2.1 Microsoft Windows Azure

O Windows Azure é um sistema operacional na *web*, é um serviço totalmente controlado pela Microsoft, é uma plataforma para desenvolvedores de aplicações para a *internet*, nele, empresas de desenvolvimento de *software* poderão disponibilizar seus aplicativos para usuários finais e pagando para a Microsoft a demanda dos recursos utilizados.

O Windows Azure foi desenvolvido para facilitar o desenvolvimento de aplicações *web*, ou seja, um sistema operacional para desenvolvedores de aplicações. Os provedores tradicionais apenas oferecem o espaço, sem contar ainda que nesses casos não existe escalabilidade no sentido de vários servidores dividindo a carga de processamento (BUFALOINFO, 2008). No Azure, o desenvolvedor conta ainda com diversos recursos que não estão presentes nos provedores tradicionais, como componentes que são utilizados para a comunicação entre processos por exemplo.

No Azure da Microsoft é possível utilizar ferramentas da própria empresa para o desenvolvimento das aplicações, como o Visual Studio e banco de dados MS SQL Server e utilizar outras linguagens "não - MS" como PHP, Ruby, IDE Eclipse e linguagem Java (GHEDIN, 2011). Já um concorrente da Microsoft para plataformas Cloud é o Google com suas APIs, e será discutido no próximo tópico.

2.2.2 Infraestrutura Google e Google APP

Google App é um serviço Google que apresenta muitas aplicações via *internet* com funcionamento similar aos tradicionais programas de escritório. Com o Google App é possível executar os aplicativos na própria infraestrutura da empresa, e dessa forma, não sendo necessário o cliente manter servidores e arquivos locais. Algumas das linguagens suportadas são o Java, o Javascript e Ruby (CODE.GOOGLE, 2011).

Como o Google App que utiliza a infraestrutura do Google, os serviços são escaláveis, podendo ser aumentados conforme a necessidade, outra característica do produto é que não é preciso pagar para poder utilizar os aplicativos e os serviços na sua configuração básica, e caso o cliente precise uma demanda maior, poderá adquirir sob medida.

No ambiente de execução Java, o aplicativo do desenvolvedor interage com o padrão Servlet Java e que pode utilizar tecnologias comuns para *web*. É possível também executar em um interpretador otimizado para Python, além de APIs avançadas e ferramentas de desenvolvimento de aplicativos, incluindo API de modelagem de dados, estrutura para aplicativos web, ferramentas para gerenciar e acessar os dados das aplicações (CODE.GOOGLE, 2011).

A Google detém algumas vantagens sobre os concorrentes, pois já disponibiliza vários serviços na web e muitos deles são integrados, como o email, redes sociais, buscadores. Conta ainda com inúmeros programadores colaboradores que utilizam as APIS para desenvolver os produtos mais diversificados. A autenticação é feita por contas Google, desse modo o desenvolvedor pode utilizar uma infraestrutura de Login da empresa para utilizar no seu aplicativo.

De acordo com Feigenbaum (2011), A Google se preocupa com a segurança do Google App e Google App Engine, recentemente recebeu a certificação SSAE-16, as auditorias são realizadas por empresas terceiras, garantindo assim a segurança de seus clientes (FEIGENBAUM, 2011).

Isso se torna necessário, pois muitos profissionais de TI ainda são receosos quanto á utilização da computação em nuvem, mais especificamente colocar todos os códigos fontes de suas aplicações em poder de terceiros, muitos softwares devem ser protegidos e seu código fonte tratado em segredo. Empresas que

oferecem computação em nuvem devem estar preparadas para fornecer segurança a seus clientes.

Caso os clientes não são desenvolvedoras de *software*, mas desejam migrar servidores para uma infraestrutura de computação em nuvem, a empresa indicada pode ser a Amazon. Nela é possível literalmente alugar um servidor e instalar praticamente qualquer software. No próximo capitulo será apresentado o produto da empresa Amazon.

2.2.3 Amazon EC2

A Amazon um grande empresa em vendas no varejo e agora aluga sua infraestrutura para terceiros. Segundo Taurion (2009), a empresa surgiu como líder no negócio de plataforma em nuvem e que pode futuramente ser mais conhecida por esses serviços que pelo próprio varejo. A Empresa descobriu que podia vender sua infraestrutura explorando novas formas de pagamento e com isso estimulou outras grandes empresas de tecnologia a investir nesses tipos de serviço (TAURION, 2009).

O principal produto da Amazon para a nuvem é o Amazon EC2. Este serviço oferece capacidade computacional redimensionável na nuvem, projetado para tornar mais fácil para os desenvolvedores. Amazon EC2 permite aumentar e diminuir a capacidade em minutos, e vários servidores podem ser instanciados simultaneamente por APIs (AMAZON, 2011).

A flexibilidade também é disponível para EC2, permite que o usuário selecione a configuração de memória, CPU, armazenamento e tráfego. Além disso, suporta instanciar diversos sistemas operacionais como Linux, Windows e Opensolar. A disponibilidade é garantida para 99,95% e ainda oferece segurança para as máquinas hospedas com ferramentas personalizadas, como é o caso de configurações de Firewall, controle de acessos e IPseg VPN criptografados (AMAZON, 2011).

Na Amazon, as máquinas virtuais são ambientes de servidor baseado em Linux e Windows e que podem executar praticamente qualquer software ou aplicativo, o suporte ao sistema operacional Opensolar também é suportado. Cada

máquina é uma instância de uma imagem, essas imagens são conhecidas como AMIs e podem ser instanciadas várias vezes quando é necessário mais desempenho (CHAGANTI, 2011).

As máquinas virtuais podem ser configuradas de acordo com a necessidade, se o usuário necessita de desempenho, então a instância pode conter mais de uma unidades de cálculo, caso seja necessário, o usuário pode optar por instanciar uma máquina com mais memória, tudo dinamicamente através de ferramentas fornecidas pela própria Amazon.

Além das instâncias de máquinas virtuais existem também suporte a armazenamento de informações através de bancos de dados, que são gerenciadores de volumes. Segundo Chaganti (2011), A Amazon dispõe uma forma de armazenamento EBS, que permite criar volumes que possam ser anexados como dispositivos em nível de bloco a uma instância em execução. Pode ser anexado vários volumes de dados para cada instância de máquina, observando que se deve anexar uma de cada vez (CHAGANTI, 2011).

2.3 MODELO SAAS

O conceito de Software como Serviço com a sigla *SaaS*, é a comercialização do *software*, onde o usuário paga somente pelo que usa. Dessa forma o serviço fica disponibilizado na web e o cliente não precisa se preocupar em montar uma infraestrutura própria para a sua necessidade.

Para Pozzebon (2011), para o modelo de *SaaS*, a única infraestrutura necessária para o cliente é seu próprio computador e uma conexão com a *internet*, o cliente não precisa se preocupar com atualizações do *software* adquirido. As demais estruturas como a aplicação, plataforma e infraestrutura serão por conta do fornecedor (POZZEBON, 2011).

Estes serviços justamente por ser vendidos a terceiros e considerados serviços críticos, devem ter alta disponibilidade e capacidade de atender demandas maiores de serviço, além de segurança das informações. Características necessárias para empresas que desejam implementar serviços desse tipo.

Empresas desenvolvedoras de *softwares* para o modelo *SaaS* devem considerar os recursos necessários, otimizando o processo, baixando o custo. Na Figura 1, a figura mostra o modelo de maturidade *SaaS*, sobre a evolução do suporte multi-inquilo.

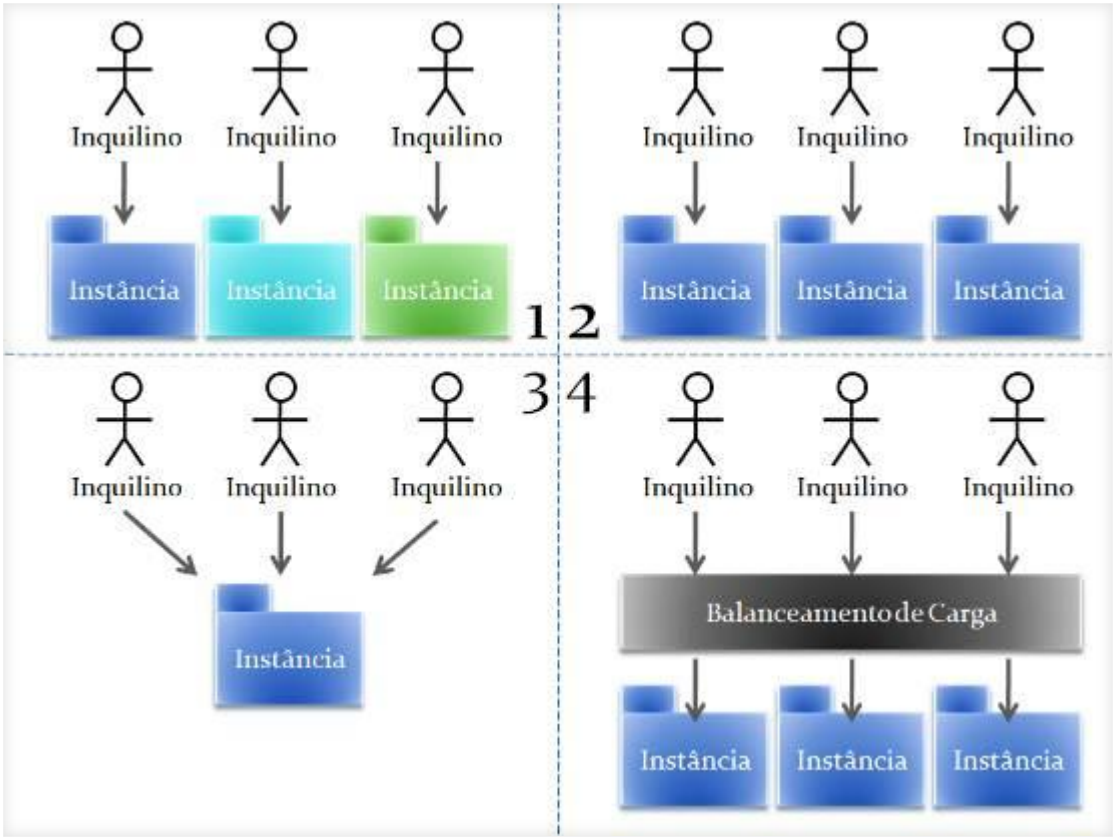


Figura 1 – Modelo de maturidade SaaS
Fonte: Adaptado de MSDN.MICROSOFT (2011)

Na Figura 1, primeiro quadrante, a figura demonstra uma instância da solução *SaaS* para cada cliente, isso garante a total atendimento ao cliente, porém como não tem compartilhamento dos recursos, é necessário maior demanda e torna-se o custo mais elevado. Além disso, a customização é maior, pois é necessário configuração individual para cada cliente (MSDN.MICROSOFT, 2011).

O segundo quadrante da Figura 1, mostra que ainda apresenta uma instância da aplicação para cada cliente, mostra ainda, uma melhora em relação ao quadrante um, pois nesse tipo de configuração não existe uma customização para cliente, facilitando a inserção de novos clientes, já que as configurações serão as mesmas.

O quadrante três da Figura 1, a figura mostra que uma única instância para múltiplos clientes. Existem uma melhora em relação aos recursos dos quadrantes anteriores, pois apresenta um completo compartilhamento de recursos. É ainda, mais complicada para ser implementada, pois na sua construção é necessário modelar o banco de dados para multiusuários.

No último quadrante da Figura 1, permite o atendimento diferenciado para clientes que necessitam de elevada demanda de recursos, onde a aplicação *SaaS* permite balanciar à carga necessária entre as várias instâncias.

Cada qual das alternativas mostradas na figura 1 tem suas vantagens e desvantagens, o fornecedor pode elaborar um produto padrão ou mesmo personalizado para seu cliente, utilizar ainda um modelo que faz balanceamento de carga se for necessário.

Migrar para o mundo *SaaS* não será uma transição fácil para empresas que não tem grandes capacidades de investimento, por outro lado será propício para as novas empresas que buscam entrar no mercado. As antigas irão acelerar seus investimentos nesse modelo, ampliando a oferta de serviços (TAURION, 2009).

Grandes empresas de softwares corporativos como a SAP, utilizam-se de práticas de aquisição de empresas menores que já estão nesse tipo de mercado. Adquirindo empresas menores entram no mercado rapidamente, pois adquirem os produtos e também o capital intelectual das empresas das empresas adquiridas (TAURION, 2009).

2.3.1 Software convencional X modelo SaaS

A diferença entre o antigo modelo de licenças para o modelo de negócios *SaaS*, é que a lucratividade no modelo antigo vem das taxas de manutenção e prestação de serviço. Nesse modelo o cliente fica amarado ao fornecedor, justamente pelos custos envolvidos. No modelo *SaaS*, a receita vem de assinaturas que até pode se assimilar aos modelos antigos, porém a forma de cobrança pode ser diferente, em alguns casos os valores das licenças podem ser irrisórios e o aplicativo incentivar o uso de propagandas, que se torna a principal fonte de renda da empresa desenvolvedora.

2.4 MODELO S+S MICROSOFT

O S+S (Software mais serviço) é um conceito disseminado pela Microsoft e segundo ela, é uma melhoria sobre o modelo *SaaS*. Enquanto o *SaaS* é disponibilizado somente pela *internet*, na filosofia da Microsoft, o S+S traz a experiência enriquecida como o uso de recursos locais. Nesse ponto, é possível compor funcionalidade diferentes, criando um mix de recursos como melhor das tecnologias web, móbile, desktop (ARTIGOSONLINE, 2011).

Este é um fato que pode gerar muitas discussões ainda, pois enquanto a empresa Google prega que todos os serviços devem rodar na nuvem, exemplo disso seria o Google Docs e Google Apps, as respostas da Microsoft vão a encontro de outras filosofias. A Microsoft defende uma abordagem Híbrida onde ela retém a capacidade de computação e software no próprio dispositivo do usuário.

2.5 WEB SERVICE NAS ORGANIZAÇÕES

As empresas geralmente empregam diversos aplicativos, muitos são sistemas legados e específicos, desenvolvidos nas mais diversas linguagens de programação. Com a popularização da *internet* e a demanda crescente por consumo de informações por parte das organizações, verificou-se a necessidade de compartilhamento das informações e integração entre os sistemas.

O *Web service* surge para contornar o problema e fazer com o que os diversos sistemas legados possam trocar informação entre si, com essa tecnologia é possível efetuas a troca de dados entre os novos aplicativos e os sistemas legados. A comunicação é feita sobre a camada de rede, sendo possível desse modo á integração entre sistemas que não estão no mesmo local desde que possuam conexão entre eles.

A utilização de *Web services* automatiza o processo de troca de informações dentro da organização, trazendo eficiência e agilidade ao processo, já que são automatizadas, elimina a intervenção humana e reduz a chance de haver problemas e falhas. A utilização de vários *Web services* a fim de trabalhar como se fosse um único sistema é caracterizado como Sistemas Distribuídos.

Segundo Iweb (2003), Web services são componentes, ou unidades lógicas que disponibilizam protocolos e que podem ser acessados através da *internet*. O modo de acesso combinam as melhores práticas do desenvolvimento feito para a *web*, sendo assim, é necessário verificar alguns requisitos como: A forma comum de representar dados em mensagens, uma linguagem de descrição do serviço, mecanismo de descrição do serviço, um mecanismo para descobrir servidores de serviço (IWEB, 2003).

Algumas especificações pode ajudar no desenvolvimento de Web services, o SOAP é uma delas. O SOAP é uma especificação de protocolo que é composta de um conjunto de regras de como utilizar o XML, bem como representar os dados. É também utilizado para definir o formato das mensagens e convenções e para representar chamadas de protocolo remoto (RPC). A especificação SOAP é um híbrido entre aplicações RPC e aplicações por mensagens, suporta ambos os formatos, sendo bastante flexível (IWEB, 2003).

2.6 SISTEMAS DISTRIBUÍDOS

Atualmente, com o resultado dessas novas tecnologias, é possível de se ter milhares de computadores interligados por uma rede de alta velocidade, e que é denominado por Tanenbaum, como “um conjunto de computadores independentes que se apresenta a seus usuários com um sistema único e coerente” (TANENBAUM e STEEN, 2007).

A computação distribuída é de forma geral paralela e distribuída, onde mais de um computador é conectado através de uma rede para realizar uma tarefa comum. Sistemas distribuídos proporcionam o acesso a recursos remotos como se fossem locais. Desta forma, um usuário pode utilizar-se de impressoras, computadores e diversos periféricos da rede como se estivem conectados no próprio

computador, proporcionando grande economia de recursos de *hardware* e *software*. Na Figura 2, a figura representa os mais diferentes componentes que podem ser considerados em sistemas distribuídos.

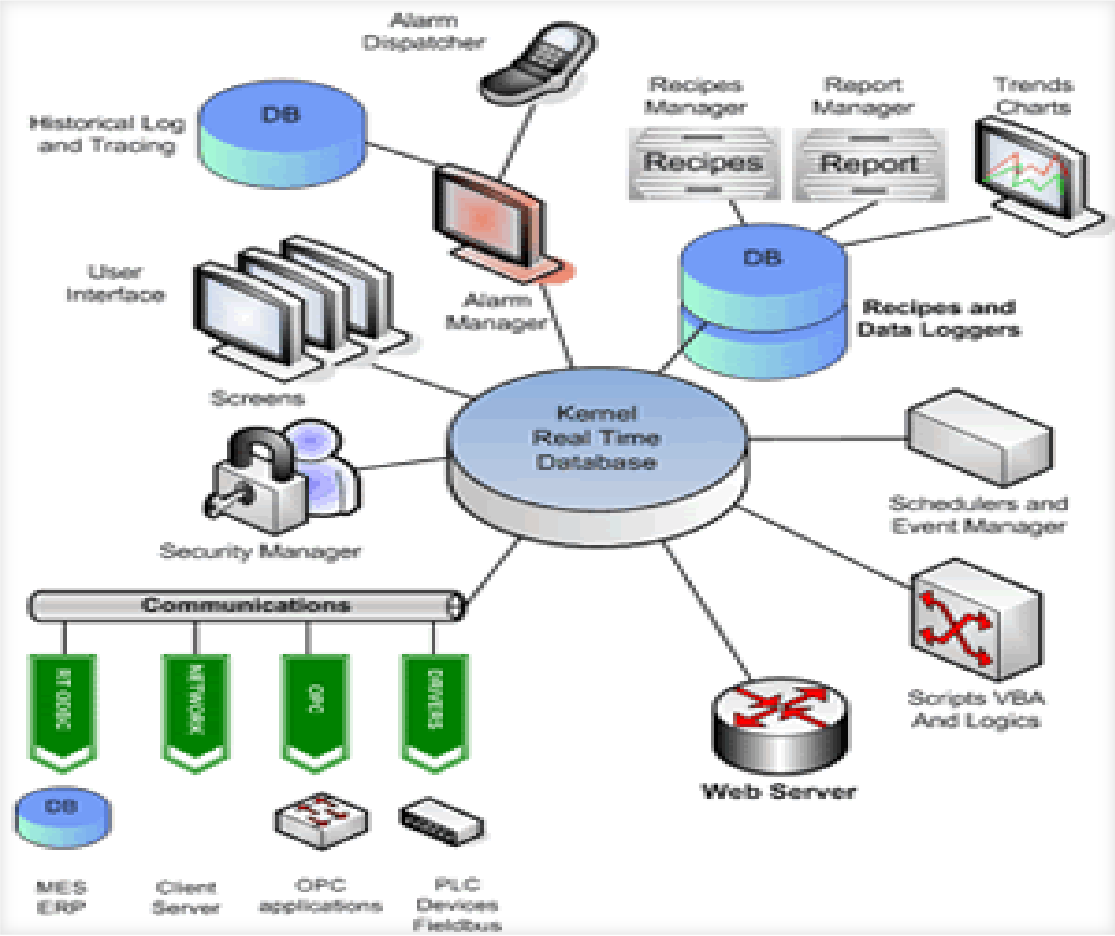


Figura 2 – Representação de Sistemas Distribuídos
Fonte: Adaptado de NOVIDADESDEINFORMATICA (2011)

A *internet* é um exemplo de sistemas distribuídos, onde, através de uma comunicação simples, é possível a troca dos mais diversos tipos de informações como música, arquivos, vídeo, entre os mais diferentes periféricos como: celulares, computadores e tablets (NOVIDADESDEINFORMATICA, 2011).

2.7 SISTEMAS DE COMPUTAÇÃO DISTRIBUÍDOS

Para Tanenbaum e Steen (2007), os sistemas distribuídos podem ser classificados em dois subgrupos:

Computação de cluster: Existe um conjunto de estações de trabalho ou PCs semelhantes, conectados por uma rede de alta velocidade. Onde cada um dessas máquinas executa o mesmo sistema operacional (TANENBAUM e STEEN, 2007).

Computação em grade: Nesse caso, os sistemas distribuídos costumam ser montados como federação de computadores, na qual cada sistema pode cair sobre um domínio administrativo diferente (TANENBAUM e STEEN, 2007).

2.7.1 Sistemas de computação de cluster

Os sistemas de computação em cluster, tornara-se viáveis quando a relação preço x desempenho diminuiu a tal ponto, de ser atraente, em termos técnicos e financeiros, a construção de supercomputadores a partir de várias estações de trabalho conectados entre por uma rede de alta velocidade (TANENBAUM e STEEN, 2007).

São denominados dessa forma quando existe mais de um computador ou sistemas que trabalham em conjunto para realizar tarefas, de tal forma que usuários tenham a impressão de ser um único computador virtual. São desenvolvidos para ter alta disponibilidade, balanceamento de carga e processamento distribuído (TANENBAUM e STEEN, 2007).

2.7.2 Sistemas de computação de grade

Os sistemas de computação em grade apresentam alto grau de heterogeneidade; os recursos de diferentes organizações são reunidos para permitir a colaboração de um grupo de pessoas, essa organização virtual tem direitos de acesso aos recursos fornecidos por aquela organização. A arquitetura prevê quatro camadas, divididas em camada-base, camada de conectividade, camada de recursos e camada de aplicação (TANENBAUM e STEEN, 2007).

2.8 COMUNICAÇÃO ENTRE SISTEMAS DISTRIBUÍDOS

A comunicação em sistemas distribuídos exige que softwares de diversas linguagens possam se comunicar. A comunicação pode ser por protocolo ou o canal de comunicação e deve conter informação que possa ser entendida por todas as máquinas. Desenvolvedores devem tomar certos cuidados para evitar mensagens inválidas e erros. Nesses casos, essas mensagens devem ser descartadas pelo servidor, evitando a sobrecarga da rede e a queda do servidor de aplicações.

O desenvolvimento de aplicações distribuídas é complexo e extremamente difícil, boa parte, por que o principal meio de conectividade é a *internet* e essa rede não é confiável. Dessa forma, foram desenvolvidos alguns protocolos de comunicação, onde os processos comunicantes tendem á seguir regras para garantir ou ainda minimizar os problemas de comunicação (TANENBAUM e STEEN, 2007).

Os três modelos amplamente utilizados para a comunicação entre sistemas distribuídos são respectivamente os protocolos: chamada de procedimento remoto (Remote Procedure Call – RPC), *middleware* orientado a mensagem (Message-Orientad Middleware – MOM) e fluxo de dados (TANENBAUM e STEEN, 2007).

2.8.1 Comunicação RPC

A chamada remota de procedimento é uma tecnologia popular para a implementação do modelo cliente-servidor da computação distribuída, onde a chamada é iniciada pelo cliente que emite uma mensagem para o servidor remoto, que executa o procedimento e retorna uma mensagem para o cliente.

O RPC foi criado com o intuito de suprir a necessidade de invocações remotas. O protocolo RPC implementa interfaces, que são rotinas *stubs* e *skeletons* para fazer a comunicação entre cliente/servidor. Rotinas *studs* que são implementadas e disponibilizadas no cliente, e quando chamadas, executam as skeletons que podem estar numa outra máquina da rede ou ainda em um servidor fora país (GUIADOHACKER, 2009).

A comunicação RPC foi proposta por Birrell e Nelson no ano de 1984, segundo os pesquisadores, a sugestão era de permitir que programas chamassem procedimentos localizados em outras máquinas. Neste contexto, um programa que é executado numa máquina **A** pode chamar um procedimento em outra máquina **B**, quando isso ocorre o processo que chamou fica suspenso até a outra máquina responder (TANENBAUM e STEEN, 2007).

A ideia fundamentada nesse modelo de comunicação é realizar uma chamada de procedimento remoto fazendo parecer que é local. Um exemplo que funcionará em uma máquina do servidor de arquivos remoto, em vez de chamar uma chamada convencional, uma versão diferente dessa função *read* é colocada na biblioteca, denominada apêndice de cliente. Esta por sua vez, faz a chamada ao sistema operacional local, só que diferentemente da chamada convencional, ela não pede que o sistema operacional lhe dê dados. Em vez disso, empacota os parâmetros em uma mensagem e requisita que essa mensagem seja enviada para o servidor, bloqueando a aplicação que chama o procedimento até que a resposta volte (TANENBAUM e STEEN, 2007).

Quando a mensagem retorna do servidor passa junto um apêndice de servidor, este é equivalente ao apêndice da máquina cliente: é um pedaço de código que transforma requisições que vêm pela rede em procedimentos locais, então são desempacotados os parâmetros da mensagem e logo após é chamada o procedimento do servidor de maneira usual. (TANENBAUM e STEEN, 2007).

O servidor sabe que o procedimento foi chamado pelo cliente, o servidor executa seu trabalho e então retorna o resultado para o cliente chamador de modo usual. O servidor enche o *buffer*, apontado pelo segundo parâmetro, com os dados. Após a conclusão da chamada, o servidor empacota o *buffer* em uma mensagem e chama “*send*” para retornar ao cliente. Depois disso chama o procedimento “*receiver*” e fica apto a executar uma próxima requisição de cliente. Já ao lado do cliente, o sistema operacional vê que está endereçado ao processo cliente. A mensagem é copiada para o *buffer* que está á espera e o processo então é desbloqueado, quando o chamador retorna o controle em seguida à chamada para “read”, os dados já estão disponíveis (TANENBAUM e STEEN, 2007).

Para Tanenbaum e Steen (2007), o resumo de chamada remota ocorre nas seguintes etapas;

- O procedimento de cliente chama o apêndice de cliente de modo normal.
- O apêndice de cliente constrói uma mensagem e chama o sistema operacional local.
- O SO do cliente envia mensagem para o SO remoto.
- O SO remoto dá a mensagem ao apêndice de servidor.
- O Apêndice de servidor desempacota os parâmetros e chama o servidor.
- O servidor faz o serviço e retorna o resultado do apêndice.
- O apêndice de servidor empacota o resultado em uma mensagem e chama seu SO local.
- O SO do servidor envia a mensagem ao SO do cliente.
- O SO do cliente dá a mensagem ao apêndice de cliente.
- O apêndice desempacota o resultado e retorna ao cliente.

O resultado nessas etapas é converter uma chamada local em remota sem que o cliente e nem o servidor saibam das etapas intermediarias ou dá existência de rede entre as máquinas.

2.8.1.1 RPC em ambiente de computação distribuída

Desenvolvido pela Open Software Foundation, chamada agora de Open Group, o DCE RPC é representativo de outros sistemas RPC e suas especificações foram adotadas como base para computação distribuída da Microsoft. O DCE é um sistema middleware que foi projetado executar uma camada de abstração, desse modo, através da instalação do *software* em um conjunto de máquinas é possível executar aplicações distribuídas. Em alguns casos é necessário ainda incluir uma parte do sistema de arquivo distribuído ao núcleo (TANENBAUM e STEEN, 2007).

O modelo cliente-servidor é o praticado no DCE, processos de usuários agem como clientes para acessar serviços remotos fornecidos pelo servidor. Esses serviços podem fazer parte do próprio DCE ou ainda aplicações escritas por programadores de aplicações. Vários serviços fazem parte do DCE, entre eles; serviço de arquivo distribuído, serviço de diretório, serviço de segurança e serviço distribuído de horário (TANENBAUM e STEEN, 2007).

2.8.1.2 Objetivos do DCE RPC

O sistema DCE RPC permite, a execução de serviços remotos por meio de uma simples chamada local. Deste modo, aplicações clientes são escritas de modo simples, e não tão complexas que podem ser entendidas pela maioria dos programadores e também facilita a execução de grandes volumes de código com o mínimo de alterações de adaptação (TANENBAUM e STEEN, 2007).

O sistema DCE RPC contribui pelo alto grau de independência entre aplicações clientes e servidoras, pois esconde detalhes entre as aplicações, sendo assim possível utilizar diferentes linguagens de programação. O sistema DCE RPC vincula automaticamente o servidor correto da aplicação cliente, faz automaticamente conversões de tipos de dados entre as aplicações, reforçando a ideia de sistemas independentes.

Utiliza-se a linguagem de definição de *interfaces*, para permitir fazer declarações de procedimentos, definições de tipos de dados, constantes e toda a informação necessária para montar parâmetros e desmontar resultados corretamente (TANENBAUM e STEEN, 2007).

Para a validação do servidor correto, ou ainda para verificar-se que o método não é defasado, é utilizado para controle um identificador global. Deste modo a primeira etapa de se criar uma aplicação cliente-servidor é chamar o programa para que ele gere um protótipo de arquivo IDL que contenha o identificador de *interface*. O identificador consiste em um número grande de 128 bits, utilizado para garantir a exclusividade do método (TANENBAUM e STEEN, 2007).

A segunda etapa para a criação de uma aplicação cliente-servidor consiste em editar o arquivo de *interface* com nomes de procedimentos e seus parâmetros. Na terceira etapa o arquivo IDL é compilado gerando assim, um arquivo de cabeçalho; O apêndice do cliente e o apêndice do servidor.

No arquivo de cabeçalho contém o identificador exclusivo, definições de tipos e constantes, protótipos e funções. Os apêndices são os procedimentos que serão chamados pelo cliente, e são responsáveis por colher e empacotar os parâmetros na mensagem de saída e depois chamar o sistema de execução para enviá-los.

2.8.2 Comunicação orientada a mensagem

As chamadas de procedimento ajudam a ocultar a comunicação entre sistemas distribuídos, fazendo com a tarefa de comunicação seja mais simples para o desenvolvedor. Porém em alguns casos, esse protocolo de comunicação não é o mais indicado, quando isso ocorre pode-se utilizar a comunicação orientada a mensagem.

Sistemas de enfileiramento de mensagens, ou simplesmente *middleware* orientado a mensagem, podem comunicar de modo assíncrono e persistente, esses sistemas oferecem suporte para armazenamento de mensagens e médio e curto prazo e não necessitam que o remetente e o receptor estejam ativos para que a mensagem chegue ao seu destino, os sistemas de email são um exemplo desse tipo de serviço (TANENBAUM e STEEN, 2007).

As mensagens em uma comunicação orientada a mensagem são enfileiradas em filas específicas e em algum momento são entregues para o destino final. Essa abordagem permite que a comunicação seja fracamente acoplada em relação ao tempo, e enquanto isso tanto o remetente, quanto o receptor podem executar em completa independência com relação ao outro (TANENBAUM e STEEN, 2007).

O canal de comunicação é bastante flexível, em geral pode se passar quaisquer dados entre a origem e o destino. Em alguns casos a mensagem pode ser grandes fragmentos que são passados de modo transparente para as aplicações. A *interface* básica pode ser extremamente simples, onde quatro primitivas; “*Put*”, “*Get*”, “*Poll*” e “*Notify*” permitem se criar uma comunicação em sistemas distribuídos.

A primitiva “*Put*” é chamada pelo remetente a fim de passar uma mensagem para destinatário, essa mensagem é anexada á fila especificada, está primitiva não é bloqueadora, logo ela pode enviar outra mensagem para a fila. A primitiva “*Get*” é responsável por retirar o item mais antigo da fila, algumas versões desse comando permite que se busque uma mensagem específica dentro da fila. A primitiva “*Poll*” faz o mesmo que o “*Get*” porem não bloqueia e caso a fila esteja vazia, o processo chamador continua (TANENBAUM e STEEN, 2007).

2.8.3 Comunicação orientada a fluxo

Enquanto a comunicação por mensagem envia a mensagem e o tempo de envio pode ser em médio prazo, na comunicação orientada a fluxo a relação tempo é crucial. Exemplo de desse tipo de comunicação é um fluxo de áudio que deve ser reproduzido em intervalos de tempo, mensagens reproduzidas mais rapidamente ou mais lentamente resultarão em uma versão incorreta do som original (TANENBAUM e STEEN, 2007).

Para Tanenbaum e Steen (2007), para garantir a qualidade do serviço é necessário assegurar que alguns requisitos sejam respeitados, por exemplo, as relações temporais em um fluxo (TANENBAUM e STEEN, 2007). Para Tanenbaum e Steen (2007), da perspectiva de aplicação algumas propriedades são importantes para não comprometer a comunicação:

- A taxa de bits requerida à qual os dados devem ser transportados.
- O máximo atraso até o estabelecimento de uma sessão, quando uma aplicação pode começar a enviar dados.
- O máximo de atraso fim a fim, isto é, quanto tempo levará até que uma unidade de dados chegue a um receptor.
- A máxima variância de atraso, ou vibração.
- O máximo atraso de viagem de ida e volta.

2.8.4 Comunicação CORBA

Em meio à evolução das linguagens de programação e a adoção de paradigmas como orientação a objetos, o protocolo RPC também evoluiu, surgindo então o CORBA, que é uma evolução que detém melhorias em relação ao seu antecessor como um conjunto de serviços para facilitar o desenvolvimento.

No CORBA é possível registrar objetos por nome, e que também implementa dois conceitos importantes; uma biblioteca para fazer o gerenciamento de portas da rede e o *garbage collection*. Além disso, no CORBA é possível fazer com que a o servidor possa se comportar como cliente e o cliente como servidor, dependendo da situação, essa melhoria não está presente no RPC (CERQUEIRA, 2004).

A Figura 3 demonstra a Arquitetura de referência da OMG.

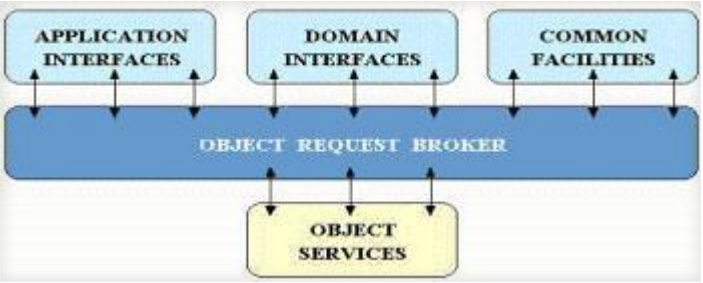


Figura 3 – Arquitetura de modelo de referência da OMG
Fonte: Adaptado de Cerqueira (2004)

Common Object Request Brocker Architecture (CORBA), trata-se de um padrão proposto pela organização internacional da indústria de *software* (OMG) que estabeleceu uma estrutura para o gerenciamento de objetos distribuídos. Para cada sistema de rede, CORBA permite que seja definida uma IDL (Linguagem de *Interface*). Estas Interfaces descrevem os serviços fornecidos e que quando compiladas geram o stubs e o skeletons (CERQUEIRA, 2004).

2.8.5 Comunicação COM e DCOM

A Microsoft também desenvolveu uma tecnologia para a criação de componentes de *software* distribuídos em computadores interligados em rede,

DCOM (*Distributed Component Object Model*) é uma extensão do COM da própria Microsoft. A tecnologia foi substituída na plataforma .NET pela API .NET Remoting.

A comunicação entre aplicações clientes e servidoras é feita através de *interfaces*. Na *interface* COM é descrito os argumentos necessários para fazer a troca de informações entre os sistemas. Um GUID (números de 128 bits) é um número utilizado para identificar uma *interface* ou uma *Co-Class*. Já o mecanismo que permite a comunicação entre aplicações de diferentes máquinas em uma rede chama-se DCOM (BECKER, 1999)

DCOM estende a tecnologia COM permitindo suportar comunicação entre objetos em computadores diferentes de forma transparente, tanto em uma rede Local quanto computadores que utilizam a *internet*. As linguagens de programação como C++, Java, Object Pascal, Visual Basic são algumas linguagens que dão suporte para se codificar objetos DCOM (BECKER, 1999).

2.8.6 Comunicação RMI

O RMI (*Remote Method Invocation*) é a uma interface de comunicação no estilo RPC para aplicações desenvolvidas em Java, essa interface de programação permite que um objeto Java possa interagir com outros Objetos de outras máquinas virtuais Java, eliminando a preocupação com detalhes de comunicação para o programador e deixa a responsabilidade para a API.

Cliente e servidor são respectivamente os dois programas que se comunicam. O servidor instância objetos e os publica através de BIND numa porta do protocolo TCP onde fica a espera que clientes possam invocá-los. O cliente por sua vez, faz referência aos métodos de um objeto remoto, deixando o RMI fazer a comunicação entre cliente e servidor. Essa denominação é conhecida como Aplicações com Objetos Distribuídos.

Para Spinola (2011), a vantagem da utilização de RMI para Java em relação a outros sistemas é o fato de que com RMI é possível utilizar qualquer tipo de objeto Java na JVM local ou remota. Mesmo em casos que o cliente não tenha conhecimento sobre o objeto é possível utilizar da tecnologia, RMI permite inclusive carregar novas classes dinamicamente, recurso altamente interessante e bem vindo.

Para Destro (2003), A tecnologia RMI está presente no Java desde sua versão 1.1, e apesar de ser relativamente ser fácil de ser utilizado, ele envolve o programador frente a um novo paradigma, o mundo dos objetos distribuídos. Sendo que o principal objetivo dos criadores da tecnologia é que os programadores Java desenvolverem programas distribuídos da mesma forma que os programas não distribuídos (DESTRO, 2003).

Para Destro (2003), a implementação da tecnologia RMI é feita em três camadas de abstração, as camadas *Stub* e *Skeleton* estão abaixo dos olhos dos programadores, nesta camada é responsável por interceptar chamadas de métodos feitas pelo cliente e as direcionam para o serviço RMI remoto. A segunda camada (*Remote Reference Layer*) sabe como interceptar e gerenciar as referências dos clientes para o serviço remoto em uma comunicação de um-para-um. A camada de transporte é a terceira e é baseada em conexões TCP/IP entre as máquinas da rede (DESTRO, 2003).

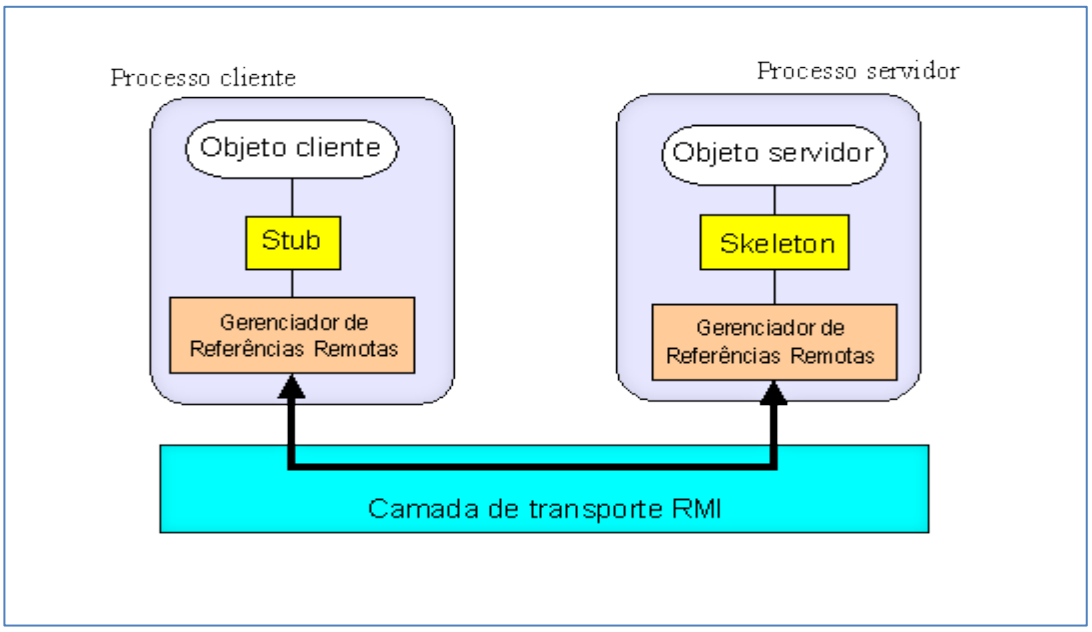


Figura 4 – Arquitetura de camadas de RMI
Fonte: Adaptado de UNICAMP (2002)

A Figura 4 ilustra a organização dessas três camadas em uma aplicação RMI, onde:

- A camada de *stub/skeleton* oferece as *interfaces* que os objetos da aplicação usam para interagir entre si;
- A camada de referência remota é o *middleware* entre a camada de *stub/skeleton* e o protocolo de transporte. Nesta camada são criadas e gerenciadas as referências remotas aos objetos;
- A camada do protocolo de transporte oferece o protocolo de dados binários que envia as solicitações aos objetos remotos pela rede (UNICAMP, 2002).

Para Destro (2003), As aplicações clientes acham os servidores que implementa o serviço usando um serviço de nomeação ou diretório, que roda como um endereço bem formatado (*host:port*) (DESTRO, 2003). O RMI também inclui um serviço chamado RMI Registry que roda em cada máquina que hospeda o serviço remoto. O servidor cria um serviço remoto e depois exporta para aquele objeto para RMI, logo após de exportá-lo aguarda as conexões dos clientes (TANENBAUM e STEEN, 2007).

Ao lado do cliente, o RMI Registry é acessado através da classe estática Naming. Está classe implementa o método *lookup()*, que o cliente utiliza para requisitar o registro, este método aceita a URL que especifica o nome do servidor e o nome do serviço desejado (DESTRO, 2003).

2.8.7 Entreprise JavaBeans

De acordo com Rafael (2010), EJB é um dos principais componentes da plataforma Java dois Entrerprise Edition, desenvolvida pela antiga Sun Microsystems e que tem por objetivo fornecer um modo simplificado de desenvolver aplicações Java baseado em componentes distribuídos, transacionais, seguras e portáveis. Como o EJB é possível criar aplicações modulares, utilizando conceito de componentes, das aplicações servidoras (RAFAEL, 2010).

Para Spínola (2010), A especificação foi introduzida no ano de 1998 como o objetivo de tornar mais fácil a confecção de sistemas distribuídos e orientados a objetos. A especificação e os servidores de aplicação tiveram sucesso em atingir seus objetivos, porém, falhas no EJB e algumas dificuldades fizeram com que fossem questionados se o EJB oferece a melhor produtividade para aplicações empresarias (SPÍNOLA, 2010).

O EJB utiliza o conceito de Beans, que na prática são vários objetos agrupados em um simples objeto, que devem ser implementadas de forma que apresente algumas restrições; quanto não ter um método construtor, não dever ter nenhum método de tratamento de eventos, todas as propriedades devem estar acessíveis e ainda que possa ser salvo e carregado posteriormente (SPÍNOLA, 2010).

Para (ORACLE, 2012), O Enterprise JavaBeans é executado em um container EJB, num ambiente de execução dentro do servidor de aplicações.

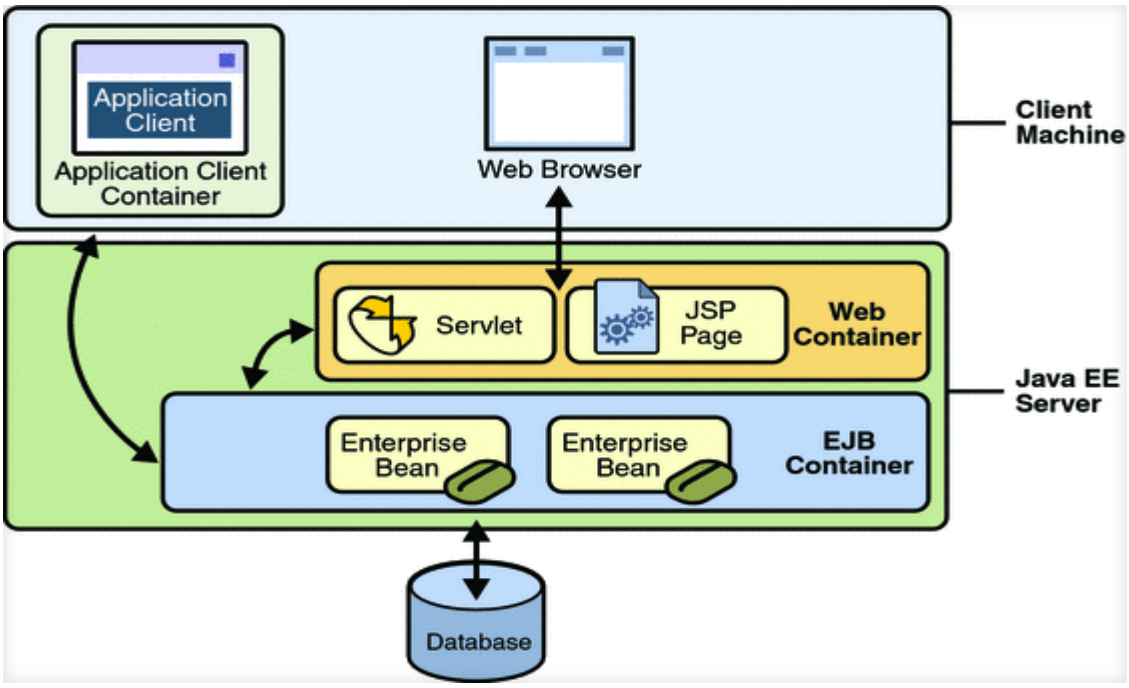


Figura 5 – Java EE Server e Containers
Fonte: Adaptado de ORACLE (2012)

Na Figura 5, é possível verificar o funcionamento do Enterprise Beans com EJB *container*, rodando como servidor de aplicação. Para Infoblogs (2007), os benefícios de se utilizar EJB em relação à utilização de classes Java são:

- EJB é bom se utilizar quando você quer uma aplicação totalmente distribuída.
- Há um grande proveito dos serviços oferecido pelos *middlewares* como: *pool*, transações, segurança, persistência gerenciada pelo *container*, cache entre outros.
- Utilizar o mesmo EJB em diferentes sistemas através do reuso, chamando remotamente.
- Se você precisa da proteção da sua lógica de negócios você pode fazer o *deploy* do seu *Web server* e do seu *aplication server* em máquinas separadas e colocar um *firewall* no meio (INFOBLOGS, 2007).

3 PROCEDIMENTOS METODOLÓGICOS

Para complementar a pesquisa bibliográfica será desenvolvida aplicações *Web services* que disponibilizam a comunicação RPC. Às aplicações servidoras disponibilizaram procedimentos que serão acessíveis às aplicações clientes. Esses procedimentos serão utilizados pelas aplicações clientes para salvar e recuperar dados XML no servidor.

3.1 OBJETIVOS DAS APLICAÇÕES

A construção da aplicação *Web service* de Armazenamento tem a finalidade de guardar e retornar dados XML que são enviados pelas aplicações clientes, e a implementação de um servidor RPC para poder se comunicar com aplicações clientes. Esta aplicação *web* vai ser implementada através da linguagem JRuby.

A construção do Web service de Gráficos tem a finalidade de gerar gráficos a partir de dados XML que o cliente envia. A comunicação entre cliente e servidor será o RestFull, está comunicação é o padrão de aplicações *Web services* implementadas na linguagem Ruby on Rails.

O aplicativo *desktop* desenvolvido em Delphi será o cliente que se comunica com o *Web service* de Armazenamento e Recuperação de XML e o *Web service* de Gráficos. A aplicação *desktop* implementa cliente RPC e utiliza componentes *indy* para se comunicar com os *Web services* distribuídos.

3.2 REQUISITOS FUNCIONAIS

A aplicação cliente desenvolvida em Delphi compreende os seguintes requisitos funcionais:

RF01: Validar informações de Data, Título e Valor

RF02: Gerar o XML com os dados inseridos

RF03: Enviar o XML ao *Web service* de Armazenamento

RF04: Gerar novo XML com os dados retornados do *Web service* de Armazenamento

RF05: Enviar o XML gerado para o *Web service* de Gráficos

RF06: Mostrar o Gráfico gerado

O *Web service* de armazenamento de arquivos XML compreende os seguintes requisitos funcionais:

RF07: Salvar o XML no DB Berkeley

RF08: Recuperar o XML através de uma consulta e enviar ao cliente

RF09: Criar *container* para armazenamento de XML

O *Web service* de armazenamento de arquivos XML compreende os seguintes requisitos funcionais:

RF10: Receber o XML do Cliente

RF11: Gerar o Gráfico com o XML recebido

3.3 REQUISITOS NÃO FUNCIONAIS

A aplicação cliente desenvolvida em Delphi compreende os seguintes requisitos não funcionais:

RNF01: A aplicação cliente desenvolvida em Delphi não utilizará arquivos de configuração ou conexão com base de dados locais

RNF02: O aplicativo Desktop deve ter acesso à *internet*

RNF03: Deve ser informado o endereço do *Web server* de Armazenamento

RNF02: A seleção do tipo de gráfico pode ser feita através do aplicativo cliente

3.4 DEFINIÇÃO DOS ATORES

Os seguintes atores estão presentes na aplicação desenvolvida em Delphi.

Figura 6 – Atores do sistema

AtorUsuario: Qualquer usuário que utiliza o aplicativo *desktop* para inserir dados

AtorSistema: O aplicativo *desktop* se comporta como um Ator quando necessita se comunicar com os *Web services* de Armazenamento e de Gráficos

3.5 LISTAS DE CASOS DE USO

Para diminuir a complexidade do sistema e facilitar o seu desenvolvimento, faz-se o seu particionamento em casos de uso. Esta técnica tem-se mostrado eficiente para organizar e direcionar todo o desenvolvimento do sistema. Um caso de uso pode ser sinalizado por um fluxo de dados, por uma condição, ou pelo tempo.

Casos de uso são tipicamente relacionados a atores. O ator é um humano ou uma máquina e o sistema, descrevendo como o software deverá se comportar e como deve ser construído.

Casos de uso de compõem as aplicações desenvolvidas:

Nr.	Descrição do Caso de Uso	Entrada	Nome	Resposta
01	Usuário lança Marco	DadosMarco	LancarMarco	Msg01
02	Usuário gera Gráfico	DadosGrafico	GerarGrafico	Msg02, VisualizaGrafico
03	Sistema Solicita Consulta	DadosSolicitacao	SolicitarXML	Msg03, DadosRetornados
04	Sistema Envia dados	DadosXML	EnviarXML	Msg04, GraficoGerado
05	Cadastrar Container	DadosContainer	CriarContainer	Msg05

Observação:

Msg01 = Marco lançado, Problemas com a Comunicação com *Web service* de armazenamento

Msg02 = Gráfico Renderizado, Problemas com Comunicação com *Web service* de gráficos

Msg03 = Dados retornados, Não existe dados para Montar XML

Msg04 = Gráfico Gerado, Não foi possível gerar o Gráfico

Msg05 = Container Criado, Problemas ao Criar Container

3.5.1 Diagrama de casos de Uso GerarGrafico e LancarMarco

Um diagrama de caso de uso descreve um cenário que mostra as funcionalidades do ponto de vista do usuário. É um classificador representando uma unidade funcional provida pelo sistema, manifestadas por sequências de mensagens intercambiáveis entre sistemas e um ou mais atores.

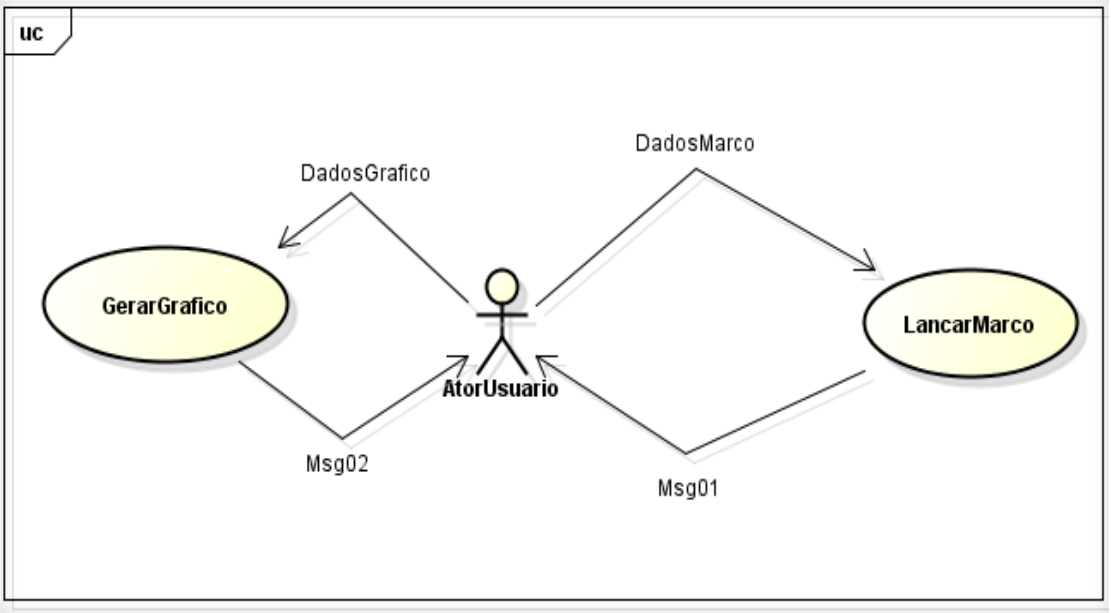


Figura 7 – Diagrama de casos de uso Lançar Marco

A Figura 7 representa o caso de uso para Lançar Marcos, Marcos são as marcações no sistema são os dados de algum valor em um determinado momento. O Usuário é responsável pelo lançamento de informações no sistema, e se necessário, gerar o Gráfico para visualizar as informações.

3.5.2 Diagrama de casos de Uso SolicitaXML e EnviaXML

O Ator Sistema descrito na Figura 8 sugere que o aplicativo *desktop* é um ator e que ele conversa com outros sistemas.

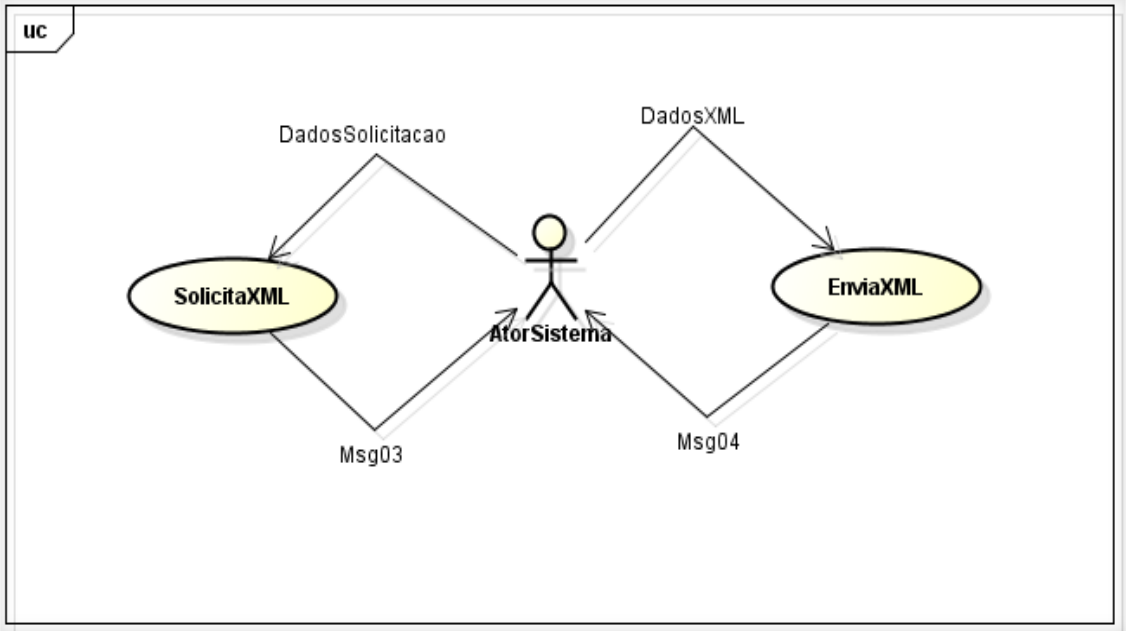


Figura 8 – Diagrama de casos de uso Solicitar XML e enviar XML

O aplicativo *desktop* atua como ator por que implementa um protocolo de comunicação cliente de XML RPC, dessa forma ele se comunica com o *Web service* de Armazenamento para solicitar consultas que resultam em dados de XML. Este ator não é considerado primário, pois não é ele quem desencadeia as trocas de mensagens entre os sistemas. O Ator Usuário é quem decide visualizar o Gráfico e por consequência desencadeia o processo.

3.5.3 Diagrama de caso de Uso CriaContainer

O Ator Usuário pode criar inúmeros *containers* de dados para armazenamento de arquivos XML. O diagrama de caso de uso CriarContainer é mostrado na Figura 9.

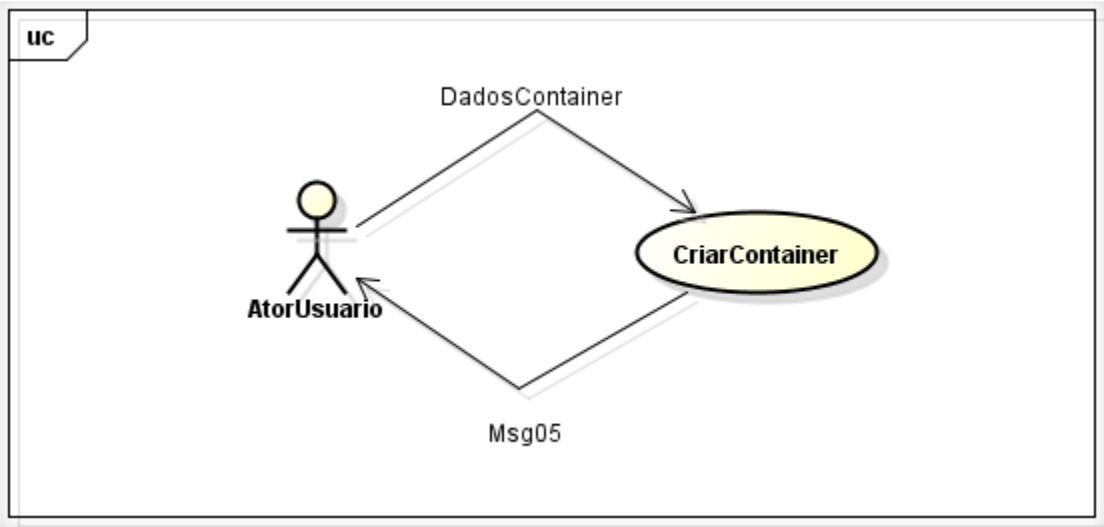


Figura 9 – Diagrama de casos de uso Criar Container **O**

O Ator usuário é responsável pelo cadastro de *containers* que serão utilizados no *Web service* de Armazenamento. O procedimento que cria novos *containers* no servidor aceita apenas um parâmetro de entrada com a descrição do *container*. Não é permitido cadastrar Marcos sem ter *container* definido anteriormente.

3.5.4 Caso de Uso Lançar Marco

Caso de Uso	LancarMarco – DC1
Atores	AtorUsuario (iniciador)
Finalidade	Lança Marco com informações de Dada, Titulo e Valor
Visão Geral	O usuário que ter uma marcação de determinado momento com um valor e uma descrição do tipo de marcação
Tipo	Primário
Pré-Condições	É necessário haver container de dados criado
Sequência Típica de Eventos	
Ação do Ator	***Resposta do Sistema***
1 – Usuário informa que deseja lançar novo Marco	
2 – Usuário informa da Data, a descrição do Marco e o valor	
3 – Usuário confirma a Marcação	4 – O aplicativo *desktop* válida as informações, são todas obrigatórias.
	5 – Sistema converte para dados XML
	6 – Envia os dados para *Web service* de Armazenamento
	7 – Sistema exibe mensagem de sucesso ao receber retorno de sucesso do *Web Service* de armazenamento
Exceções	
4 – Existe alguma informação inválida 4.1 Sistema emite mensagem dizendo qual informação é necessária 4.2 Sistema aborta e não tenta gerar os dados para XML 6 – Aplicativo não tem acesso a *internet* 6.1 Sistema emite mensagem que aplicativo não dispõe de *internet* 6.2 Sistema cancela envio do XML para o *Web service* de Armazenamento 7 – O *Web service* de armazenamento não localiza o *container* de dados 7.1 – O Sistema emite mensagem que não encontrou *container* de dados 7.2 – O Sistema cancela o armazenamento dos dados	

3.5.5 Caso de Uso Gerar Gráfico

Caso de Uso	GerarGrafico – DC2
Atores	AtorUsuario (iniciador)
Finalidade	Gerar o Gráfico baseado nas informações de Marcos lançados
Visão Geral	Após o Usuário lançar os marcos é possível gerar o Gráfico demonstrando dos resultados
Tipo	Primário
Pré-Condições	É necessário haver Marcos lançados
Sequência Típica de Eventos	
Ação do Ator	***Resposta do Sistema***
1 – Usuário informa que deseja visualizar gráfico	2 – Aplicativo *desktop* solicita consulta para Web Service de armazenamento buscando todos os marcos lançados
	3 - *Web service* de armazenamento faz buscas na base de dados e retorna arquivos XML para aplicativo *desktop*
	4 - Aplicativo *desktop* recebe arquivos XML e gera um novo XML juntando todos os resultados
	5 - Aplicativo *desktop* envia XML que produziu para *Web service* de Gráficos
	6 - *Web service* de Gráficos processa arquivo XML e retorna Gráfico
	7 - Aplicativo *desktop* recebe Gráfico e Imprime para usuário
Exceções	
2 – Aplicativo não tem acesso a *internet* 2.1 Sistema emite mensagem que aplicativo não dispõe de *internet* 2.2 Sistema cancela a requisição de consulta para o *Web service* de Armazenamento 4 – *Web service* de Armazenamento não retorna resultados 4 - Aplicativo *desktop* mostra mensagem que não existe resultados 4 – Aplicativo aborta e não tenta conectar a *Web service* de Gráficos 5 – Aplicativo não tem acesso a *internet* 5.1 Sistema emite mensagem que aplicativo não dispõe de *internet* 5.2 Sistema cancela o envio de dados XML para *Web service* de Gráficos	

3.5.6 Caso de Uso Criar Container

Caso de Uso	CriarContainer – DC3
Atores	AtorUsuario (iniciador)
Finalidade	Criar Container para armazenamento de dados no *Web service* de Armazenamento
Visão Geral	O usuário pode utilizar a aplicação para qualquer fim, deixando o usuário criar a base de dados, o usuário pode gerar Gráfico com as informações que queira somente.
Tipo	Primário
Pré-Condições	Não é necessário tem dados cadastrados anteriormente
Sequência Típica de Eventos	
Ação do Ator	***Resposta do Sistema***
1 – Usuário informa que deseja cadastrar um *container* de dados novo	2 – Aplicativo desktop solicita ao Web service de armazenamento para criar uma novo *container*, passando o nome por parâmetro.
	3 – O *Web service* de armazenamento cria o *container*
	4 – o Aplicativo recebe a mensagem de sucesso e imprime para o usuário
Exceções	
2 – Aplicativo não tem acesso a *internet* 2.1 Sistema emite mensagem que aplicativo não dispõe de *internet* 2.2 Sistema cancela a requisição de consulta para o *Web service* de armazenamento 3 – Não é informado o nome do *container* 3.1 – O aplicativo mostra mensagem que é necessário informar o nome do *container* 3.2 – O sistema aborta comando 4 – *Web service* de Armazenamento não retorna resultados 4 - Aplicativo *desktop* mostra mensagem que não existe resultados 4 – Aplicativo aborta e não tenta conectar a *Web service* de Gráficos 5 – Já existe um *container* com o nome passado por parâmetro 5.1 Sistema emite mensagem dizendo que *container* já existe 5.2 O sistema aborta comando	

4 RESULTADOS E DISCUSSÕES

A pesquisa bibliográfica mostrou que as diferentes formas de comunicação entre sistemas distribuídos, entre elas: A Comunicação por chamada de procedimento remoto, A Comunicação por mensagens e Comunicação orientada a fluxo. Utilizou-se a comunicação de chamada de procedimento remoto no desenvolvimento das aplicações *Web service* e *desktop*.

O usuário cadastra informações no aplicativo *desktop* e essas informações são convertidas em dados XML e então enviados ao *Web service* de armazenamento. O aplicativo *desktop* faz consultas e o *Web Service* de armazenamento resulta em dados XML que são convertidos em novos dados XML e enviados para o *Web service* de gráficos, transformando as informações cadastradas em gráficos.

4.1 CONSTRUÇÃO DO WEB SERVICE DE ARMAZENAMENTO

A IDE de desenvolvimento do *Web service* de armazenamento foi o NetBeans na versão 6.9.1. Foi utilizada esta versão, pois a mesma oferece suporte para a programação na linguagem JRuby, linguagem que foi utilizada no desenvolvimento da aplicação.

O JRuby é a implementação da linguagem Ruby para a plataforma Java, desenvolvida pela JRuby Team. Essa linguagem permite que se rode script Ruby na máquina virtual Java e como o JRuby foi desenvolvido em Java, também é possível utilizar todo o legado de fontes de outras aplicações. Utilizar se de ambas as sintaxes Ruby + Java, usando dessa forma as melhores características das duas linguagens.

O Armazenamento dos dados XML no *Web service* de armazenamento não será numa base de dados relacional, para este propósito será utilizado o banco de dados especifico para XML, o Berkeley DB XML da Oracle. O Berkeley da Oracle é um projeto *open source* administrado pela Oracle e é um banco de dados baseado

em documentos que são armazenados em recipientes, e são indexados com base em seu conteúdo.

O Berkeley XML é uma base de dados embutida e que tem um analisador de documentos, sendo possível efetuar consultas, alterar e criar novos documentos de XML. Pode-se ainda utilizar-se do XQuery para a recuperação de documentos em uma linguagem bastante similar a SQL dos bancos de dados relacionais.

4.1.1 Configuração do Berkeley na IDE NetBeans

O DB Berkeley XML para ser configurado no NetBeans, requer a que seja incluído três arquivos no Classpath do projeto no NetBeans. Para acessar o Classpath deve-se clicar com botão direito sobre o projeto, ir a propriedades, e na categoria Java, incluir os arquivos db.jar, dbxml.jar e dbxmlexamples.jar que estão na pasta “jar” no diretório onde foi instalado o Berkeley DB XML.

4.2 SISTEMA DE ARMAZENAMENTO DE ARQUIVOS XML

O Sistema de armazenamento de arquivos visa ser um *Web Service* especializado no armazenamento e recuperação de arquivos XML, desse modo, o NetBeans deve conversar com o Berkeley que gerencia os arquivos XML. Visto ainda que o sistema é um *Web service* específico, é determinante que esse serviço possa oferecer meios para que outras aplicações possam se comunicar. Para desenvolver a comunicação entre os sistemas distribuídos, foi o escolhido o procedimento de chamada remota para ser implementado nesse *Web service*.

O protocolo RPC fornece meios para a chamada de procedimento remoto, desse modo o *Web service* de armazenamento disponibiliza procedimentos que outras aplicações possam utilizar. O XMLRPC utiliza XML para codificar os suas chamadas e o HTTP como mecanismo de transporte. É relativamente fácil de ser implementado nas mais diversas linguagens, como Java e JRuby, Perl e PHP, entre muitas outras. Para a instalação no JRuby é necessário incluir algumas classes nativas da linguagem.

Na Figura 10, é incluído duas classes nativas do JRuby. A primeira delas é o Java, será utilizado mais tarde para comunicar com a base de dados Berkeley. A segunda faz a introdução de um servidor RPC que será utilizado para a comunicação com as outras aplicações.

```
1 require 'java'
2 require 'xmlrpc/server'
3
```

Figura 10 – Inclusão de classes na aplicação JRuby

Na Figura 11 é demonstrado a inclusão de classes na aplicação, para que a aplicação JRuby possa conversar com o banco de dados da Oracle. É necessário incluir arquivos .jar da instalação do Berkeley no *classpath* da aplicação.

```
3
4 include_class 'com.sleepycat.dbxml.XmlManager'
5 include_class 'com.sleepycat.dbxml.XmlQueryContext'
6
```

Figura 11 – Inclusão de classes do Berkeley DB XML

Na Figura 12, é mostrado um exemplo básico de implementação de serviço XMLRPC em uma aplicação JRuby, é definido uma declaração de bloco *index* onde será inicializado o serviço. Na linha 8 é onde ocorre a criação do servidor XMLRPC, observe que é possível passar por parâmetro a porta do servidor que se deseja levantar o serviço.

```
7   def index
8     s = XMLRPC::Server.new(3005)
9
10    s.add_handler("my_server.add") do |a,b|
11      a + b
12    end
13
14    s.set_default_handler do |name, *args|
15      raise XMLRPC::FaultException.new(-99, "Metodo não encontrado")
16    end
17
18    s.serve
19  end
```

Figura 12 – Implementação de servidor RPC

Ainda na figura 12, especificamente nas linhas 10 a 12, é definido um procedimento que será publicado no servidor RPC, o nome do procedimento exposto será “*my_server.add*” e o procedimento deve receber dois parâmetros de entrada e resultará em um parâmetro de retorno, que no caso do exemplo, a soma dos parâmetros de entrada, simulando uma conta de adição de uma calculadora.

Caso o procedimento chamado pelo cliente não ser encontrado em meios aos procedimentos expostos pelo servidor ou ainda á quantidade de parâmetros não corresponder, o servidor retorna uma exceção com a mensagem do erro, essa exceção é mostrada nas linhas 14 á 16 do exemplo da figura 12.

4.2.1 Trabalhando com Berkeley DM XML

O Berkeley DB XML será o gerenciador de dados XML da aplicação de armazenamento. Como o Berkeley trabalha com *containers* e é possível criar vários containers para cada aplicação, devemos Como o Berkeley trabalha com containers e é possível criar vários contudo criar um container para que a aplicação possa armazenar os arquivos XML.

```
14
15      s.add_handler("my_server.create_container") do |nome_container|
16        my_manager = XmlManager.new
17        my_manager.createContainer(nome_container);
18        my_manager.close()
19      end
20
```

Figura 13 – Criação de um Container no Berkeley DB XML

Na figura 13, é exposto um procedimento que será responsável pela criação do container que armazenará os dados XML. Este procedimento aceita um parâmetro de entrada que especifica o nome com que será criado o container em disco.

No caso de um cliente conter um arquivo XML e quiser armazena-lo no servidor, o cliente pode utilizar o procedimento mostrado na Figura 14.

```
21    s.add_handler("my_server.xml_upload") do |dados,nome_container|
22      my_manager = XmlManager.new
23      my_container = my_manager.openContainer(nome_container)
24      my_container.putDocument(dados['datafile'].original_filename, dados['datafile'].read)
25      my_manager.close()
26    end
27
```

Figura 14 – Upload de arquivo XML

A Figura 14, o procedimento de *upload* recebe dois parâmetros, o primeiro é o arquivo XML e o segundo a descrição do *container* no qual deverá ser gravado o arquivo XML.

Na Figura 15, o procedimento disponível para aplicações clientes é responsável pelo armazenamento de dados XML. Este procedimento é bastante parecido com o procedimento demonstrado anteriormente. A diferença, é que anteriormente o arquivo é enviado para o servidor, e neste caso, o texto do XML é passado por parâmetro.

```
27
28    s.add_handler("my_server.salva_xml") do |nome,texto,nome_container|
29      my_manager = XmlManager.new
30      my_container = my_manager.openContainer(nome_container)
31      my_container.putDocument(nome, texto)
32      my_manager.close()
33    end
34
```

Figura 15 – Upload de dados XML

Para recuperar arquivos de dados XML do Berkeley é utilizado o procedimento da Figura 16. Esse procedimento aceita dois parâmetros de entrada, o primeiro é o nome do arquivo e o segundo é o nome do *container* utilizado para armazenar o XML.

```
34
35    s.add_handler("my_server.recupera_xml") do |docName,nome_container|
36      my_manager = XmlManager.new
37      my_container = my_manager.openContainer(nome_container)
38      doc = my_container.getDocument(docName).getContentAsString()
39    end
40
```

Figura 16 – Recuperar arquivo XML no Berkeley DB XML

O banco de dados da Berkeley da Oracle oferece o XQuery como linguagem de consulta, para proporcionar maior flexibilidade para extrair os dados no banco de dados. O XQuery utiliza XPath para sintaxe de expressões, dessa forma trata de partes específicas de um documento XML. A linguagem também permite que se crie novos arquivos XML baseados nos resultados da consulta, ou mesmo formular uma consulta complexa pesquisando dentro dos nodos do XML.

Na Figura 17, é mostrado á utilização de consultas, onde são retornados todos os arquivos XML que tem atributo “*title*” igual á despesas.

```
40
41    s.add_handler("servidor.executa_sql") do |nome_container|
42      import com.sleepycat.dbxml.XmlContainer;
43      import com.sleepycat.dbxml.XmlException;
44      import com.sleepycat.dbxml.XmlManager;
45      import com.sleepycat.dbxml.XmlQueryContext;
46      import com.sleepycat.dbxml.XmlQueryExpression;
47      import com.sleepycat.dbxml.XmlResults;
48
49      myQuery = "collection('"+nome_container+"')/chart[@title='DESPESAS']";
50
51      my_manager = XmlManager.new
52      my_container = my_manager.openContainer(nome_container)
53      context = my_manager.createQueryContext();
54
55      results = my_manager.query(myQuery, context)
56    end
57
```

Figura 17 – Consultas XQuery com nodos XML

4.3 CONSTRUÇÃO DO WEB SERVICE DE GRÁFICOS

O Web service de gráficos será responsável por gerar gráficos com os arquivos XML que serão enviados pelos aplicativos clientes. Os gráficos são uma das melhores maneiras de representar uma informação muito técnica, ou mesmo uma informação numérica.

Para o desenvolvimento do trabalho de conclusão, e as respectivas análises de comunicação entre sistemas distribuídos, foi desenvolvido este Web service que consome arquivos XML que foram produzidas entre a aplicação desktop e o Web service de Armazenamento e que demonstram o resultado através de gráficos.

4.3.1 Desenvolvimento do Web service de gráficos

O *Web service* de gráficos receberá o arquivo de dados XML das aplicações clientes via *post*, desse modo é necessário pegar os valores postados para o serviço e armazená-los em variáveis. A figura 18 está demonstrando esse procedimento, foi definido um método *index* e que dentro dele é setado as variáveis que foram passadas por parâmetro, no caso o XML e o arquivo de parâmetros do gráfico.

Quando não está definido no código a view que aplicação deve renderizar, por padrão Ruby on Rails entende que a view deve ser o mesmo nome do método, nesse caso "*index*", então caso não tenha uma View com nome "index.rhtml" causará um erro na aplicação.

```
1 class GLinhaController < ApplicationController
2   def index
3     @configuracao = params[:envia_xml]
4     @xml = params[:envia_xml]
5   end
6 end
7
```

Figura 18 – Parâmetros recebidos na aplicação de Gráficos

A view descrita acima é a view mostrada na Figura 19. Ela é composta de códigos HTML, códigos Javascript e ainda valores da aplicação servidor do ROR, que no caso, são as duas variáveis recebidas via *post* do cliente. O código em Javascript é utilizado para invocar objetos Javascript das bibliotecas do dos Gráficos, passados o arquivo de configurações do gráfico e ainda os valores de dados XML, é possível verificar o resultado final, que é o gráfico da aplicação.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
       <title>amCharts example</title>
    </head>
    <body style="background-color:#EEEEEE">

<script type="text/javascript" src="\javascripts\swfobject.js"></script>

<div id="amcharts_1327961746641">You need to upgrade your Flash Player</div>
<script type="text/javascript">
    var so = new SWFObject("/amcharts/flash/amline.swf", "amline", "680", "280", "8", "#FFFFFF");
    so.addVariable("path", "amline/");
    so.addVariable("chart_settings", encodeURIComponent("<%= @configuracao.gsub("\"", "'") %>"));
    so.addVariable("chart_data", encodeURIComponent("<%= @xml.gsub("\"", "'") %>"));
    so.write("amcharts_1327961746641");
</script>

</body>

</html>
```

Figura 19 – Renderização do Gráfico

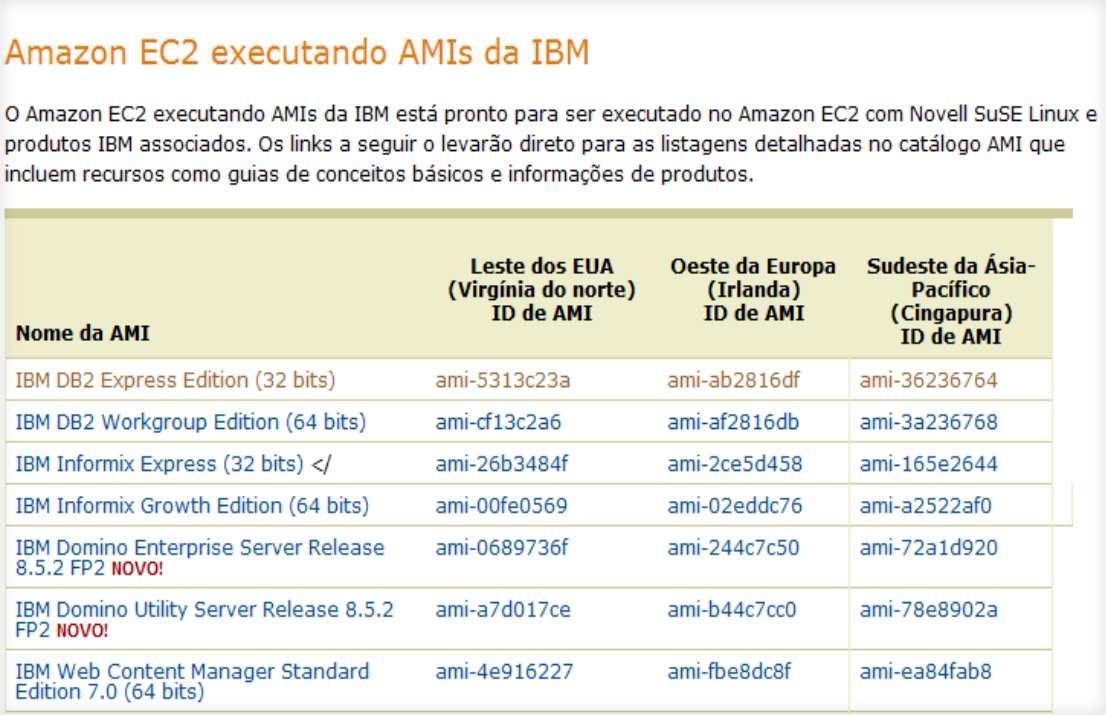
Para a visualização correta dos gráficos é necessário ter instalado as versões mais recentes do Flash ou ainda de um navegador atualizado que suporte HTML 5 para a visualização de gráficos gerados a partir de códigos Javascript.

4.4 IMPLEMENTANDO SERVIDORES NA AMAZON

Para utilizar os serviços da Amazon é necessário criar uma conta na página da empresa no endereço http://aws.amazon.com/pt/ec2/. Será necessários alguns dados cadastrais da conta e um cartão de crédito para o pagamento do serviço. A Amazon tem vários serviços na nuvem, um dois principais é o Amazon EC2 que foi projetado para tornar a escalabilidade computacional mais fácil para desenvolvedores.

A grande versatilidade do serviço EC2 é que existem muitos sistemas operacionais totalmente configurados e prontos para ser iniciados, inclusive com os ambientes necessários para o desenvolvimento ou o suporte para aplicações de servidor. Por exemplo, caso a algum cliente deseja um servidor de banco de dados, pode utilizar o DB2 da IBM. A IBM disponibiliza seus softwares configurados numa AMI (*Amazon Machine Image*) na Amazon.

Na Figura 20, mostra as várias máquinas preparadas para rodar softwares da IBM. Cada usuário da Amazon pode inicializar uma máquina virtual padrão sem softwares adicionais já instalados, personalizar da melhor forma que deseja, salvar e disponibilizar para os outros usuários, dá mesma forma que a IBM fez.



Amazon EC2 executando AMIs da IBM

O Amazon EC2 executando AMIs da IBM está pronto para ser executado no Amazon EC2 com Novell SuSE Linux e produtos IBM associados. Os links a seguir o levarão direto para as listagens detalhadas no catálogo AMI que incluem recursos como guias de conceitos básicos e informações de produtos.

Nome da AMI	Leste dos EUA (Virgínia do norte) ID de AMI	Oeste da Europa (Irlanda) ID de AMI	Sudeste da Ásia-Pacífico (Cingapura) ID de AMI
IBM DB2 Express Edition (32 bits)	ami-5313c23a	ami-ab2816df	ami-36236764
IBM DB2 Workgroup Edition (64 bits)	ami-cf13c2a6	ami-af2816db	ami-3a236768
IBM Informix Express (32 bits) </	ami-26b3484f	ami-2ce5d458	ami-165e2644
IBM Informix Growth Edition (64 bits)	ami-00fe0569	ami-02eddc76	ami-a2522af0
IBM Domino Enterprise Server Release 8.5.2 FP2 NOVO!	ami-0689736f	ami-244c7c50	ami-72a1d920
IBM Domino Utility Server Release 8.5.2 FP2 NOVO!	ami-a7d017ce	ami-b44c7cc0	ami-78e8902a
IBM Web Content Manager Standard Edition 7.0 (64 bits)	ami-4e916227	ami-fbe8dc8f	ami-ea84fab8

Figura 20 – AMIs da IBM
Fonte: Adaptado de Amazon (2012)

Para procurar uma AMI, é necessário *logar* na conta da Amazon, ir para o serviço EC2, ir em Imagens, AMIs, faze a busca entre todas as imagens prontas e disponíveis para o serviço, observando a plataforma que se deseja.

Na Figura 21, é efetuado uma consulta no site da Amazon, onde resultou as AMIs disponíveis para todas as plataformas que o pessoal do site Bitnami configurou. A consulta resulta na descrição da AMI, se é publica ou paga e o sistema operacional.

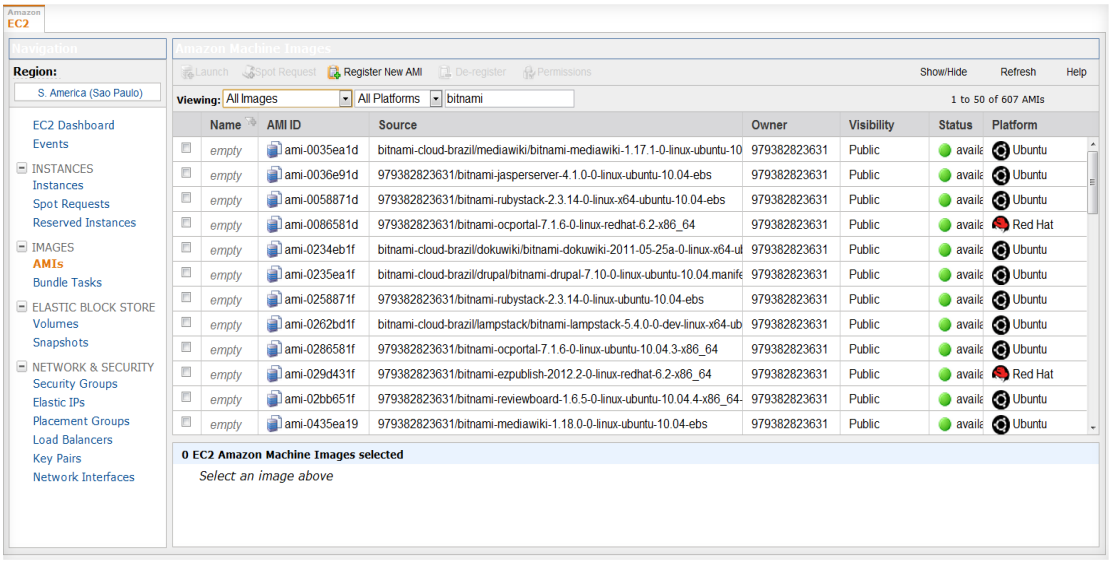


Figura 21 – AMIs disponíveis na Amazon
Fonte: Adaptado de Amazon (2012)

Também é possível iniciar uma AMIs estando conectado na conta da Amazon e selecionando uma AMI através de um link do site da Bitnami. A Figura 22 foi adaptada do site da Bitnami, que mostra alternativas de servidores para aplicações com suporte JRuby. O site disponibiliza quatro imagens com opção de dois sistemas operacionais, para serem inicializados na Amazon, ainda é possível baixar e instalar a máquina virtual no computador do desenvolvedor, utilizando softwares como VirtualBox da Oracle para iniciar a VM.

O Site da Bitnami disponibiliza ainda vários outros servidor totalmente configuradores, como servidores de aplicações para Ruby on Rails, Djando, JBoss, LAMP, WAMP. Outras AMIs configuradas para Infraestrutura são ferramentas de: Bug Tracker (gerenciamento de bug), Business Inteligence, CMS, softwares de criação de Fóruns, Portais, Gerenciamento de Projetos, Wiki e e-Commerce.

Para a implementação do servidor que atuará na armazenagem de dados XML, será utilizado uma AMI da Bitnami para JRuby, configurado no Linux com sistema operacional Ubuntu. As tecnologias já configuradas dentro da AMI para JRuby da Bitnami são o Glassfish, Java, JDBC, Mysql, Ruby on Rails, Subversion, Tomcat e Apache.

Figura 22 – AMIs JRuby da Amazon no site Bitnami
Fonte: Adaptado de Bitnami (2012)

A Figura 22, mostra as AMIs disponíveis para Amazon configuradas no sistema operacional Linux com o JRuby configurado. Ao clicar no link de uma das AMI acima, previamente *logado* na conta da amazon, o link direciona para a instalação da máquina virtual na Amazon, a partir desse ponto existe poucos passos para instanciar a VM.

No primeiro passo demonstrado na Figura 23, a Amazon mostra as informações sobre a AMI, como Sistema Operacional, a Arquitetura, o Tipo de Volume que será armazenado a imagem.

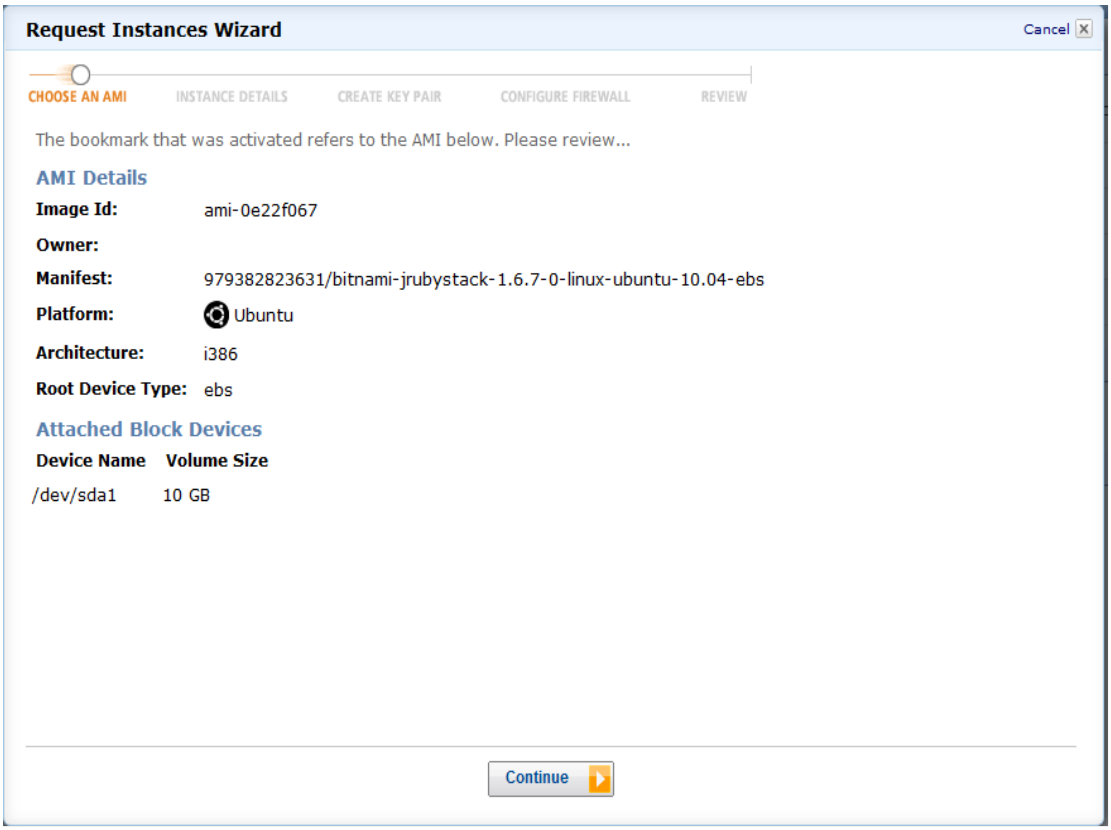


Figura 23 – Configurando máquina virtual parte 1
Fonte: Adaptado de Amazon (2012)

No segundo passo da instalação do servidor JRuby, demonstrado na Figura 24, é definido as primeiras configurações da nova Máquina Virtual que consiste no número de Instâncias que serão executadas e o tipo de máquina. Número de instâncias é a quantidade de máquinas igual á AMI original executadas em paralelo. Já a opção *Instance Type* tem como objetivo selecionar o número de CPU *units* (unidades de processamento da Amazon), a quantidade de Cores do processador e a Quantidade de Memória utilizada na máquina que será inicializada.

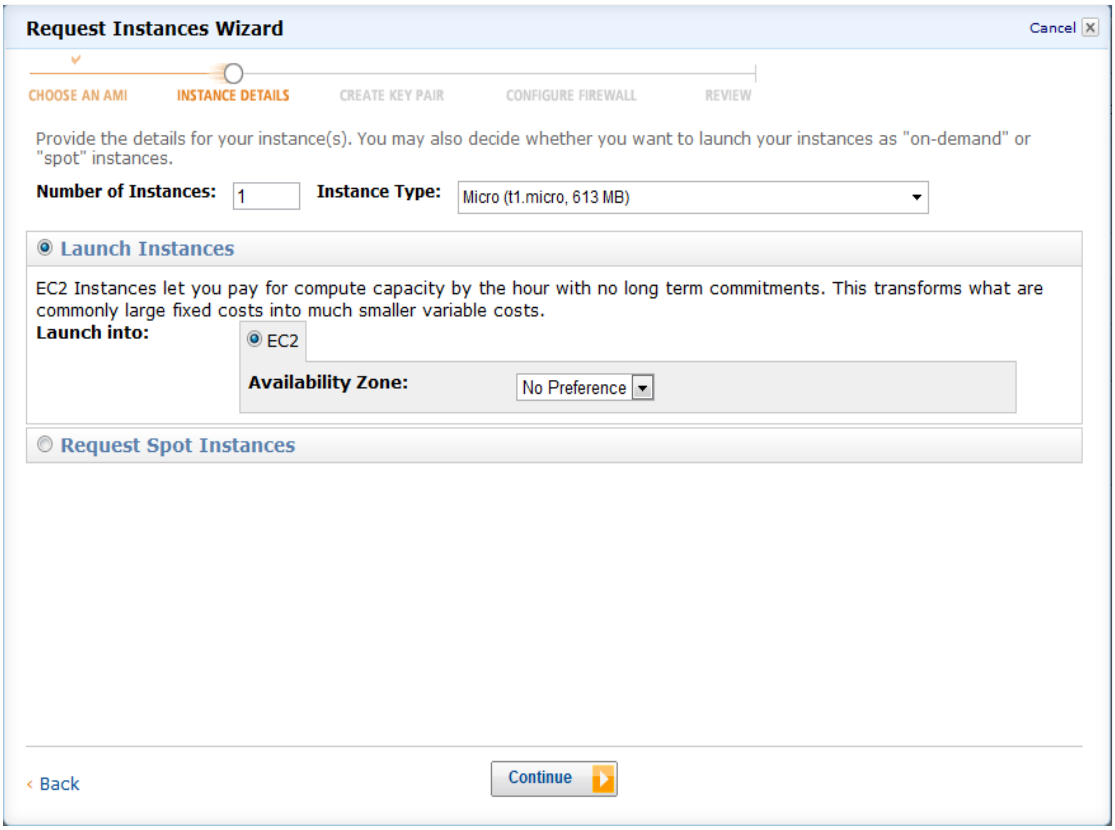


Figura 24 – Configurando máquina virtual parte 2
Fonte: Adaptado de Amazon (2012)

Na seleção *Lauch Instances*, ainda na Figura 24, marcada a opção o valor cobrado será pela capacidade computacional por hora, calculado sobre o tempo que a máquina estiver rodando. Na seleção de *Request Spot Instances*, o cliente define um preço que quer pagar para executar sua AMI, caso a Amazon disponha de processamento e o preço que oscila baseado na oferta e demanda, for abaixo do preço definido pelo cliente, a AMI é executada. Essa configuração permite redução de custo quando existe uma flexibilidade de tempo para a utilização da AMI.

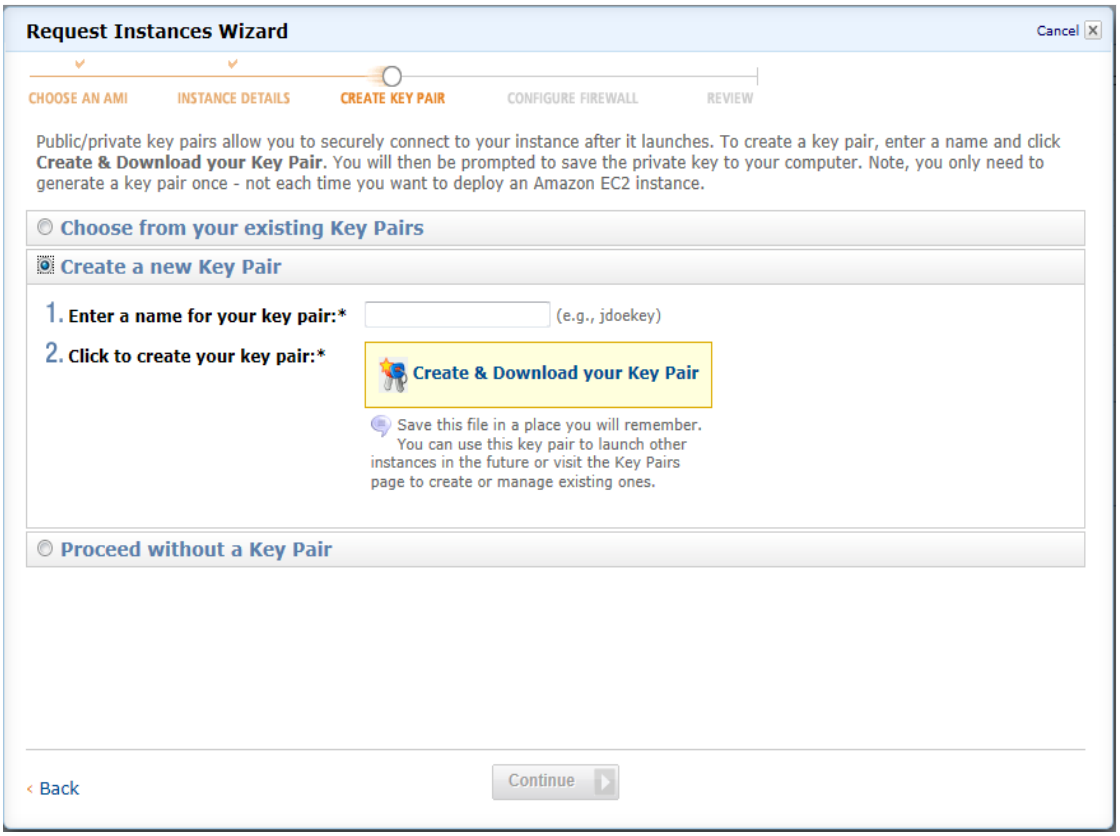


Figura 25 – Configurando máquina virtual parte 3
Fonte: Adaptado de Amazon (2012)

A figura 25 mostra o terceiro passo que deve ser configurado para instanciar uma máquina virtual na Amazon. Neste passo é necessário criar uma chave de segurança que será utilizado quando se tentar conectar no servidor remoto. Caso o cliente já tenha uma chave de segurança não é necessário criar uma nova.

A Amazon utiliza chaves de segurança para tornar as máquinas virtuais mais seguras, visto que todas as máquinas devem oferecer meios de conexão externa para configurações no servidor ou mesmo para subir novas versões dos *softwares* do desenvolvedor.

Na Amazon é possível criar várias diretivas de segurança para as máquinas virtuais. Essas diretivas funcionam similarmente ao *firewall* do Windows, onde pode ser liberadas ou bloqueadas portas externas para acesso as máquinas virtuais. Portas como FTP, SSL, e o para conexão RDP que é o padrão de conexão no Windows podem ser liberadas nessa configuração.

A Figura 26 mostra o penúltimo passo para iniciar um servidor na Amazon. Neste passo é necessário configurar o *firewall* da máquina, para isso a máquina dever ser incluída num grupo de segurança. No grupo estão configuradas todas a portas de entrada e saída da máquina virtual, caso não tenha nenhum grupo é possível cadastrar um novo grupo selecionando a opção "*create a new Segurity Group*".

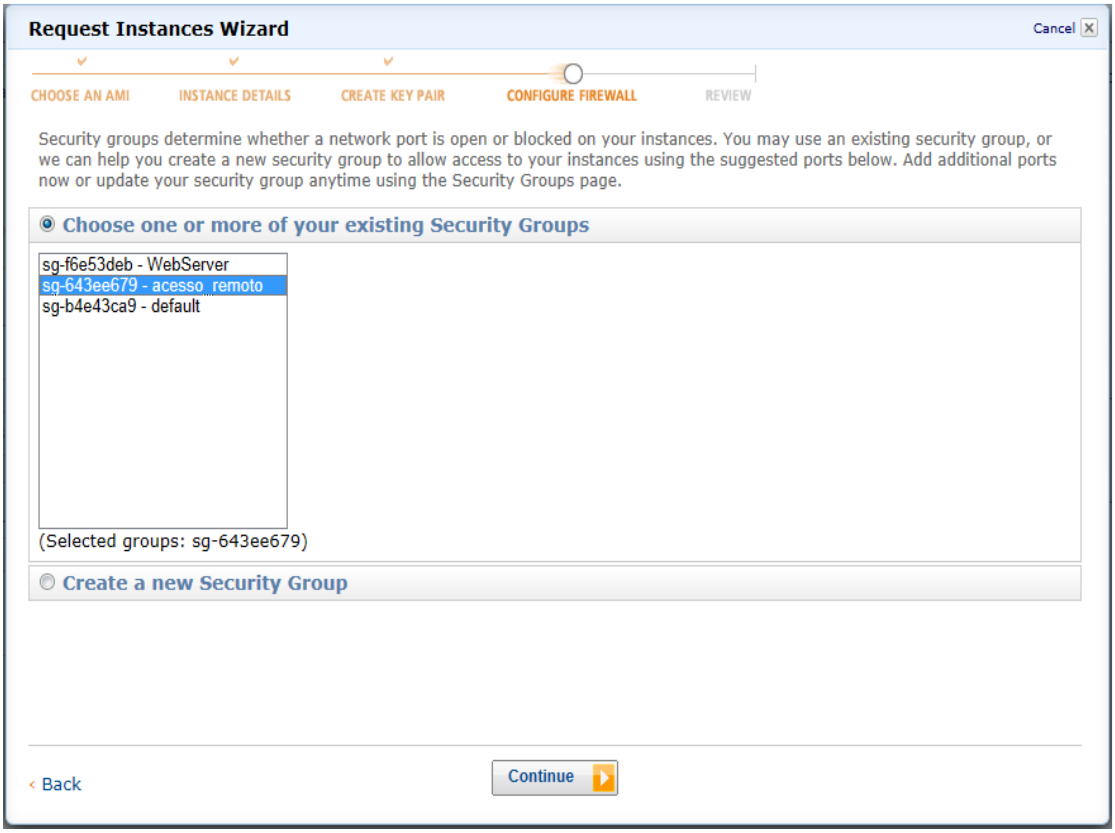


Figura 26 – Configurando máquina virtual parte 4,
Fonte: Adaptado de Amazon (2012)

A Figura 27 já mostra a máquina virtual pronta para ser inicializada, se todos os passos anteriores estão de acordo a máquina pode ser concluída clicando em *Lausch*. Caso é necessário rever algum ponto, pode ser feito clicando o *link* da configuração que se deseja alterar.

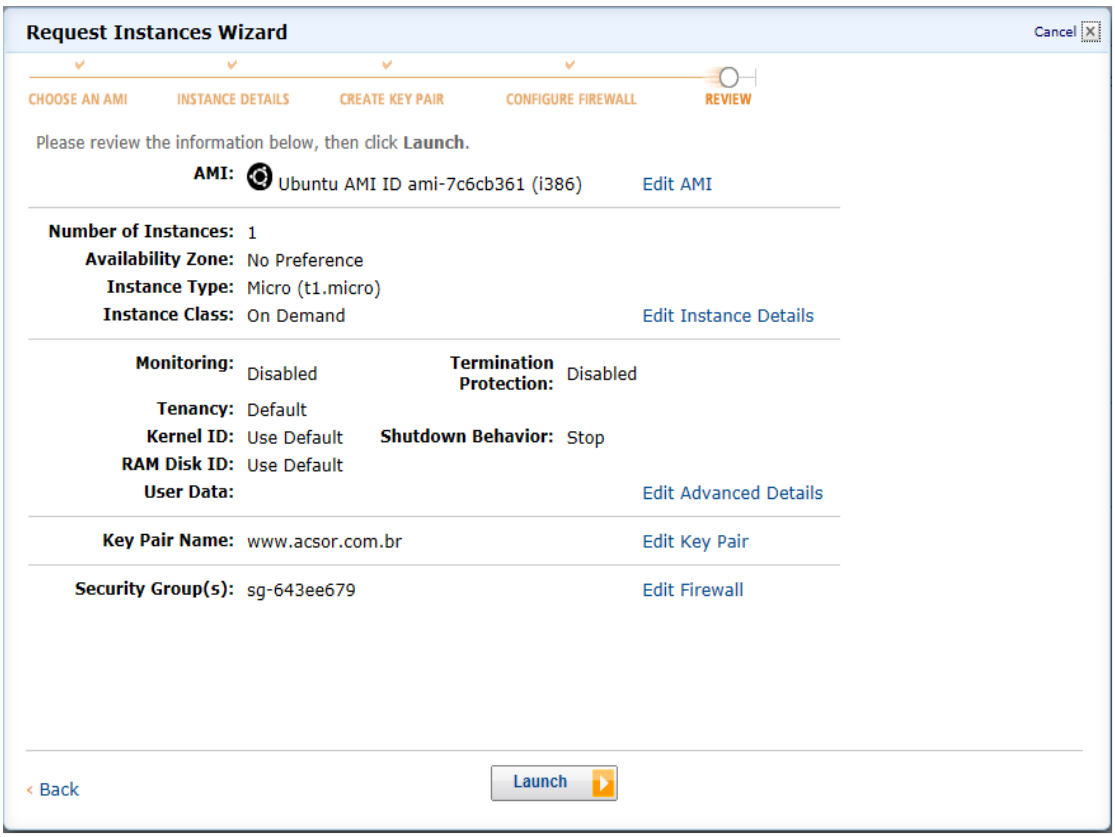


Figura 27 – Configurando máquina virtual parte 5
Fonte: Adaptado de Amazon (2012)

Ao terminar a configuração da máquina virtual já é possível visualizar ela, ao clicar no menu lateral esquerdo, na opção "*instances*" no site de gerenciamento da Amazon. A Figura 28 mostra as máquinas disponíveis, configuradas e já inicializadas dos servidores JRuby e Ruby on Rails que serão utilizados nesse trabalho.

A primeira máquina é o servidor para aplicações JRuby, que foi disponibilizado a aplicação de Armazenamento e Recuperação de dados XML. A segunda máquina é o servidor que mantém aplicações Ruby on Rails, que foi utilizado para manter a aplicação que gera os gráficos desenvolvida nesse trabalho.

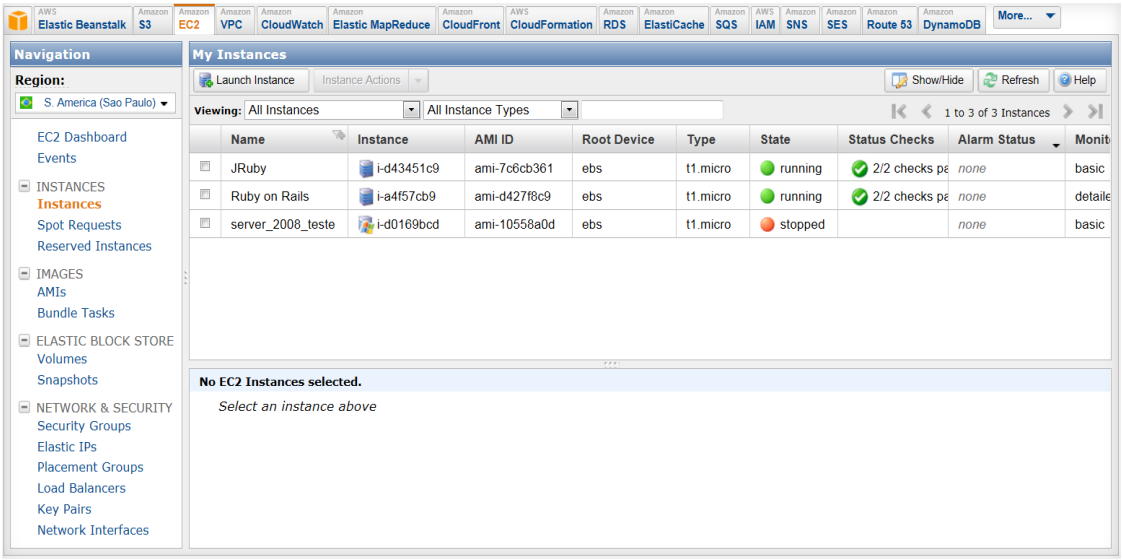


Figura 28 – Servidores de aplicações WEB rodando na Amazon
Fonte: Adaptado de Amazon (2012)

4.5 CONSTRUÇÃO DO APLICAÇÃO DESKTOP

A aplicação *desktop* fará a ligação entre os dois *Web services* citados acima. Ele deverá implementar o protocolo cliente para RPC para conversar com o *Web service* de Armazenamento e ainda postar arquivos via HTTP para a aplicação de Gráficos.

Para a implementação do cliente RPC em Delphi será utilizado componentes open source desenvolvidos para Delphi, o dxmlrpc, que podem ser baixados no site http://sourceforge.net/projects/delphixml-rpc/files/.

A versão do dxmlrpc foi implementada utilizando a palheta de componentes *indy 9*, sendo necessária está versão para funcionar corretamente e não precisar de ajustes extras. Foi utilizado a versão 7 do Delphi para a implementação do cliente *desktop* para comunicação com *Web service* JRuby e Ruby on Rails.

A primeiro passo para criar uma aplicação cliente para RPC é criar um objeto do tipo TRpcCaller da biblioteca *dxmlrpc* e adicioná-lo as variáveis Hostname, HostPort e EndPoint do Web service que está rodando o servidor RPC.

A Figura 29, mostra como pode ser criado o objeto TRpcCaller e ainda os valores sendo setados no componente, na função do exemplo os parâmetros estão fixos, mas podem ser configurados dinamicamente. O parâmetro HostName é o endereço da maquina remota que está rodando o servidor RPC, neste parâmetro pode ser direcionado para o servidor JRuby na Amazon, configurado anteriormente.

```
function TForm1.ResultRpcCaller: TRpcCaller;
begin
  Result := TRpcCaller.Create;
  Result.HostName := 'localhost';
  Result.HostPort := 3005;
  Result.EndPoint := '/servico';
end;
```

Figura 29 – Resulta Objeto de conexão com serviço RPC

Para criar um *container* no *Web Service* de armazenamento foi criado um procedimento no servidor chamado "*my_server.create_container*" e publicado conforme já descrito na figura 13. Agora é a vez dá aplicação *desktop* utilizar daquele procedimento, sendo implementado na aplicação cliente do RPC da aplicação *desktop*.

A figura 30 representa procedimento feito em Delphi para consumir o procedimento feito no servidor RPC do JRuby. É utilizado para isso a um componente de parâmetros do servidor remoto do tipo. Para resultar na conexão com o servidor e é criado um objeto do tipo TRpcFunction cuja a propriedade "*ObjectMethod*" identifica o procedimento que se deseja conectar no servidor RPC, e são adicionados os parâmetros que serão passados.

```
procedure TForm1.CriaContainer;
var
  RpcFunction: TRpcFunction;
begin
  RpcCaller := ResultRpcCaller;

  RpcFunction := TRpcFunction.Create;
  RpcFunction.ObjectMethod := 'my_server.create_container';
  RpcFunction.AddItem(EdNomeContainer.Text + '.DBXML');

  RpcResult := RpcCaller.Execute(RpcFunction);
  ProcessaRpcResult(RpcResult);
end;
```

Figura 30 – Procedimento em Delphi que invoca método Criar Container

Para que a aplicação *desktop* possa utilizar procedimentos remotos do servidor, o nome do procedimento deve ser o mesmo que o do servidor, a lista de parâmetros com os tipos de dados também devem ser os mesmos. A cada novo parâmetros repete-se o *AddItem* do objeto TRpcFunction para adicionar novo valor.

Com o objeto de conexão é evocado a função Execute passando por parâmetro a instância do objeto TRpcFunction resultando num objeto do tipo IRpcResult, que contém o resultados da execução do procedimento remoto.

A passagem de arquivos XML via procedimento de chamada remota torna-se a programação um pouco mais complexa do que simplesmente passagem de parâmetros de texto. No caso de passagem de qualquer arquivo ou objeto por parâmetro, deve-se perceber que a aplicação remota deve entender os dados enviados. No caso de envio de um arquivo XML, os dados devem ser serealizados.

Para que os dados sejam serializados, o dados XML foram convertidos em *Buffer* de um componente do tipo TMemoryStream.

```
procedure TForm1.EnviaXML;
var
  Stream: TMemoryStream;
  RpcFunction: TRpcFunction;
  NomeArquivo : String;
begin
  RpcCaller := ResultRpcCaller;

  RpcFunction := TRpcFunction.Create;
  RpcFunction.ObjectMethod := 'my_server.salva_xml';

  NomeArquivo := GeraChave(30);
  RpcFunction.AddItem(NomeArquivo);

  Stream := TMemoryStream.Create;
  try
    Stream.WriteBuffer(Pointer(Memo1.Text)^, Length(Memo1.Text));
    RpcFunction.AddItemBase64StrFromStream(Stream);
  finally
    FreeAndNil(Stream);
  end;
  RpcFunction.AddItem(EdNomeContainer.Text + '.DBXML');

  RpcResult := RpcCaller.Execute(RpcFunction);
  ProcessaRpcResult(RpcResult);
end;
```

Figura 31 – Procedimento envia dados XML através de protocolo RPC

A Figura 31, pode ser observado o procedimento escrito em Delphi para enviar um arquivo XML para uma aplicação remota utilizando o protocolo RPC. Este mesmo procedimento já foi explicado anteriormente quando implementado no servidor da aplicação de Armazenamento.

A Figura 32 demonstra como ficou a aplicação desenvolvida em Delphi. O botão “Cria Container” chama um procedimento remoto no *Web service* de armazenamento responsável pela criação de um *container* para armazenamento de arquivos XML.

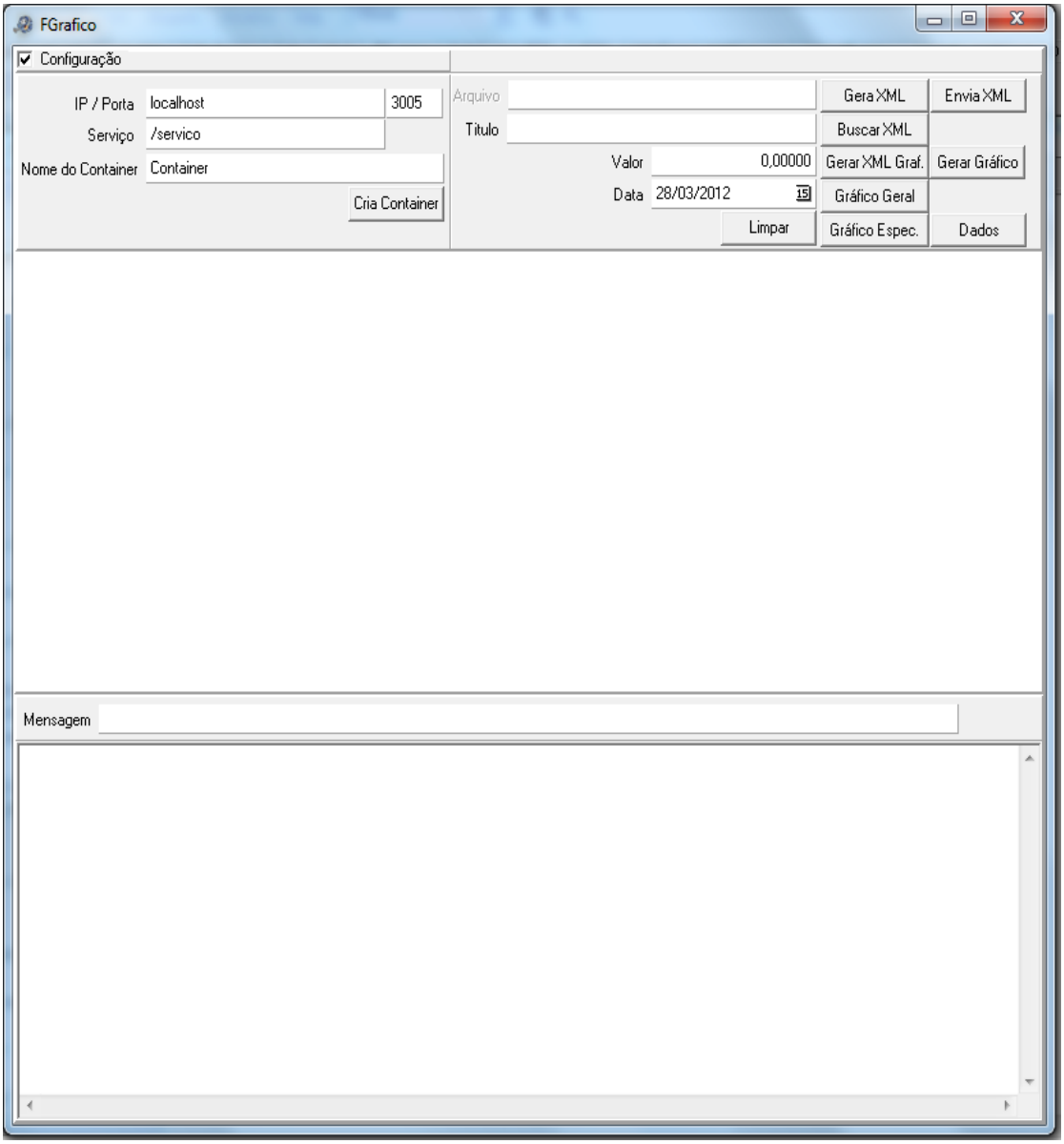


Figura 32 – Aplicação Delphi

Na Figura 22, é informado um Título, um Valor e uma Data, após isso é acionado o botão “Gera XML” que transforma os dados informados em um documento XML e mostra o resultado no quadro de texto na parte inferior da aplicação

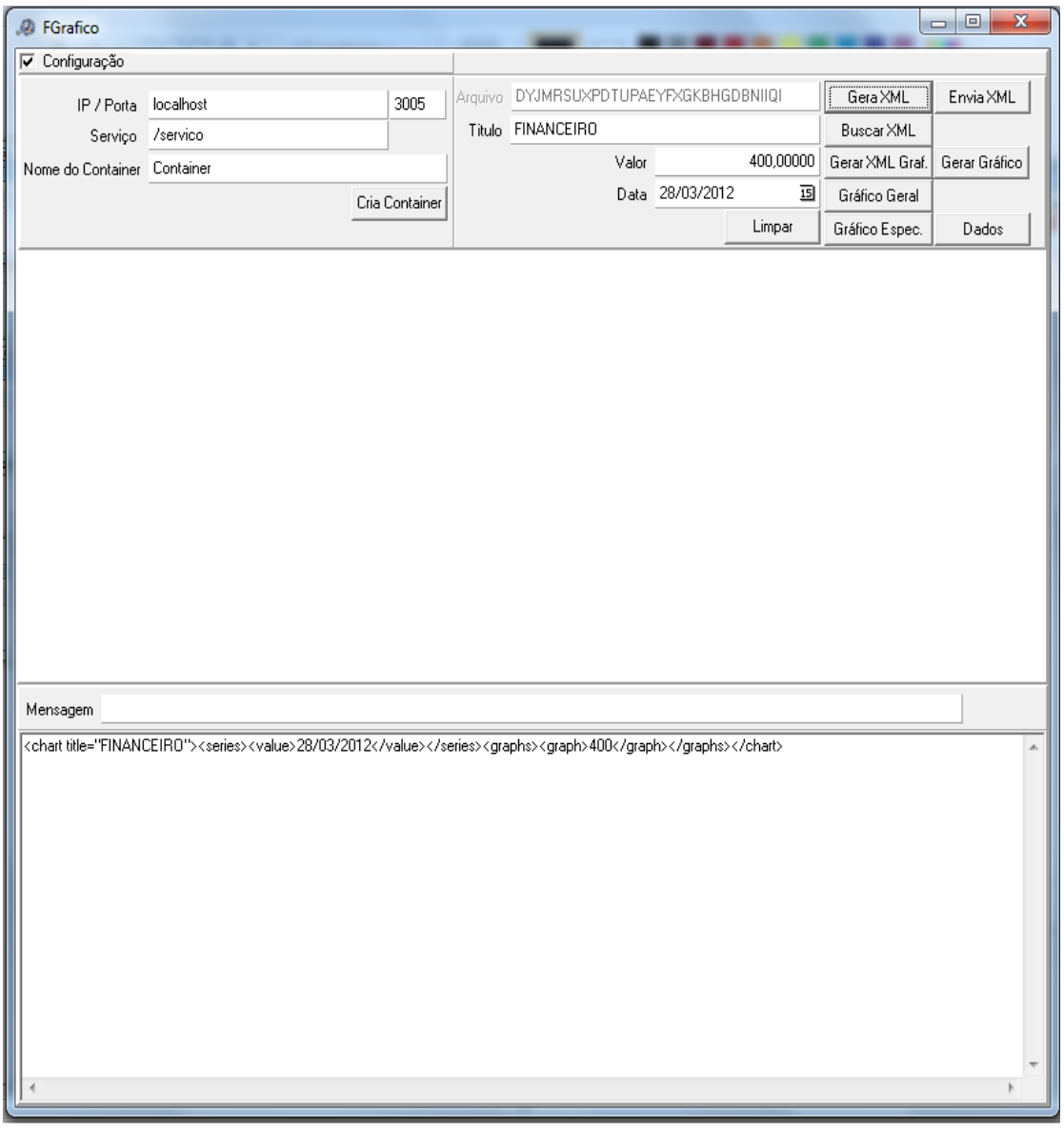


Figura 33 – Geração de dados XML na aplicação

.

Na Figura 34, o arquivo de dados XML é enviado ao servidor para ser armazenado quando o botão “Envia XML” é acionado. O resultado de sucesso é o retorno que a aplicação recebe do servidor e é mostrado no campo texto “Mensagem”.

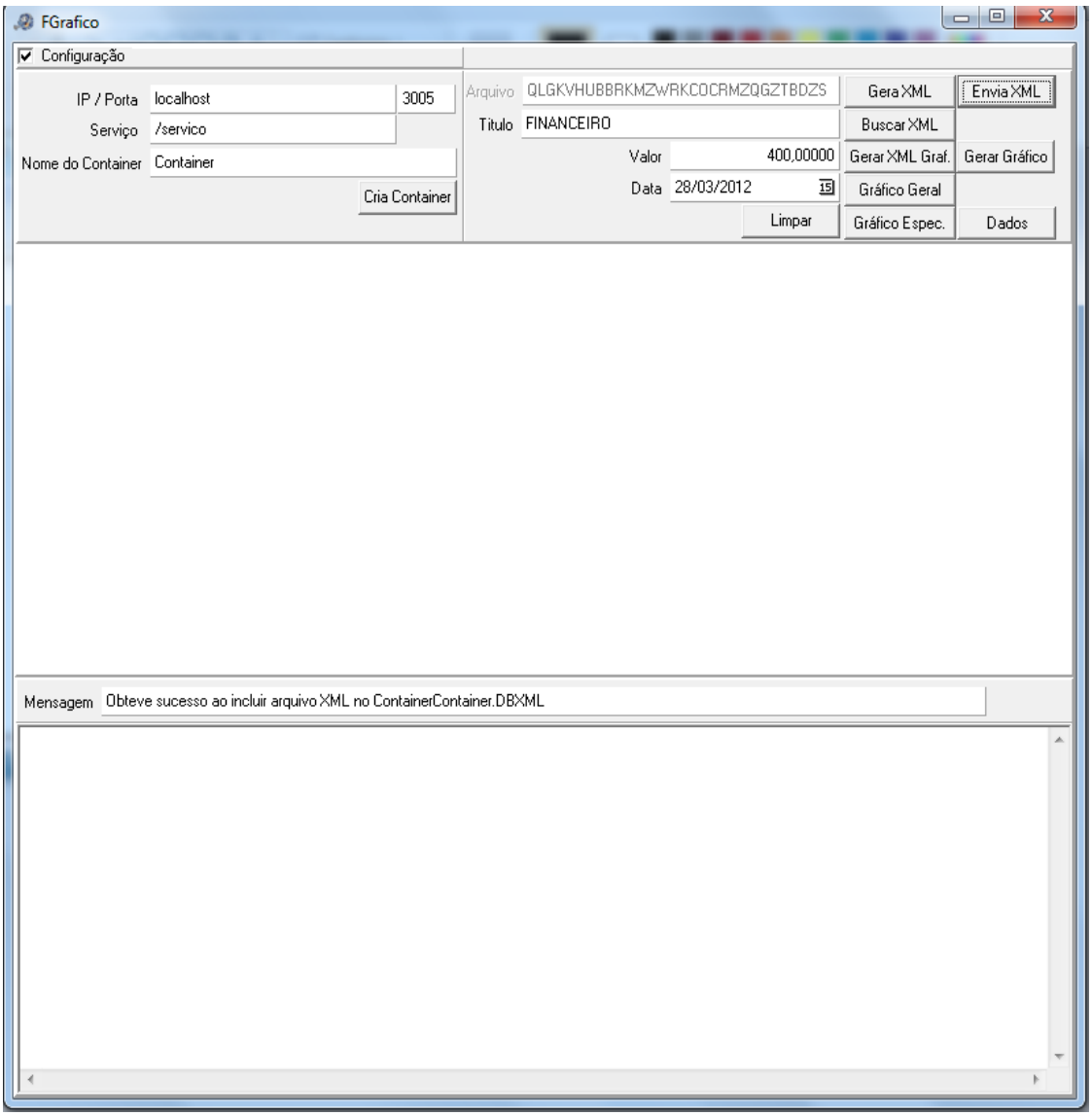


Figura 34 – Enviando dados XML para Web Service

Após enviar alguns arquivos de dados para o servidor é necessário recupera-los para gerar um novo arquivo de dados que será interpretado pelo *Web service* de Gráficos. Esse processo é mostrado na Figura 35.

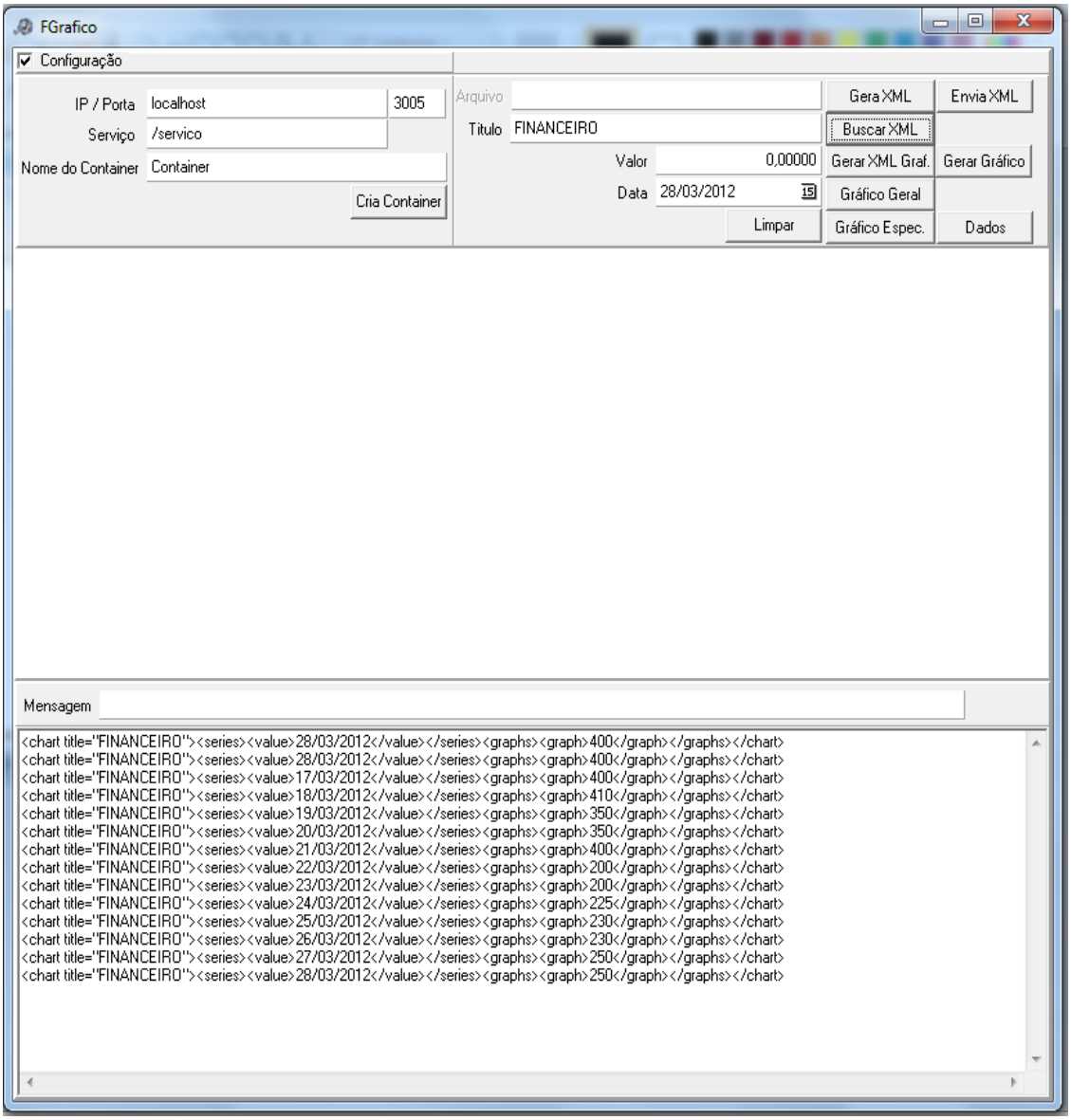


Figura 35 – Retornando dados XML do Web Service

Ao acionar o botão “Gerar XML Graf” os arquivos de dados que foram retornados do servidor de armazenamento, são convertidos em um novo arquivo de dados XML que é o padrão necessário para que o *Web service* de Gráficos possa interpreta-los e que consiga gerar um Gráfico como resultado. Esse procedimento é mostrado na Figura 36.

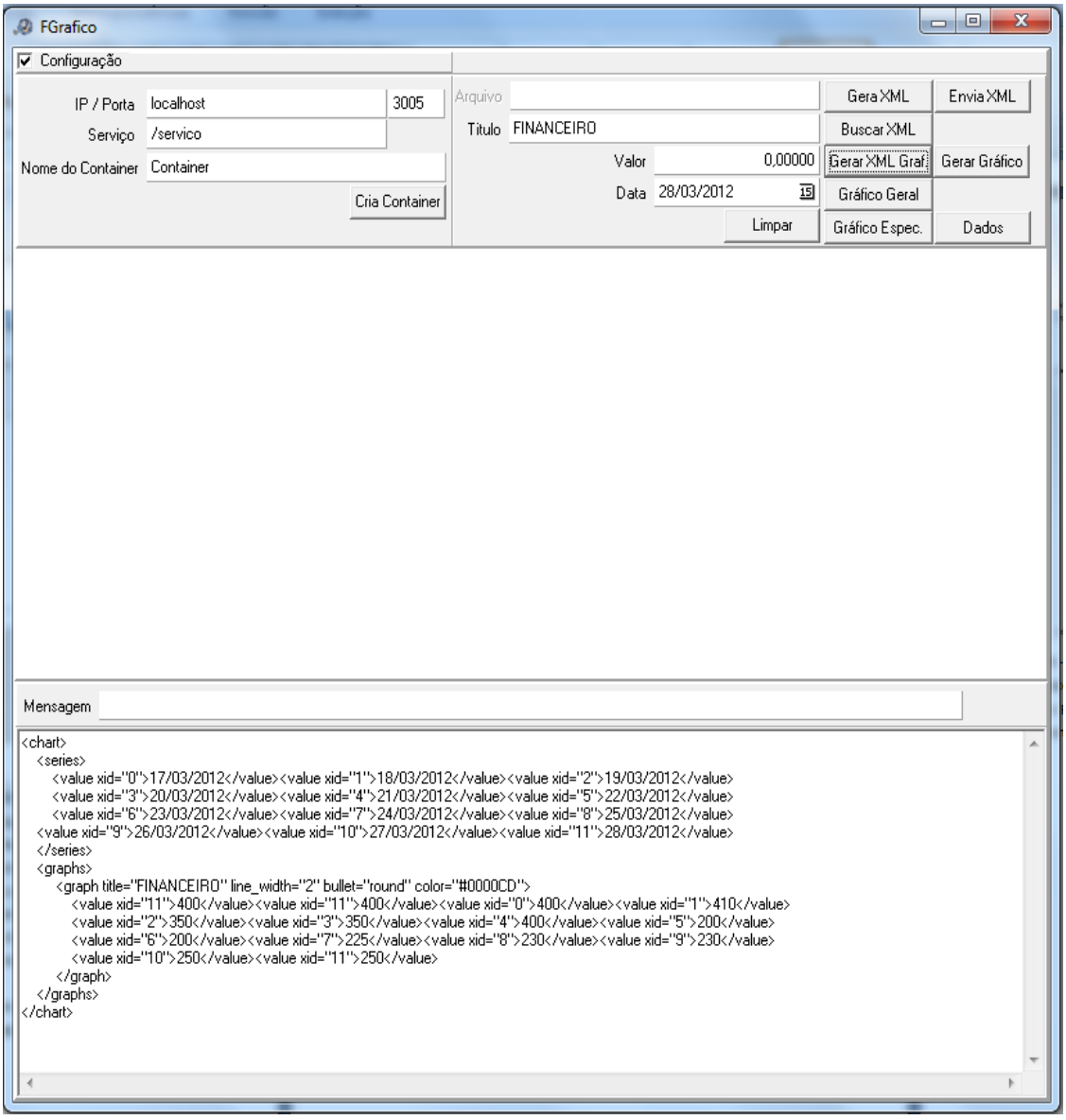


Figura 36 – Gerando XML de dados do Gráfico

Ao gerar e enviar várias vezes a informação de Título, Valor e Data para o servidor é possível visualizar as informações através de um gráfico conforme demonstrado na figura 38. No caso do exemplo, foi gerado 12 pontos de informação, transformado em XML e enviados ao servidor de Armazenamento XML, cada um representando um ponto no gráfico.

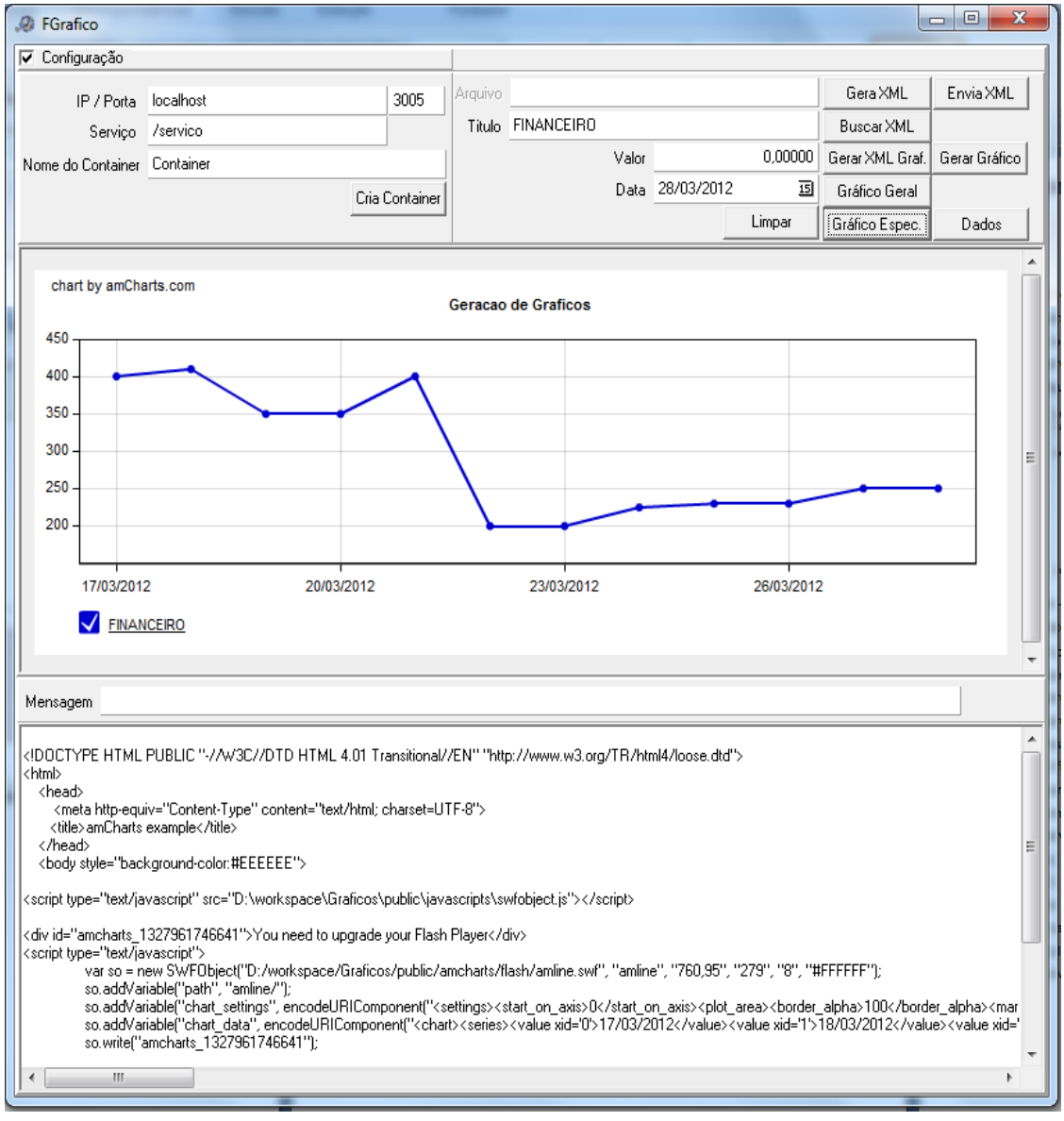


Figura 37 – Demonstrando o Gráfico gerado no Web Service

A cada título diferente postado no servidor é gerado uma nova linha no gráfico. No Exemplo da Figura 38, foram lançados quatro títulos distintos: VENDAS, COMPRAS, ESTOQUE, FINANCEIRO, cada um com vários pontos de informação, gerando assim quatro linhas com seus respectivos valores no gráfico.

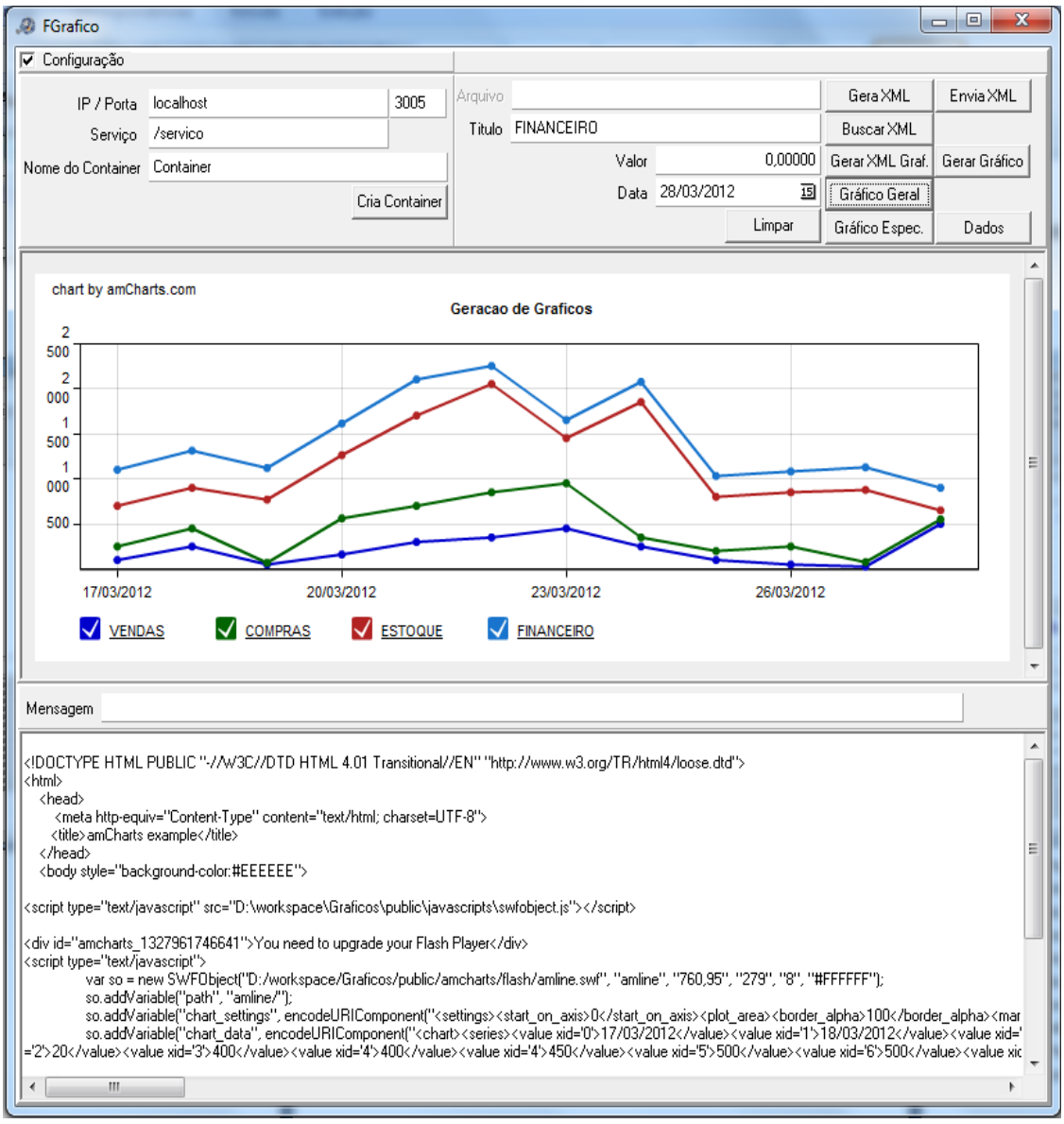


Figura 38 – Demonstrando o Gráfico com dados

Nessa tela acima, é demonstrado o resultado final da comunicação entre uma aplicação desenvolvida para Windows e que foi implementada para se utilizar de *Web services* para armazenar e recuperar as informações XML e transformar a informação em gráficos.

5 CONSIDERAÇÕES FINAIS

Neste capitulo serão apresentados considerações finais sobre o trabalho efetuado, e as sugestões para trabalhos futuros.

5.1 CONCLUSÃO

Verificou-se que a computação em nuvem é algo bem recente, nem mesmo as grandes empresas de tecnologia estão em consenso sobre o que é e o que se delimita essa tecnologia. Muitos produtos ainda sequer foram inventados, algumas empresas estão indo partindo para a virtualização de máquinas, outras desenvolvendo APIs de integração com suas plataformas.

Existe um enorme mercado para empresas de todos os tipos, algumas empresas podem terceirizar sua infraestrutura de TI para diminuir gastos. Empresas de desenvolvimento de software podem focar no desenvolvimento de produtos, sem se preocupar com a demanda, pois podem alocar dinamicamente mais poder computacional baseado no consumo.

É notavelmente fácil a criação de servidores na nuvem, em especial ao exemplo mostrado no trabalho, onde em alguns passos, utilizando máquinas configuras e sem custo, instanciar um servidor na infraestrutura da Amazon, e utilizada para hospedar os serviços web.

Empresas que pretendem migrar seu servidores para uma estrutura de computação em nuvem podem ver no produto da Amazon a melhor alternativa. Desenvolvedores com experiência em programação em Visual Studio, ou que queiram desenvolver serviços tem a opção de utilizar a plataforma da Microsoft, ou ainda as APIs da Google.

Verificou-se ainda que Web services que funcionam com software como serviço, devem disponibilizar formas de comunicação para as aplicações clientes. A chamada de procedimento remota implementada para o protocolo HTTP atende muito bem na maioria dos casos, ressaltando que não é em todos os projetos a solução mais adequada.

Desenvolvedores tem em mãos toda a infraestrutura necessária para prover aplicações *web* de grande porte e alta capacidade, para que tenham sucesso sem necessitar de grandes investimentos em infraestrutura. De um lado plataformas escaláveis e pagamento sob demanda sobre os recursos, que poderão ser pagas mediante a consumo, do outro lado os protocolos de comunicação facilmente implementáveis que promover facilidades quando se é necessário trocar informações entre sistemas distribuídos.

5.2 TRABALHOS FUTUROS

Como trabalhos futuros pode-se sugerir:

- Implementação de comunicação utilizando SOAP e REST: A aplicação desenvolvida neste trabalho focou principalmente a utilização do protocolo RPC, mas existe muitas formas de comunicação entre Web services. SOAP e REST são duas delas.

  O SOAP é um protocolo para troca de informações estruturadas em plataformas distribuídas, ele utiliza XML para o transporte das informações e baseia-se nas camadas como Chamada de Procedimento Remoto (RPC) e Protocolo de transferência de Hipertexto (HTTP).

  O REST implementa um conjunto de operações nativas do protocolo HTTP, essas operações são o POST, GET, PUT e DELETE. Cada recurso em REST teria seu próprio identificar por onde poderia ser feitos as operações GRUD.
- Implementar serviços utilizando a plataforma da Microsoft com o Windows Azure, demonstrando suas características, vantagens e desvantagens.
- Desenvolvimento de aplicações para *desktop* utilizando o *Simple Storage Service* da Amazon, que é um serviço específico para armazenamento para a *internet*, pensado para facilitar a web-escala de computação para desenvolvedores.

- Desenvolvimento de aplicações SaaS. Utilizando o software como serviço, vendendo o serviço para terceiros e lucrando em cima de uma das plataformas Cloud.

6 OBRAS CITADAS

ADMINISTRADORES. Pequenas e médias empresas ampliam investimentos em TI. www.administradores.com.br, jan. 2011. Disponivel em: <http://www.administradores.com.br/informe-se/administracao-e-negocios/pequenas-e-medias-empresas-ampliam-investimentos-em-ti/42080/>. Acesso em: 15 nov. 2011.

AMAZON. Amazon Elastic Compute Cloud. http: //aws.amazon.com, 2011. Disponivel em: <http://aws.amazon.com/ec2/>. Acesso em: 15 nov. 2011.

ARTIGOSONLINE. SaaS vs S+S. http: //www.artigosonline.com.br, 2011. Disponivel em: <http://www.artigosonline.com.br/saas-vs-ss/>. Acesso em: 05 nov. 2011.

BECKER, G. P. Programação com DCOM. http: //www.inf.ufrgs.br, jul. 1999. Disponivel em: <http://www.inf.ufrgs.br/gppd/disc/cmp167/trabalhos/sem99-1/T1/gustavo/DCOM.htm>. Acesso em: 15 nov. 2011.

BUFALOINFO. Windows Azure: O que é. http: //www.bufaloinfo.com.br, mar. 2008. Disponivel em: <http://www.bufaloinfo.com.br/exibenoticias.aspx?entryid=1165265889739981343>. Acesso em: 01 nov. 2011.

CERQUEIRA, A. O modelo de arquitetura CORBA. http://www.linhadecodigo.com.br, abr. 2004. Disponivel em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=299>. Acesso em: 27 out. 2011.

CHAGANTI, P. Computação em Nuvem com Amazon Web Services - Parte 03: servidores on demand com EC2. http: //imasters.com.br, maio 2011. Disponivel em: <http://imasters.com.br/artigo/20808/cloud/computacao-em-nuvem-com-amazon-web-services-parte-03-servidores-on-demand-com-ec2>. Acesso em: 15 nov. 2011.

CHAGANTI, P. Computação em nuvem com o Amazon Web Services - Parte 02. http: //imasters.com.br, abr. 2011. Disponivel em: <http://imasters.com.br/artigo/20587/cloud/computacao-em-nuvem-com-o-amazon-web-services-parte-02>. Acesso em: 28 out. 2011.

CODE.GOOGLE. O que é o Google App Engine? http: //code.google.com, 2011. Disponivel em: <http://code.google.com/intl/pt-BR/appengine/docs/whatisgoogleappengine.html>. Acesso em: 15 nov. 2011.

CODEGEAR. http://www.codegear.com/. http: //www.codegear.com, 2012. Disponivel em: <http://www.codegear.com/br/article/37534>. Acesso em: 01 mar. 2012.

CONVERGENCIA DIGITAL. Adesão das empresas à nuvem crescerá 1300% em quatro anos. http: //convergenciadigital.uol.com.br, fev. 2011. Disponivel em: <http://convergenciadigital.uol.com.br/cgi/cgilua.exe/sys/start.htm?infoid=25230&sid=97>. Acesso em: 15 nov. 2011.

DESTRO, D. Introdução ao RMI. http: //www.guj.com.br, abr. 2003. Disponivel em: <http://www.guj.com.br/articles/37>. Acesso em: 28 out. 2011.

FEIGENBAUM, E. Segurança em primeiro lugar: Google Apps e Google App Engine recebem a certificação SSAE-16. http: //googlebrasilblog.blogspot.com, ago. 2011. Disponivel em: <http://googlebrasilblog.blogspot.com/2011/08/seguranca-em-primeiro-lugar-google-apps.html>. Acesso em: 15 nov. 2011.

GHEDIN, R. P. afinal o que e o windows-azure. http: //www.winajuda.com, 2011. Disponivel em: <http://www.winajuda.com/2008/10/27/afinal-o-que-e-o-windows-azure/>. Acesso em: 29 out. 2011.

GUIADOHACKER. O que é RPC (Chamada de procedimento remoto). http: //forum.guiadohacker.com.br, out. 2009. Disponivel em: <http://forum.guiadohacker.com.br/showthread.php?t=4323>. Acesso em: 25 out. 2011.

INFOBLOGS. Quando EJB é apropriado. http: //www.infoblogs.com.br, jul. 2007. Disponivel em: <http://www.infoblogs.com.br/view.action?contentId=18151&Quando-EJB-e-apropriado.html>. Acesso em: 14 nov. 2011.

IWEB. webservices. http: //www.iweb.com.br, 2003. Disponivel em: <http://www.iweb.com.br/iweb/pdfs/20031008-webservices-01.pdf>. Acesso em: 01 fev. 2012.

MSDN.MICROSOFT. O modelo de maturidade SaaS. http: //msdn.microsoft.com, 2011. Disponivel em: <http://msdn.microsoft.com/pt-br/library/dd875466.aspx>. Acesso em: 05 nov. 2011.

NOVIDADESDEINFORMATICA. Sistemas Distribuidos: Oferecendo ao Usuário o que ele Precisa. http: //www.novidadesdeinformatica.com.br, 2011. Disponivel em: <http://www.novidadesdeinformatica.com.br/hardware/sistemas-distribuidos-oferecendo-usuario-ele-precisa>. Acesso em: 05 nov. 2011.

ORACLE. The Java EE 5 Tutorial. http: //docs.oracle.com, 2012. Disponivel em: <http://docs.oracle.com/javaee/5/tutorial/doc/bnbyl.html>. Acesso em: 15 fev. 2012.

POZZEBON, R. Saas - Software as a service: O que é? oficinadanet, jun. 2011. Disponivel em: <http://www.oficinadanet.com.br/artigo/business_intelligence/saas-software-as-a-service-o-que-e>. Acesso em: 05 nov. 2011.

RAFAEL, B. Afinal, O que é o EJB? http: //javanamente.blogspot.com, ago. 2010. Disponivel em: <http://javanamente.blogspot.com/2010/10/afinal-o-que-e-o-ejb.html>. Acesso em: 28 out. 2011.

SPÍNOLA, E. O. Introdução ao EJB 3.0 e o Enterprise Beans Components - Parte I. http: //www.devmedia.com.br, 2010. Disponivel em: <http://www.devmedia.com.br/post-1596-Introducao-ao-ejb-3-0-e-o-enterprise-beans-components-parte-i.html>. Acesso em: 28 nov. 2011.

SPINOLA, E. O. Introdução ao Java RMI. devmedia, 2011. Disponivel em: <http://www.devmedia.com.br/post-2012-Introducao-ao-Java-RMI-Parte-I.html>. Acesso em: 15 out. 2011.

TANENBAUM, A. S.; STEEN, M. V. Sistemas Distribuídos princípios e paradigmas. 2ª Edição. ed. São Paulo: [s.n.], 2007.

TAURION, C. Cloud Computing. Rio de Janeiro: Brasport, 2009.

TAURION, C. Cloud Computing: Google ou Amazon? http: //imasters.com.br, maio 2010. Disponivel em: <http://imasters.com.br/artigo/16971/cloud/cloud-computing-google-ou-amazon>. Acesso em: 15 nov. 2011.

UNICAMP. Arquitetura RMI. http: //www.dca.fee.unicamp.br, 2002. Disponivel em: <http://www.dca.fee.unicamp.br/cursos/PooJava/objdist/rmiarq.html>. Acesso em: 10 fev. 2012.