UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ - UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO

WILDENEY RIGO

ESTUDO COMPARATIVO DE PERFORMANCE ENTRE OS PROTOCOLOS HESSIAN E AMF EM APLICAÇÕES FLEX

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

WILDENEY RIGO

ESTUDO COMPARATIVO DE PERFORMANCE ENTRE OS PROTOCOLOS HESSIAN E AMF EM APLICAÇÕES FLEX

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná – Campus Medianeira, como requisito parcial para obtenção do título de Tecnólogo.
Orientador: Prof. *Msc.* Fernando Schütz.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso Superior de Tecnologia em
Desenvolvimento de Sistemas de Informação

TERMO DE APROVAÇÃO

Estudo comparativo de performance entre os protocolos Hessian e AMF em aplicações Flex.

Por

Wildeney Rigo

Este Trabalho de Diplomação (TD) foi apresentado às 13:00 h do dia 14 de junho de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Desenvolvimento de Sistemas de Informação, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. *Msc.*Fernando Schütz UTFPR – *Campus* Medianeira (Orientador)	Prof. *Msc.* Neylor Michel UTFPR – *Campus* Medianeira (Convidado)
Prof. Itamar Pena Nieradka UTFPR – *Campus* Medianeira (Convidado)	Prof. Juliano Rodrigo Lamb UTFPR – *Campus* Medianeira (Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Gostaria de agradecer a meus pais, por sempre me apoiarem em minhas decisões e por estarem sempre me motivando. Agradeço a minha esposa, que passou muitas noites me ajudando a finalizar este trabalho e também me motivando e me auxiliando durante todo o tempo.

Gostaria de agradecer, também, a todos os meus amigos que me incentivaram em finalizar meus estudos. Aos meus amigos de trabalho, que me auxiliaram na confecção deste projeto.

Ao meu orientador, que me guiou na finalização do trabalho, e a todos que acreditaram em mim.

Muito obrigado.

RESUMO

As aplicações ricas para internet são aplicações *Web* com características e funcionalidades de softwares do tipo *desktop*, porém, mantendo toda parte de dados em um servidor. Flex vem se destacando cada vez mais como opção de alta interatividade e na utilização de serviços *Web* para a parte de negócio, mas para garantir uma boa performance, a escolha de um protocolo que faça intermédio as aplicações Flex e os serviços *Web* é crucial para garantir uma bom desempenho da entrega de dados da parte de negócio.
Efetuando testes em diversos cenários e utilizando métricas na qual compare o protocolo AMF, sendo este um protocolo nativo do Flex e que já proporciona um bom resultado, com o protocolo Hessian, ainda não tão difundido nas aplicações Flex, mas que promete uma boa compactação dos dados.

Palavras-chave:RIA. Flex. Performance. *Web*. AMF. Hessian.

ABSTRACT

The rich internet applications are Web applications with features and functionalities like desktop softwares, however, keeping everything about data on a server. Flex is standing out as a good choice for high performance, but the choice of a protocol which makes its way through Flex applications and Web services is crucial to guarantee a good performance of data delivery on business side.
Performing tests in various scenarios and using metrics that compares the AMF protocol, a native protocol of Flex and already provides a good result, with the Hessian protocol, although not as widespread on Flex applications, but it promises a good data compression.

Keywords: RIA. Flex. Performance. Web. AMF. Hessian.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AIR	*Adobe Integrated Runtime*
AJAX	*Asynchronous Javascript and XML*
AMF	*Action Message Format*
API	*Application Programming Interface*
ASP	*Active Server Pages*
AVM	*ActionScript Virtual Machine*
B2B	*Business-to-Business*
DOM	*Document Object Model*
E4X	*ECMAScript for XML*
ECMA	*European Computer Manufacturers Association*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IBM	*International Business Machines*
IDE	*Integrated Development Environment*
JSP	*Java Server Pages*
MXML	*Magic Extensible Markup Language*
PHP	*Hypertext Preprocessor*
REST	*Representational State Transfer*
RIA	*Rich Internet Applications*
RNP	Rede Nacional de Ensino e Pesquisa
SDK	*Software Development Kit*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SWF	*Shockwave Flash*
XML	*Extensible Markeup Language*

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da Internet e a grande procura por aplicações leves, objetivas, com interfaces amigáveis, e a garantia de que funcionará em qualquer lugar, faz com que as RIAs (*Rich Internet Application*) em português, Aplicações Ricas para Internet, sejam a melhor opção.

Adobe Flex, um *framework*[1] de código aberto para construção de alta interatividade e aplicações *Web* expressivas implantadas na maioria dos navegadores conhecidos, *desktops* e sistemas operacionais, fornece uma linguagem que suporta padrões de *design* comuns e, também, inclui uma biblioteca de componentes para criação de aplicações ricas.

As aplicações Flex necessitam de serviços *Web* e para isso é necessária a utilização de alguns protocolos para realizar a transferência de dados, para tanto, vale testar o desempenho de alguns deles na compactação e também o tamanho das informações geradas por cada um deles.

O AMF (*Action Message Format*) é um formato binário compacto utilizado para serializar objetos em *ActionScript*, nativo do Flex.

O protocolo binário Hessian faz com que os serviços da *Web* sejam aproveitados sem a necessidade de um *framework* de grande porte e sem complicação, por ser um protocolo binário, adequado para o envio de dados binários, sem a necessidade de estender outros protocolos anexados. (FERGUSON, 2007).

1.1 OBJETIVOS

A seguir serão apresentados os objetivos deste projeto.

1.1.1 Geral

Realizar um estudo comparativo utilizando o *framework Flex* que realize requisições para um serviço *Web* e que avalie o desempenho dos protocolos para

[1] Segundo Horstmann (2006), *framework* é um conjunto de classes cooperativas que implementam os mecanismos essenciais para um domínio de problemas específicos.

efetuar com maior performance a transferência das informações e objetos gerando gráficos desta análise.

1.1.2 Específicos

Os objetivos específicos são:

- Desenvolver um referencial teórico fazendo uso do *framework* Flex para acesso a serviços *Web*;
- Avaliar o desempenho entre os protocolos AMF e Hessian para definir qual trará melhor desempenho para a aplicação Flex;
- Desenvolver uma aplicação simples para executar as requisições efetuadas do Flex à um servidor *Web*;
- Apresentar Gráficos que demonstram as performances de compactação, envio de dados pela *Web* e tempo de *parse*;
- Apresentação dos resultados obtidos através das comparações.

1.2 JUSTIFICATIVA

Com as interativas Aplicações Ricas para Internet (RIAs), os usuários esperam conteúdo envolvente com tempos de resposta rápidos. Mais e mais, que o conteúdo seja obtido em tempo real a partir de um serviço na rede. Para minimizar o tempo de espera e a possibilidade de perder os usuários, o tempo gasto na rede devem ser otimizados. Devido a isso, um estudo comparativo com esses dois protocolos de compactação se torna um bom referencial para desenvolvedores que precisam escolher entre qual tecnologia utilizar.

1.3 ORGANIZAÇÃO DA MONOGRAFIA

O presente trabalho está dividido em cinco capítulos organizados da forma descrita a seguir.

O primeiro capítulo inicia com a introdução, que aborda brevemente sobre os protocolos e serviços *Web*, seguindo pelos objetivos e a justificativa que levaram a realização deste trabalho.

No segundo capítulo, pode-se verificar todo o referencial teórico no qual este trabalho foi baseado. O referencial teórico mostra as definições de RIAs e as tecnologias que envolvem este tipo de aplicação, como os *Web Services*, *middlewares* e linguagens de programação.

Os materiais e métodos utilizados podem ser vistos no capítulo três que descreve as estruturas, física e lógica, utilizadas para a construção dos testes propostos para a amostragem visualizada no capítulo quatro.

O capítulo cinco apresenta a conclusão e as sugestões de trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo tem por objetivo analisar os subsídios teóricos utilizados na estruturação deste estudo. As aplicações ricas para internet são aplicações *Web* com características e funcionalidades de softwares do tipo *desktop*, porém, mantendo toda parte de dados em um servidor.

2.1 RIA – RICH INTERNET APPLICATION

Com o super crescimento da Internet nos anos 90, surge um novo modelo de aplicações trabalhando em um navegador agindo como um *thin client*[2], tendo como função renderizar HTML (*Hypertext Markup Language* – Linguagem de Marcação de Hipertexto) e enviar requisições de volta a um servidor que entregasse as páginas de forma dinâmica, solucionando o problema de distribuição dos tempos de cliente/servidor tendo através de *download* a aplicação do servidor cada vez que o usuário final necessitasse, sendo assim, as atualizações podiam ser feitas em um único local centralizado e automaticamente distribuída para toda a base de usuários.

Quando as aplicações de internet começam ser utilizadas para o *core business*[3], a sustentabilidade das aplicações fica crucial, tendo relacionamento direto à sua arquitetura. Muitas dessas aplicações foram construídas com pouco conhecimento aos princípios de arquiteturas de aplicações, dificultando a manutenção e atualizações. Hoje em dia é fácil construir uma arquitetura sólida para uma aplicação, separando a área de negócio do acesso de dados e apresentação. RIAs (*Rich Internet Application*, que significa em português Aplicações Ricas para Internet), são executadas a partir de um navegador não sendo necessitado instalar, rodando localmente em um ambiente seguro também conhecido como *Sandbox* (ADOBE, 2008).

Com a introdução de elementos como *Web Services*, o conceito de uma arquitetura orientada a serviços (SOA, em inglês *Service-oriented architecture*) tornou-se mais viável para aplicações baseadas na *Web*.

Assim, segundo Tapper, Labriolla, Boles e Talbot (2009), as RIAs são capazes de:

---

[2] Clientes com poucos comandos, comuns a “todos” os navegadores e que por isto funcionam em qualquer plataforma de hardware e de software (Cruz, 2006).
[3] Parte central de um negócio ou uma área de negócios.

- Permitir um tempo de execução eficiente e de alta performance para a execução de código, conteúdo e comunicações;
- Fornecer modelos de objetos eficazes e extensíveis para facilitar a interatividade;
- Possibilitar o uso de objetos Server-side via *WebServices* ou outras tecnologias do gênero.

Tendo como uma das metas principais, reduzir a quantidade de informações extras, transmitidas com cada solicitação, são capazes de oferecer ao usuário funcionalidades que não podem ser feitas apenas em HTML, a interface é mais reativa a ações do usuário do que em aplicações *Web* padrão, tendo uma sensação de estar usando uma aplicação *Desktop*.

Equilíbrio entre carga de processamento entre o cliente e o servidor permitindo que o mesmo servidor possa lidar com vários clientes. O lado cliente pode interagir com o servidor de forma assíncrona, sendo uma ação sendo realizada na interface não necessite esperar uma resposta do servidor.

Seu fluxo na rede é significativamente reduzido, sendo somente os dados sendo transmitidos. Entretanto, o uso destas técnicas pode neutralizar, ou até mesmo reverter o potencial deste benefício, isso porque o código não pode prever exatamente o que cada usuário irá fazer logo em seguida (NET,2009).

2.1.1 Deficiências e Restrições

Algumas RIAs que são executadas em um Sandbox, podem restringir acessos a recursos do sistema. O tempo de carregamento da aplicação também é um obstáculo, embora as aplicações não necessitem ser instaladas (ADOBE, 2008).

O usuário precisa efetuar no mínimo uma vez o *download* da aplicação, para isso pode-se reduzir este impacto com compactação de *scripts* e um carregamento progressivo. RIAs também tem sua dependência por uma conexão de Internet, isso porque a aplicação requer que fique permanentemente conectado à rede.

2.1.2 Adobe Flex

De acordo com Schmitz (2009), Flex é um *framework open source* para o desenvolvimento de aplicações *Web* que serão acessadas por meio de um navegador *Web*, cujo diferencial é que as aplicações criadas podem utilizar interfaces ricas, com diversos recursos e componentes, além de funcionalidades como “arrastar e soltar”, mover objetos, transparência, entre outros, aperfeiçoando a experiência com o usuário e tornando-a a melhor possível.

Programadores de aplicações tradicionais encontraram um desafio para adaptar-se sobre a plataforma Adobe Flash, da qual era originalmente designado à metáfora de animação. Flex procura minimizar este problema provendo um fluxo de trabalho e modelo de programação que é familiar para esses desenvolvedores. MXML oferece uma maneira de construir e elaborar interfaces gráficas. Interatividade é obtida através do uso do *ActionScript*.

O *framework* Flex facilita o desenvolvimento de sistemas, além da possibilidade de se construir outras aplicações com diversos recursos. Por ser uma ferramenta *open source* pode-se usá-lo na criação de aplicações comerciais.

Pode-se dizer que, o Flex, é uma poderosa ferramenta de apresentação, pois é a parte visual do sistema. Toda a parte de lógica, principalmente na persistência dos dados com o banco de dados, deverá ser realizada por uma linguagem de servidor, como PHP (*Hypertext Preprocessor*), Java, Coldfusion, ASP.NET, Ruby on Rails, entre outras.

Em termos de arquitetura, aplicações Flash são similares a aplicações AJAX em que ambas são capazes de atualizações dinâmicas à interface do usuário e ambas incluem habilidade de enviar e carregar dados em *background* (TAPPER; LABRIOLA; BOLES; TALBOT, 2009).

Segundo Tapper, Labriola, Boles e Talbot (2009), o Flex proporciona a próxima geração de ferramentas e serviços de desenvolvimento que possibilita aos desenvolvedores de todo o mundo construir e implantar RIAs na plataforma Flash.

O Flex SDK (*Software Development Kit* – Kit de Desenvolvimento de *Software*) vem com um conjunto de componentes de interface do usuário, incluindo botões, caixas de listagem, árvores, grades de dados, controles de texto diversos e vários recipientes de *layout*.

O Flex está fundamentado na utilização de duas linguagens de programação, que são a base para todo o desenvolvimento de sistemas e são chamadas de MXML (*Magic Extensible Markup Language,* em português significa Linguagem Mágica de Marcação Extensível) e *ActionScript*. Na Figura 1, que representa a arquitetura do Flex, podem ser visualizadas as duas linguagens de programação citadas.

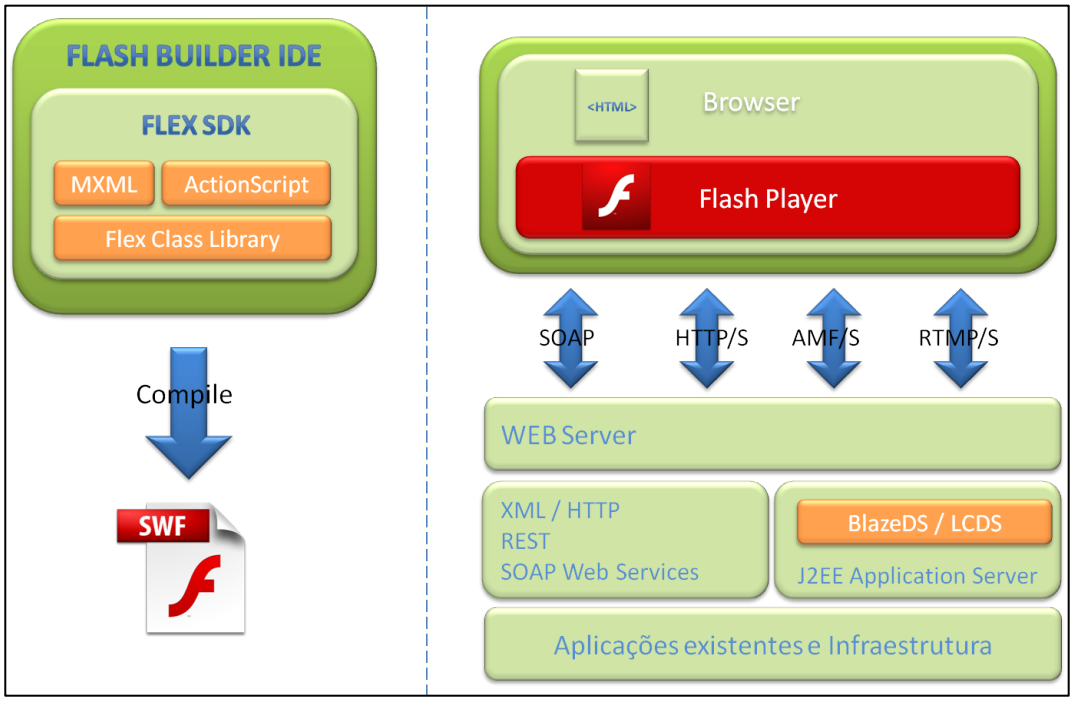


Figura 1 – Arquitetura Flex
Fonte: Adaptado de Dimyati (2008).

Para Adobe (2008), MXML é uma linguagem de marcação de texto baseada em XML utilizada para criar os componentes de interface do usuário. Esta linguagem de marcação, também pode ser usada para definir aspectos não-visuais de uma aplicação, tais como o acesso a fontes de dados sobre as ligações de servidor e de dados entre componentes de interface e fontes de dados no servidor.

Assim como o HTML, MXML fornece *tags* que definem interfaces de usuário. No entanto, esta linguagem é mais estruturada do que HTML, e fornece um conjunto de marcas muito mais extenso. Por exemplo, MXML inclui marcas de componentes visuais, tais como grades de dados, as árvores, os navegadores guia e menus, bem como componentes não visuais que fornecem conexões de *WebService*, ligação de dados, e efeitos de animação.

Além disso, uma das maiores diferenças entre as duas linguagens de marcação, mencionadas anteriormente, é que as aplicações MXML definidas são compiladas em arquivos SWF (*Shockwave Flash)* e executados pelo Adobe Flash

Player, para aplicações *Web,* ou Adobe AIR, para aplicações *Desktop.* MXML também suporta componentes personalizados escritos em arquivos MXML e *ActionScript* (ADOBE, 2008).

2.2 WEB SERVICES

Web Service é uma solução usada para integração de sistemas e comunicação entre diferentes aplicações que permite construir serviços capazes de serem interoperáveis em uma rede. Não dependem de nenhum protocolo específico para a troca de informações, embora, na prática seja comumente usado o HTTP (*Hypertext Transfer Protocol*) em conjunto com XML (*Extensible Markeup Language*) (ERL, 2005 apud QUINGERSKI, 2008).

XML pode ser entendida como uma linguagem de marcação criada para descrever, armazenar e manipular dados estruturados. Por ser uma linguagem extensível, não se limita a certo número de tags, podendo ser criadas novas tags assim que necessário.

Esta tecnologia pode ser utilizada de diversas maneiras. *Web Services* podem ser executados em cliente desktop e portáteis para acessar aplicativos de Internet e, também, para *business-to-business* (B2B), permitindo que vários clientes acessem uma aplicação em um mesmo momento (NEWCOMER, 2002).

Para executar os *Web* Services é necessária a utilização de formatos de dados para realizar a compactação e serialização dos objetos.

2.2.1 AMF – Action Message Format

Action Message Format (AMF) é um formato binário compacto que é usado para serializar objetos *ActionScript*. Uma vez serializado, o objeto codificado pode ser utilizado para persistir e recuperar o *public state* de uma aplicação através das sessões ou permitir que dois *endpoints* se comuniquem através da troca de dados com tipos específicos (ADOBE, 2006).

A primeira versão do AMF, conhecido como AMF 0, suporta o envio de objetos complexos por referência que ajuda a evitar o envio de instancias redundantes em um objeto. O AMF também permite restaurar os relacionamentos e referências circulares, evitando problemas como a recursividade infinita durante a

serialização. Uma nova versão do AMF, conhecido como AMF 3 para coincidir com o lançamento do *ActionScript* 3.0, melhora o AMF 0 enviando características de objetos e strings de referência, além de objeto de instâncias (ADOBE, 2006).

2.2.2 Hessian

De acordo com Ong (2010), *Hessian* é um protocolo binário *open source*, rápido e eficiente com inúmeras implementações de código aberto que resolve os problemas da rede de serviços enfrentados por desenvolvedores de Flash e Flex.

Este protocolo permite receber eventos de *streaming* a partir do servidor da mesma forma que o Flex recebe qualquer outro evento. O Hessian é uma alternativa de *Web Service* que oferece suporte para Java, C#, Ruby, PHP e Python, além do apoio *ActionScript*.

Para Ferguson e Ong (2007), Hessian é dinamicamente tipado, compacto e versátil em várias linguagens. É um protocolo que tem os seguintes objetivos:

- É preciso auto-descrever os tipos serializados, ou seja, não requerem esquema externo ou definições de interface;
- Ele deve ser de linguagem independente, incluindo o suporte de linguagens de script;
- Deve ser bem legíveis e bem escritos;
- Deve ser o mais compacto possível;
- Deve ser simples para que ele possa ser efetivamente testado e implementado;
- Deve ser o mais rápido possível;
- Deve apoiar cadeias de caracteres *Unicode*;
- Ele deve suportar dados binários de 8 bits sem sair disso ou usar acessórios;
- Tem suporte a encriptação, compressão, assinatura e envolver contexto de transação.

A maioria dos protocolos de plataformas é baseada em XML e, portanto, sacrifica uma quantidade significativa de desempenho, a fim de assegurar a interoperabilidade. A vantagem do Hessian é que consegue interoperabilidade entre plataformas com o mínimo de degradação do desempenho.

2.3 MIDDLEWARE

Middleware é o termo utilizado para designar as camadas de *software* que não constituem diretamente aplicações, mas que facilitam o uso de ambientes ricos em tecnologia da informação. Esta camada concentra serviços de identificação, autenticação, autorização, diretórios, certificados digitais, entre outras ferramentas para segurança (RNP, 2006).

Vários *middlewares* são utilizados para o desenvolvimento de aplicações do tipo RIA. Para a integração Java-Flex, pode-se utilizar o BlazeDS.

Conforme Adobe (2009), BlazeDS é um servidor baseado em Java *Remoting* e tecnologia *Web* de mensagens que permite aos desenvolvedores conectarem-se facilmente ao *back-end* de dados distribuídos e envio de dados em tempo real para o Adobe Flex e aplicativos Adobe AIR.

Algumas aplicações executadas no *Desktop* com o Adobe AIR ou no navegador com Flash Player, normalmente são conectadas a um servidor quando se faz necessário o carregamento ou a manipulação de dados.

Independente da tecnologia do lado do servidor, algum tipo de comunicação em rede é necessário quando o aplicativo cliente precisa se comunicar com o servidor. Existem várias formas de realizar essa comunicação, geralmente acontece através de HTTP, por ser um método simples e de fácil configuração, mas o seu uso pode reduzir o desempenho do aplicativo (WARD; TIWARI, 2008).

Por algum tempo, o Flash Player apoiou o AMF como protocolo de transporte que alivia os congestionamentos desnecessários associados a protocolos baseados em texto.

O BlazeDS inclui uma implementação Java de AMF que é usado para comunicação remota com os objetos do servidor Java, bem como para a passagem de mensagens entre clientes.

Como pode ser visualizado na Figura 2, o BlazeDS é um conjunto de servlets e ouvintes que pode ser adicionado na aplicação *Web*J2EE para ter acesso a camada de serviço através do protocolo AMF.

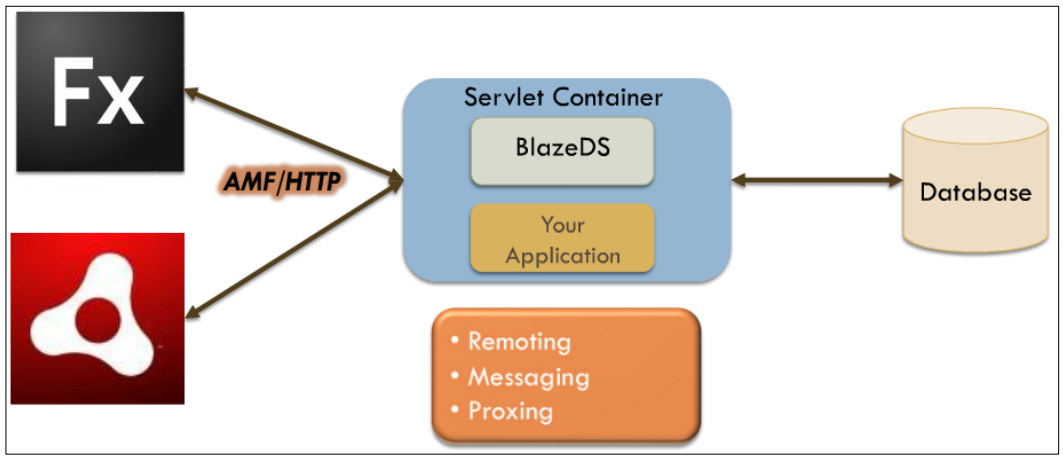


Figura 2 – Visão geral do BlazeDS
Fonte: Luu (2009).

2.4 LINGUAGENS DE PROGRAMAÇÃO

Um dos principais objetivos das linguagens de programação é permitir que os desenvolvedores trabalhem com alta produtividade, expressando as intenções com facilidade. As linguagens de programação são esquematizadas para adotar uma sintaxe de alto nível facilitando seu entendimento por programadores.

Existem várias linguagens de programação sendo utilizadas para o desenvolvimento de aplicações do tipo RIA. As seções seguintes apresentam algumas destas tecnologias que serão utilizadas no decorrer deste trabalho.

2.4.1 Java

Java é uma linguagem de programação orientada a objetos, desenvolvida inicialmente para ser a linguagem-base de projetos de *software* para produtos eletrônicos. É uma linguagem de alto nível, simples e fortemente tipada e com diversas características herdadas de outras linguagens.

A linguagem é muito parecida com C++, mas muito mais simples, pois não possui sobrecarga de operadores, herança múltipla, diretivas de pré-processamento e a memória alocada dinamicamente é gerenciada pela própria linguagem, que usa algoritmos de *garbage collector* para desalocar regiões de memória que não estão mais em uso (SILVEIRA, 2003).

2.4.2 ActionScript

ActionScript é uma linguagem de programação orientada a objeto da Adobe Flash *Plataform* e criada com base em ECMAScript, sendo utilizada principalmente para construir aplicações RIA. É uma linguagem, originalmente, criada para desenvolver conteúdo interativo desde animações simples a interfaces interativas, com riqueza de dados e aplicações em *browsers*, *desktops* e dispositivos móveis.

Remoaldo (2008) afirma que, devido a problemas de compatibilidade entre o JavaScript e o JScript, a ECMA (*European Computer Manufacturers Association),* uma indústria sem fins lucrativos, que está envolvida na normalização das tecnologias de informação e comunicação, com o acordo da Netscape e da Microsoft, resolveu criar uma norma de nome ECMA-262 e que incorpora elementos tanto do JavaScript como do JScript. A linguagem passou então a ser conhecida por ECMAScript.

De acordo com a Adobe (2009), as versões do *ActionScript* ofereceram a flexibilidade necessária para criar verdadeiramente experiências online. Com o *ActionScript* 3.0 essa experiência avança ainda mais, proporcionando excelente desempenho e facilidade de desenvolvimento para criar aplicações altamente complexas, grandes conjuntos de dados e bases de código reutilizáveis.

ActionScript 3.0 é totalmente compatível com a especificação ECMAScript *Language*, *Third Edition* (ECMA-262). Ele também contém a funcionalidade baseada no trabalho em curso sobre o ECMAScript Edition 4, ocorrendo dentro do corpo de padrões ECMA (*European Computer Manufacturers Association)*.

ActionScript é executado pela *ActionScript Virtual Machine* (AVM), construído no Flash Player. AVM1, a máquina virtual usada para executar código legado *ActionScript*, e as atribuições do Flash Player torna possível um amplo leque de mídias interativas e aplicações ricas para Internet.

Ao longo do tempo, os desenvolvedores começaram a forçar a AVM1 aos seus limites e seus requisitos de projeto estavam necessitando de um grande avanço. *ActionScript* 3.0 introduziu uma nova e otimizado *ActionScript Virtual Machine*, AVM2, que ultrapassa desempenho da AVM1. Como resultado, o código do *ActionScript* 3.0 executa até 10 vezes mais rápido que o código legado *ActionScript* (ADOBE, 2009).

Segundo a Adobe (2009), *ActionScript* 3.0 possui os seguintes requisitos:

- Segurança: é possível escrever código de fácil manutenção e sem ambigüidades;
- Simplicidade: A linguagem é bastante intuitiva para que os desenvolvedores sejam capazes de ler e escrever sistemas sem que precise de referências;
- Desempenho: os desenvolvedores podem escrever programas complexos de forma eficiente e responsável;
- Compatibilidade: A linguagem compatibilidade e uma significativa sobreposição com os padrões da indústria. *ActionScript* 3.0 é um dialeto do ECMAScript, que formaliza os recursos do *ActionScript* 2.0, adiciona as capacidades do ECMAScript para XML (E4X), e unifica o idioma em um todo coerente.

ActionScript 3.0 é composto por duas partes, a linguagem principal, que define os elementos básicos da linguagem de programação, tais como declarações, expressões, condições, loops e tipos e a API (*Application Programming Interface*) que é composta de classes que representam e dão acesso a funcionalidade do Flash Player específicas.

3 MATERIAIS E MÉTODOS

O ambiente experimental utilizado se baseia em algumas ferramentas *open source* necessárias para a realização deste trabalho e, também, de ferramentas proprietárias.

3.1 ESTRUTURA FÍSICA

O ambiente foi testado em uma única máquina com um processador AMD Athlon II X2 250 3.00 GHz, 2 GB de memória RAM, por ser a disponível no momento dos testes. O uso de uma máquina diferente pode implicar em diferentes resultados principalmente no que se refere a processamento.

3.2 ESTRUTURA LÓGICA

A estrutura lógica consiste em expor o sistema operacional utilizado, bem como os aplicativos necessários para a realização dos testes.

3.2.1 Sistema Operacional

Os testes foram realizados com o sistema operacional Windows 7 Ultimate do tipo 32 bits.

3.2.2 Aplicativos

Para a criação de RIAs, existem vários aplicativos que podem ser utilizados para a sua codificação. No caso de aplicações ricas para internet desenvolvidas com o *framework* Flex, os mais utilizados são o Flash *Builder* e o Eclipse, aplicativos apresentados nos tópicos que se seguem.

3.2.2.1 Flash Builder

O *software* Adobe Flash *Builder*, formalmente Adobe Flex *Builder*, é uma ferramenta baseada no Eclipse para construir expressivas aplicações móveis, *Web* e *Desktop* utilizando *ActionScript* e o *framework opensource* Flex.

Para realizar a implementação deste trabalho foi utilizada a versão 4.5.

3.2.2.2 Eclipse

Eclipse é uma IDE desenvolvida em Java, com código aberto, para a construção de programas de computador. O projeto Eclipse foi iniciado na IBM (*International Business Machines*) que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade.

A versão utilizada para este trabalho foi a *Helios Service Release* 1.

3.2.3 Servidores

Devido a sua facilidade de uso e grande popularidade entre os desenvolvedores de aplicações RIA, o TomCat foi selecionado como servidor da aplicação testada neste trabalho.

O TomCat é um servidor de aplicações *Web* escrito em Java, distribuído como *software* livre e desenvolvido como código aberto, sendo um subprojeto da Jakarta Apache Foundation (MARCELO, 2005).

A principal característica deste servidor de aplicações é o foco em aplicações de Servlets e *Java Server Pages* (JSP). O TomCat é robusto e eficiente o suficiente para ser utilizado mesmo em um ambiente de produção.

A versão utilizada para a realização dos testes foi a 7.0.

3.3 IMPLEMENTAÇÃO

Esta seção visa explorar os detalhes da codificação dos testes realizados de acordo com a proposta vista anteriormente em outros capítulos, com o objetivo de avaliar, com três métricas escolhidas, os protocolos em questão.

A implementação foi realizada utilizando dois projetos, um implementado em Java, onde constava a parte servidora do sistema, e outro em Flex, para apresentação dos dados.

Para a realização dos testes foram analisadas as seguintes métricas:

- Tamanho de dados: A métrica de tamanho de dados consiste em avaliar o pacote gerado pelo protocolo em Kilobytes, em cenários com variados números de dados para a transmissão;
- Tempo de transmissão: No tempo de transmissão foi medido o tempo que o objeto já com um tipo definido, é codificado para o protocolo, somado com o tempo de resposta que o servidor envia os dados ao cliente. O tempo de codificação foi adicionado nessa métrica devido ao pacote passar informações do tempo no servidor nos dados já codificados; e
- Tempo de Decodificação: será o tempo em que o protocolo decodifica o pacote compactado binário para um objeto de tipo definido.

Como o objetivo é testar o desempenho dos protocolos, não foi utilizado um banco de dados. Os objetos para acesso foram construídos na instanciação do servidor.

3.3.1 Codificação

Para a codificação, foi necessário entender os protocolos detalhadamente, estudando o código-fonte dos protocolos, onde se encontrava o código para a compressão, em Java, (visualizado nos Quadros 1 e 2) e descompressão, em Flex, (visualizado nos Quadros 3 e 4) dos dados.

```
//Faz o parser para AMF.
response.setHeader("Content-Type", "application/x-amf");
ServletOutputStream out = response.getOutputStream();

ActionMessage requestMessage = new ActionMessage(MessageIOConstants.AMF3);
MessageBody amfMessage = new MessageBody();
amfMessage.setData( benchmarkData );
requestMessage.addBody(amfMessage);

AmfMessageSerializer amfMessageSerializer = new AmfMessageSerializer();
amfMessageSerializer.initialize
     (SerializationContext.getSerializationContext(), out, new AmfTrace());
amfMessageSerializer.writeMessage(requestMessage);

out.close();
```

Quadro 1 - Código da serialização do servidor no protocolo AMF.
Fonte: Autoria própria.

```
//Faz o parser para Hessian.
ServletOutputStream out = response.getOutputStream();
HessianSerializerOutput output = new HessianSerializerOutput(out);
output.writeObject(benchmarkData);
output.close();
out.close();
```

Quadro 2 - Código da serialização do servidor no protocolo Hessian.
Fonte: Autoria própria.

```
const packet:AMFPacket = AMFDecoder.decodeResponse(loader.data) as AMFPacket;
const obj:BenchmarkData = packet.messages[0].body;
```

Quadro 3 - Código da decodificação no cliente do protocolo AMF.
Fonte: Autoria própria.

```
var input:Hessian2Input = new Hessian2Input( ByteArray(loader.data) );
var obj:BenchmarkData = BenchmarkData(input.readObject(BenchmarkData));
```

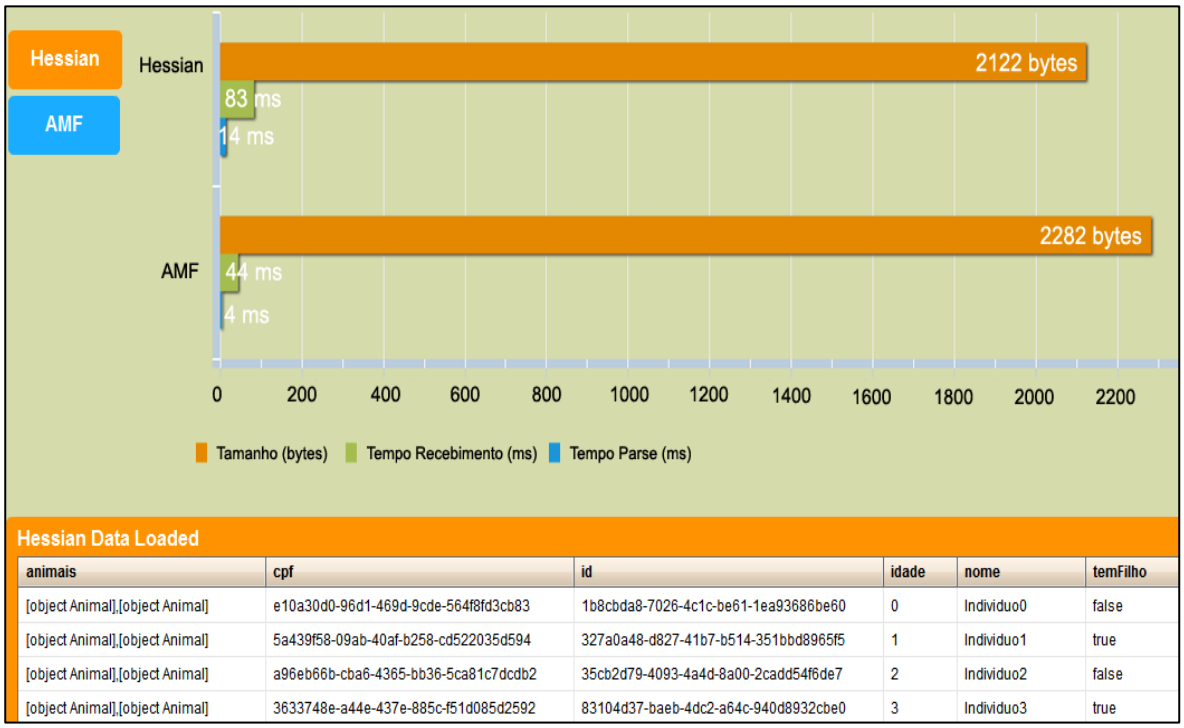
Quadro 4 - Código da decodificação no cliente do protocolo Hessian.
Fonte: Autoria própria.

O teste consiste em invocações remotas para um método que retorna um objeto que contem uma lista de dados com tipos específicos, um número do tipo *long* que contem o tempo antes dos dados serem empacotados no protocolo.O tipo dos dados que estão dentro da lista são: três do tipo *String*, um do tipo inteiro, outro do tipo *boolean* e uma lista. Os objetos desta última lista citada contêm três dados do tipo *String*. Os dados retornados são estáticos, não havendo, assim, sobrecarga na construção.

4 RESULTADOS

Os testes foram realizados nos seguintes cenários:

- Comparação de tempo de requisição/resposta, tempo de parse, e tamanho do pacote com uma lista contendo 10 dados (Figura 3);
- Comparação de tempo de requisição/resposta, tempo de parse, e tamanho do pacote com uma lista contendo 100 dados (Figura 4);
- Comparação de tempo de requisição/resposta, tempo de parse, e tamanho do pacote com uma lista contendo 1000 dados (Figura 5);
- Comparação de tempo de requisição/resposta, tempo de parse, e tamanho do pacote com uma lista contendo 10000 dados (Figura 6).

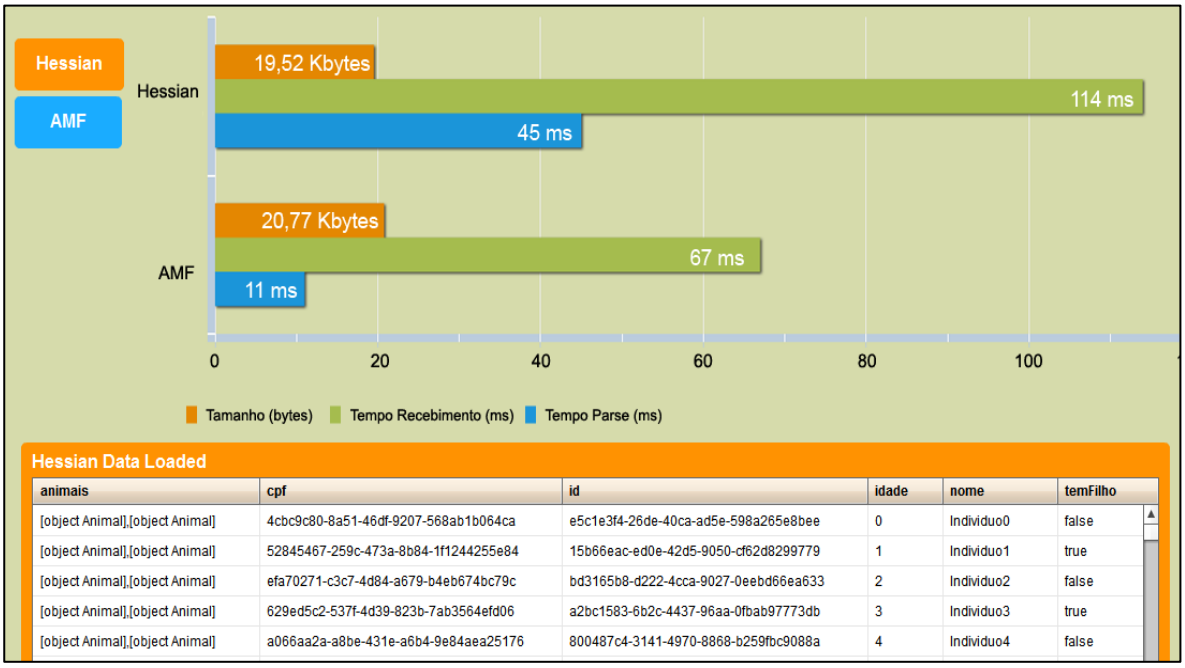


Hessian Data Loaded

animais	cpf	id	idade	nome	temFilho
[object Animal],[object Animal]	e10a30d0-96d1-469d-9cde-564f8fd3cb83	1b8cbda8-7026-4c1c-be61-1ea93686be60	0	Individuo0	false
[object Animal],[object Animal]	5a439f58-09ab-40af-b258-cd522035d594	327a0a48-d827-41b7-b514-351bbd8965f5	1	Individuo1	true
[object Animal],[object Animal]	a96eb66b-cba6-4365-bb36-5ca81c7dcdb2	35cb2d79-4093-4a4d-8a00-2cadd54f6de7	2	Individuo2	false
[object Animal],[object Animal]	3633748e-a44e-437e-885c-f51d085d2592	83104d37-baeb-4dc2-a64c-940d8932cbe0	3	Individuo3	true

Figura 3 – Resultado do cenário de teste com uma lista de 10 objetos.

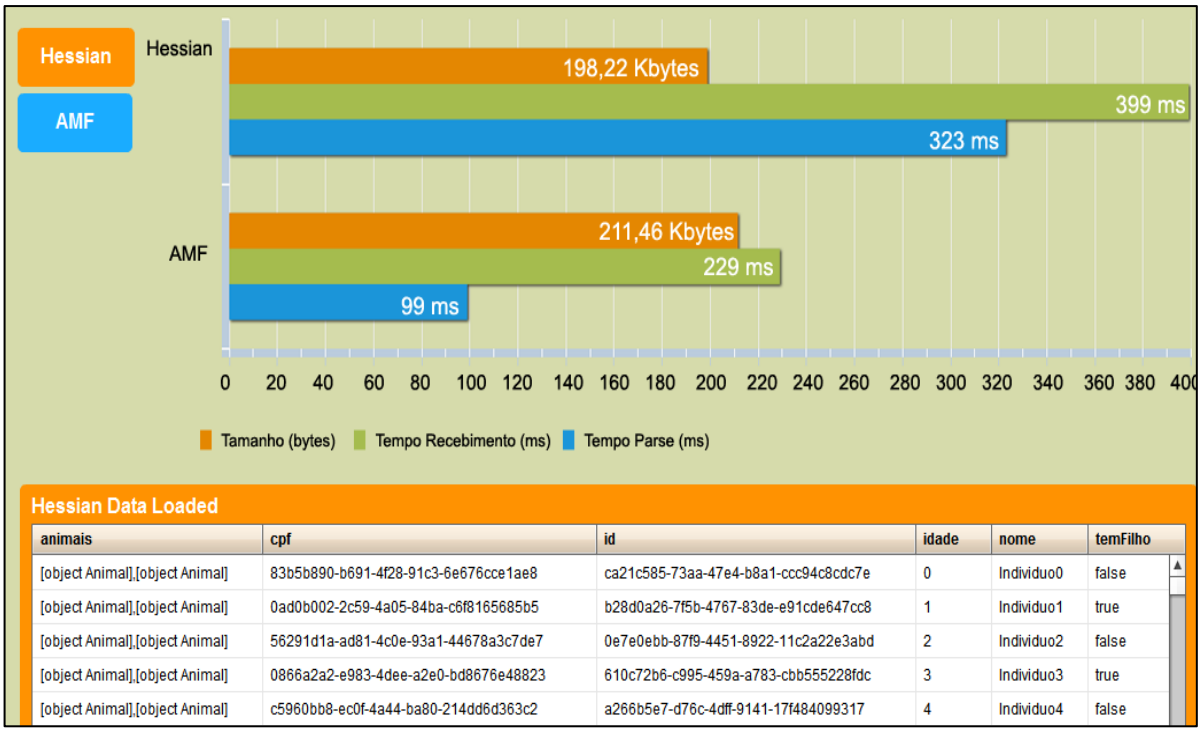
Analisando a Figura 3, pode-se verificar que a compactação dos dados com o protocolo Hessian foi melhor, mas seu tempo de *parse* teve uma pequena desvantagem de 10 milissegundos.



animais	cpf	id	idade	nome	temFilho
[object Animal],[object Animal]	4cbc9c80-8a51-46df-9207-568ab1b064ca	e5c1e3f4-26de-40ca-ad5e-598a265e8bee	0	Individuo0	false
[object Animal],[object Animal]	52845467-259c-473a-8b84-1f1244255e84	15b66eac-ed0e-42d5-9050-cf62d8299779	1	Individuo1	true
[object Animal],[object Animal]	efa70271-c3c7-4d84-a679-b4eb674bc79c	bd3165b8-d222-4cca-9027-0eebd66ea633	2	Individuo2	false
[object Animal],[object Animal]	629ed5c2-537f-4d39-823b-7ab3564efd06	a2bc1583-6b2c-4437-96aa-0fbab97773db	3	Individuo3	true
[object Animal],[object Animal]	a066aa2a-a8be-431e-a6b4-9e84aea25176	800487c4-3141-4970-8868-b259fbc9088a	4	Individuo4	false

Figura 4 - Resultado do cenário de teste com uma lista de 100 objetos.

Na Figura 4 pode-se notar que o protocolo AMF obteve uma performance excelente no quesito de desserialização dos dados, mas sua compactação mostrou-se inferior a do protocolo Hessian.



animais	cpf	id	idade	nome	temFilho
[object Animal],[object Animal]	83b5b890-b691-4f28-91c3-6e676cce1ae8	ca21c585-73aa-47e4-b8a1-ccc94c8cdc7e	0	Individuo0	false
[object Animal],[object Animal]	0ad0b002-2c59-4a05-84ba-c6f8165685b5	b28d0a26-7f5b-4767-83de-e91cde647cc8	1	Individuo1	true
[object Animal],[object Animal]	56291d1a-ad81-4c0e-93a1-44678a3c7de7	0e7e0ebb-87f9-4451-8922-11c2a22e3abd	2	Individuo2	false
[object Animal],[object Animal]	0866a2a2-e983-4dee-a2e0-bd8676e48823	610c72b6-c995-459a-a783-cbb555228fdc	3	Individuo3	true
[object Animal],[object Animal]	c5960bb8-ec0f-4a44-ba80-214dd6d363c2	a266b5e7-d76c-4dff-9141-17f484099317	4	Individuo4	false

Figura 5 - Resultado do cenário de teste com uma lista de 1000 objetos.

Com o volume de dados ampliado a diferença entre os protocolos torna-se mais evidente, nas Figuras 5 e 6 os gráficos mostram que a compactação do Hessian é mais proveitosa, mas seu tempo de *parse* permanece em desvantagem perante ao protocolo AMF.

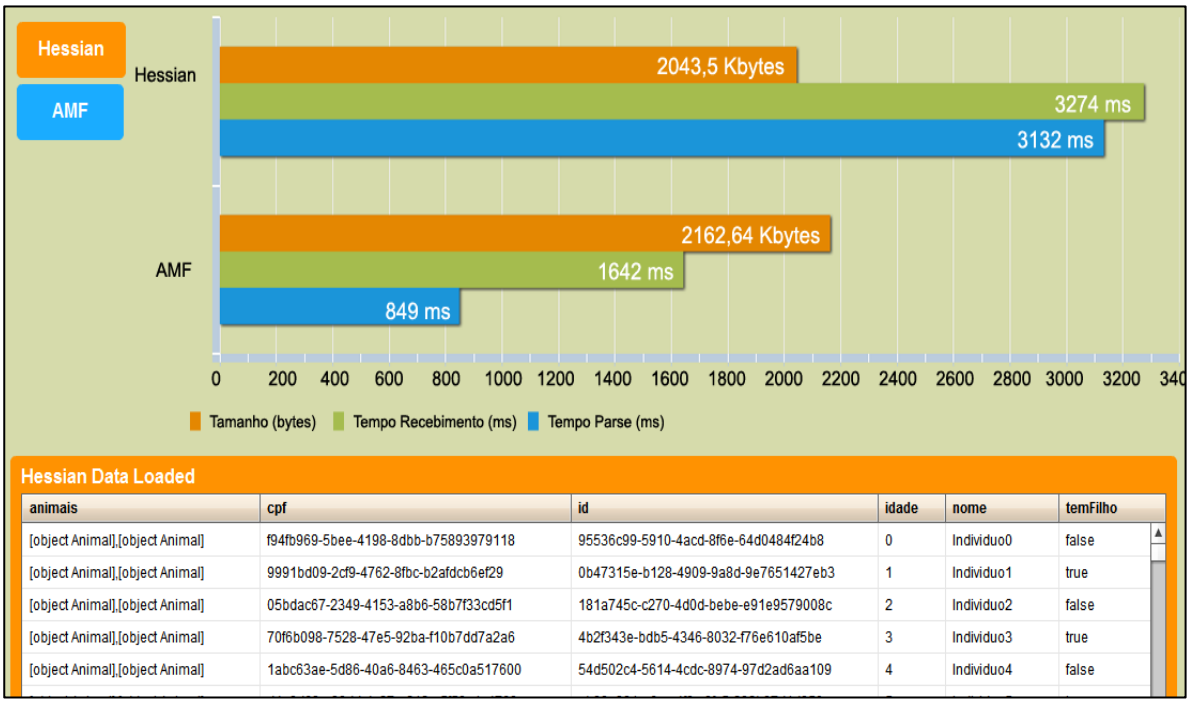


Figura 6 -Resultado do cenário de teste com uma lista de 10000 objetos.

Tabela 1 – Dados obtidos através dos testes com quatro cenários

	Hessian			AMF		
Cenários	Tamanho	TR (ms)	TP (ms)	Tamanho	TR (ms)	TP (ms)
10	2122 bytes	83	14	2282 bytes	44	4
100	19,52 Kb	114	45	20,77 Kb	67	11
1000	198,22 Kb	399	323	211,46 Kb	229	99
10000	2043,5 Kb	3274	3132	2162,64 Kb	1642	849

Fonte: Autoria própria.

Na Tabela 1, podem ser visualizados os resultados obtidos nos quatro cenários onde foram realizados os testes, sendo TR o tempo de recebimento e TP o tempo de *parse*, ambos em milissegundos (ms). Apresentando as comparações efetuadas de forma sucinta, nela temos a percepção da diferença entre os volumes dos dados comparados.

5 CONSIDERAÇÕES FINAIS

Neste capítulo estão descritos a conclusão da pesquisa e testes realizados e os trabalhos futuros.

5.1 CONCLUSÃO

Este trabalho de conclusão de curso apresentou os protocolos de serviços *Web*, *Web Services* utilizados para a comunicação entre esses protocolos, finalizando com a realização de um teste e a apresentação do resultado do mesmo.

Com os testes aqui apresentados, foi possível comparar os protocolos AMF e Hessian e concluir que, devido ao tempo de *parse* do AMF, este se mostrou ser mais rápido e proveitoso, pois mesmo o Hessian tendo um melhor desempenho na compactação, seu tempo de *parse* é muito inferior ao do protocolo AMF, suas diferenças se apresentam claramente quando o volume de dados é maior.

Através deste estudo e dos testes realizados, foi possível verificar que o uso do protocolo AMF se torna muito mais viável.

5.2 TRABALHOS FUTUROS

Utilizando-se da pesquisa e dos testes realizados neste trabalho, visar a criação de um novo protocolo que possa superar os três quesitos avaliados neste trabalho.

6 REFERÊNCIAS

ADOBE. **Visão geral do Flex**. Disponível em <http://www.adobe.com/br/products/flex/overview/>. Acesso em 20 ago. 2010, 15:15.

ADOBE. **Adobe Flex.** Disponível em <http://www.adobe.com/products/flex/>. Acesso em 20 fev. 2011, 11:23.

ADOBE. **Using WebService components.** Disponível em <http://livedocs.adobe.com/flex/3/html/help.html?content=data_access_3.html>. Acesso em 23 nov. 2010, 12:37.

_____. **Developing Applications in MXML.** Disponível em <http://livedocs.adobe.com/flex/3/html/help.html?content=mxml_2.html>. Acesso em 16 mai. 2011, 13:15.

_____. **ActionScript 3.0 overview.** Disponível em <http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html>. Acesso em 16 mai. 2011, 14:01.

_____. **Action Message Format – AMF 3.** Disponível em<http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf>.Acesso em 18 mai. 2011, 20:12.

_____. **ActionScript Technology Center.** Disponível em <http://www.adobe.com/devnet/actionscript.html>. Acesso em 30 ago. 2010, 15:32.

_____. **BlazeDS** Disponível em <http://opensource.adobe.com/wiki/display/blazeds/BlazeDS>. Acesso em 20 abr. 2011, 01:03.

CRUZ, Tadeu. **Uso e Desuso de Sistemas de Workflow.** E-papers, 2006.

DIMYATI. Irsan. **Adobe Flex Overview.** Disponível em <http://irsandimyati.wordpress.com/2008/11/15/adobe-flex-3-overview/>. Acesso em 23 mai. 2011, 12:35.

FERGUSON, Scott. **Hessian 2.0 Web Services Protocol.** Disponível em <http://hessian.caucho.com/doc/hessian-ws.html>. Acesso em 28 de novembro de 2010, 01:55.

FERGUSON, Scott; ONG, Emil. **Hessian 2.0 Serialization Protocol.** Disponível em <http://hessian.caucho.com/doc/hessian-serialization.html>. Acesso em 25 mai. 2011, 02:14.

HORSTMANN, Cay. **Padrões e Projeto Orientados a Objetos.** Segunda Edição. Bookmark, 2006.

LUU, Hien. **Flex e Java Using BlazeDS.** Disponível em <http://www.slideshare.net/hluu/ria-with-flex-java-using-blazeds>. Acesso em 25 mai. 2011, 12:43.

MARCELO, Antonio. **Apache – Configurando o Servidor Web para Linux.** Brasport. 2005.

NET, Oficina da. **RIA – Rich Internet Application**. Disponível em <http://www.oficinadanet.com.br/artigo/1374/ria_-_rich_internet_application/2>. Acesso em 03 mai. 2011, 16:42.

NEWCOMER, Eric. **Understanding Web Services.** Addison Wesley, 2002.

ONG, Emil. **Fast and Efficient Client-Server Communication in Flash and Flex Using Hessian 2-50.**Disponível em <http://flashflex.com/fast-and-efficient-client-server-communication-in-flash-and-flex-using-hessian-2-50/>. Acesso em 18 mai. 2011, 02:21.

ORACLE. **Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools.** Disponível em <http://java.sun.com/developer/technicalArticles/WebServices/soa2/WSProtocols.html>. Acesso em 20 dez. 2010, 15:50

QUINGERSKI, Leandro. **Utilizando o modelo REST para construir Web Services.** 2008. 65 f. Monografia (Graduação em Sistemas de Informação) – União de Faculdade Dinâmica das Cataratas, Faculdade Dinâmica das Cataratas, Foz do Iguaçu, 2008.

REMOALDO, Pedro. **O Guia Prático do Dreamweaver CS3 com PHP, JavaScript e Ajax.** 1ª edição. Centro Atlântico, 2008.

RNP, Rede Nacional de Ensino e Pesquisa. **O que é middleware.** Disponível em <http://www.rnp.br/noticias/2006/not-060926.html>. Acesso em 25 mai. 2011, 20:35.

SCHMITZ, Daniel Pace. **Desenvolvendo Sistemas com Flex e PHP.** Novatec, 2009.

SILVEIRA, I.F. **Linguagem JAVA.** Disponível em <http://www.infowester.com/lingjava.php>. Acesso em 25 mai. 2011, 13:33.

TAPPER, Jeff; LABRIOLA, Michael; BOLES, Matthew; TALBOT, James. **Adobe Flex 3 - Treinamento direto da fonte.** Alta Books, 2009.

WARD, James; TIWARI, Shashank. **Building Web and Desktop Applications with BlazeDS and AMF.** Disponível em <http://www.infoq.com/articles/blazeds-intro>. Acesso em 25 mai. 2011, 13:23.

WEBBER, Jim; PARASTATIDIS, Savas; ROBINSON, Ian. **REST in Practice – Hypermedia and Systems Architecture.** O'Reilly Media, 2010.