REVISTA BRASILEIRA DE ADMINISTRAÇÃO CIENTÍFICA

Journal homepage:
www.arvore.org.br/seer

UM FRAMEWORK PARA INTEROPERABILIDADE ENTRE COMPONENTES DISTRIBUÍDOS HETEROGÊNEOS

RESUMO

A popularização do middleware ocorrida nos últimos anos promoveu o surgimento de diferentes modelos tecnológicos. Devido a essa diversidade, a interoperabilidade entre diferentes modelos de componentes de software torna-se imprescindível para promover a integração entre partes heterogêneas. Os problemas envolvidos com a interoperabilidade são tratados, em geral, pela adoção de sistemas de middleware capazes de intermediar e estabelecer a comunicação entre plataformas distintas. Nesse contexto, este trabalho propõe, implementa e avalia um framework para interoperabilidade entre diferentes modelos de componentes de software (i.e. OpenCOM e Fractal). A avaliação do framework proposto, a parte aplicada da pesquisa, ocorreu por meio de experimentos usando um cenário de heterogeneidade implementado baseado na composição de um Servidor Web Comanche. Os resultados alcançados ilustram a facilidade de uso e a simplificação para interoperabilidade entre componentes distribuídos heterogêneos providos pela solução proposta, bem como a avaliação do desempenho e a extensibilidade do framework proposto.

PALAVRAS-CHAVES: Modelos de Componentes; Middleware; Componentes Heterogêneos; Interoperabilidade; Sistemas Distribuídos.

A FRAMEWORK FOR INTEROPERABILITY BETWEEN HETEROGENEOUS DISTRIBUTED COMPONENTS

ABSTRACT

The middleware popularization occurred in recent years promoted the emergence of different technological models. Because of this diversity, interoperability between different models of software components becomes essential to promote the integration of heterogeneous parts. The problems involved with interoperability are treated in general by adopting middleware systems that are able to mediate and establish communication between different platforms. In this context, this work proposes, implements and evaluates a framework for interoperability between different models of software components (i.e. OpenCOM and Fractal). The evaluation of the proposed framework, the applied part of the research, occurred through experiments using a heterogeneity scenario based on the composition of a Comanche Web Server. The achieved results illustrate the ease of use and simplification for interoperability between heterogeneous distributed components provided by the proposed solution, as well as evaluating the performance and extensibility of the proposed framework.

KEYWORDS: Models of Components; Middleware; Heterogeneous Components; Interoperability; Distributed Systems.

Revista Brasileira de Administração Científica,
Aquidabã, v.4, n.2, Ago 2013.

*ISSN **2179-684X***

*SECTION: **Articles***
*TOPIC: **Sistemas e Tecnologia da Informação***

Anais do Simpósio Brasileiro de Tecnologia da Informação (SBTI 2013)

DOI: 10.6008/ESS2179-684X.2013.002.0017

Sidney Cassemiro do Nascimento
Instituto Federal de Educação, Ciência e Tecnologia de Sergipe, Brasil
http://lattes.cnpq.br/4670165199734592
sidneycn@gmail.com

Felipe Oliveira Carvalho
Universidade Federal de Sergipe, Brasil
http://lattes.cnpq.br/9791260608741448
ocfelipe@gmail.com

Tarcisio da Rocha
Universidade Federal de Sergipe, Brasil
http://lattes.cnpq.br/8661248119285296
tarcisiorocha@gmail.com

Received: 07/07/2013
Approved: 05/08/2013
Reviewed anonymously in the process of blind peer.

Referencing this:

*NASCIMENTO, S. C.; CARVALHO, F. O.; ROCHA, T.. Um framework para interoperabilidade entre componentes distribuídos heterogêneos. **Revista Brasileira de Administração Científica**, Aquidabã, v.4, n.2, p.239-256, 2013. DOI: http://dx.doi.org/10.6008/ESS2179-684X.2013.002.0017*

INTRODUÇÃO

A demanda de soluções em sistemas distribuídos complexos vem substituindo a visão de sistemas distribuídos homogêneos onde aplicações de domínio específico são desenvolvidas usando plataformas e *middleware* projetados especificamente para esse domínio.

Soluções tecnológicas independentes têm sido interconectadas para criar estruturas ainda mais ricas, os chamados sistemas de sistemas (SoS). Um dos principais desafios dessas interconexões é a questão da interoperabilidade: a habilidade desses sistemas se conectarem, de trocar dados e de se comunicar (BLAIR et al., 2011; INVERARDI et al., 2010; BROMBERG et al., 2011).

Uma técnica que tem sido amplamente utilizada nos últimos anos no desenvolvimento de plataformas de sistemas distribuídos é a ESBC (Engenharia de *Software* Baseada em Componentes) (ROUVOY; MERLE, 2009). Como consequência do sucesso no uso de componentes de *software*, vários modelos de componentes diferentes emergiram. OpenCOM, Fractal, Spring, EJB, CCM e SCA são alguns exemplos destes modelos. A heterogeneidade de modelos de componentes para os mais diversos domínios de aplicações é um desafio para o desenvolvimento de sistemas distribuídos contemporâneos. Um desses desafios é quando as partes que devem compor esses sistemas são desenvolvidas em plataformas que usam modelos de componentes diferentes. Essa falta de padronização limita a reutilização e a composição de componentes.

Um modelo de componentes de *software* define um conjunto de normas para a implementação do componente, atribuição de nomes, interoperabilidade, customização, composição, evolução e implantação (CRNKOVIC et al., 2011). Um *framework* é uma abstração que fornece um grau muito elevado de reutilização dos componentes de *software*. Desta forma, um *framework* para interoperabilidade entre diferentes modelos de componentes de *software* pode ser usado para promover flexibilidade e padronização no desenvolvimento de aplicações distribuídas, permitindo reutilização e composição de componentes desenvolvidos em modelos de programação distribuída distintos.

A seção seguinte contextualiza a fundamentação teórica adotada nesse estudo e posteriormente apresentam-se os procedimentos metodológicos. Na sequência estão os resultados da pesquisa e, finalmente, as conclusões e trabalhos futuros.

REVISÃO TEÓRICA

A fim de fundamentar o projeto e a implementação do *framework* proposto, será apresentado nesta seção o embasamento teórico que foi utilizado no decorrer dessa pesquisa.

Invocação Remota de Métodos

Segundo Coulouris et al. (2011), uma das formas de se estabelecer comunicação entre objetos remotos distribuídos é através da invocação a método remoto. Os objetos que podem receber invocações a métodos remotos são chamados de objetos remotos e implementam uma interface remota. A diferença entre objetos remotos e objetos locais é que os objetos remotos estão localizados em máquinas virtuais diferentes (HAROLD, 2004). Devido à possibilidade de falhas isoladas tanto nos objetos invocadores quanto nos invocados, as invocações remotas têm semânticas diferentes das invocações a métodos locais. Embora elas possam ser feitas de modo bastante semelhante às invocações locais, transparência total não é necessariamente desejável.

De acordo com Reilly e Reilly (2002), cada serviço RMI (*Remote Method Invocation*) é definido por uma interface que descreve os métodos dos objetos que podem ser executados remotamente. Esta interface deve ser compartilhada por todos os desenvolvedores que escrevem *software* para esse serviço - que funciona como um modelo para aplicações que irão utilizar e fornecer implementações do serviço. Várias implementações da interface podem ser criadas, e os desenvolvedores não precisam estar cientes de que a implementação está sendo utilizada e nem onde a mesma está localizada. A arquitetura da RMI consiste de quatro camadas (RUIXIAN, 2000), conforme mostrada na Figura 1.

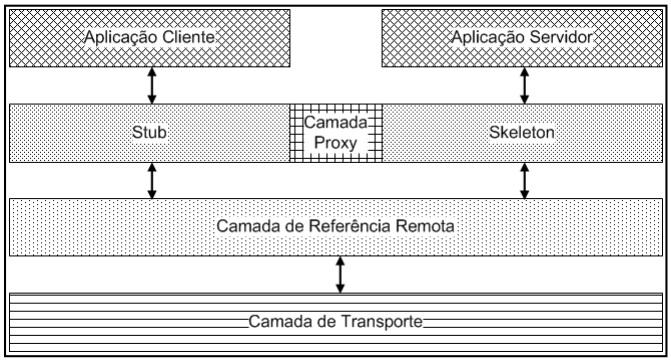


Figura 1: Arquitetura em Camadas da RMI.
Fonte: adaptado de Ruixian (2000).

A primeira camada é a de **aplicação**, onde se encontram as implementações das aplicações clientes e servidoras. O *stub* e o *skeleton* representam o lado cliente e servidor, respectivamente, e estão localizados na segunda camada, que é a ***proxy*** - responsável por todas as chamadas para o objeto remoto, empacotamento dos parâmetros (*marshalling*) e retorno do objeto. A terceira camada é a de **referência remota** que corresponde a uma abstração entre a camada ***proxy*** e a quarta e última camada, a de **transporte**.

A aplicação cliente não interage diretamente com o servidor. Na realidade, a aplicação cliente interage apenas com o objeto *stub* - que corresponde ao *proxy* do lado cliente para o objeto remoto e a interface de aplicação para o objeto remoto. O *stub* passa a chamada para o

objeto remoto através da camada de referência remota no cliente, que repassa para a camada de transporte. Em seguida, a camada de transporte no cliente passa os dados através da rede para a camada de transporte no lado servidor, que se comunica com a camada de referência remota no servidor e repassa a chamada para o *skeleton* - o correspondente ao *proxy* do lado servidor para os objetos remotos, que tem comportamento semelhante ao de uma classe *stub*. O *skeleton* se comunica com o próprio servidor que faz a chamada do método para o objeto remoto. No outro sentido (servidor-cliente), o fluxo é simplesmente inverso.

Na plataforma Java, o mecanismo padrão para dar suporte a invocação de métodos remotos é o RMI. Em particular, RMI permite que os objetos invoquem métodos em objetos remotos usando a mesma sintaxe das invocações locais (COULOURIS et al., 2011). Em RMI Java, supõe-se que os parâmetros de um método são parâmetros de entrada e o resultado de um método é um único parâmetro de saída. RMI permite a passagem de qualquer objeto Java serializável em invocações de métodos definidos na interface do componente. Todos os tipos primitivos e objetos remotos são serializáveis. A verificação de tipo se aplica igualmente às invocações remotas e locais. O código para empacotar e desempacotar argumentos e para enviar mensagens de requisição e resposta pode ser gerado automaticamente por um compilador de interface, a partir da definição da interface remota.

Serviços *Web*

As interfaces de serviços disponibilizadas na WWW (*World Wide Web*) para comunicação entre aplicações são denominadas de serviços *web* (*web services*) (W3C, 2012). Segundo Deitel e Deitel (2010), um serviço *web* é um componente de *software* armazenado em um computador que pode ser acessado por um aplicativo (ou outro componente de *software*) em outro computador por uma rede. Um serviço *web* fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores *web* (COULOURIS et al., 2011). Um cliente normalmente acessa as operações de um serviço *web* por meio de solicitações e respostas formatadas em XML (*Extensible Markup Language*) (W3C, 2012) - um formato textual usado para representação e empacotamento de dados - e, normalmente, transmitidas por HTTP (*HyperText Transfer Protocol*) (W3C, 2012) - um protocolo para transportar mensagens entre clientes e servidores pela Internet.

Atualmente, existem duas arquiteturas de serviços *web* (PETERSON; DAVIE, 2007): *(i)* **SOAP** (*Simple Object Access Protocol*) (W3C, 2012) - baseada em dois *frameworks* que foram padronizados pelo W3C (*World Wide Web Consortium*) denominados de WSDL (*Web Services Definition Language*) (W3C, 2012) - um *framework* para especificação de protocolos de aplicação (interfaces) e SOAP - um *framework* para especificação de protocolos de transporte (PETERSON; DAVIE, 2007). O SOAP é um protocolo independente de plataforma que utiliza a XML para fazer chamadas de procedimento remoto, geralmente via HTTP (DEITEL; DEITEL, 2010). Serviços *web*

podem ser publicados através de UDDI (*Universal Description, Discovery and Integration*) (OASIS, 2012) - um *framework* de plataforma independente para descrever e integrar os serviços de negócios usando a Internet. A UDDI é baseada na arquitetura SOAP e *(ii)* **REST** (*REpresentational State Transfer*) - baseada na arquitetura *Web* com algumas extensões para suportar serviços *web* (PETERSON; DAVIE, 2007). REST fornece uma arquitetura de rede que utiliza os mecanismos de solicitação/resposta tradicionais da *Web* como solicitações GET e POST (DEITEL; DEITEL, 2010). Os serviços *web* são considerados como recursos identificados por URIs (*Uniform Resource Identifier*) e acessados via HTTP (PETERSON; DAVIE, 2007). URI é uma cadeia de caracteres que é usada para identificar um recurso, onde um recurso pode ser qualquer coisa que tenha uma identificação, tal como um documento, uma imagem ou um serviço (PETERSON, DAVIE, 2007).

Componentes Distribuídos

Definido de forma geral, componente é um bloco construtivo modular para *software* (PRESSMAN, 2010). Mais formalmente, (SZYPERSKI, 1997) um componente é uma unidade de composição com interfaces especificadas contratualmente e com dependências de contexto explícitas. Um componente de *software* pode ser implementado de forma independente ou através de composição de componentes de terceiros.

Um modelo de componentes define um conjunto de normas para a implementação do componente, atribuição de nomes, interoperabilidade, customização, composição, evolução e implantação (CRNKOVIC et al., 2011).

Middleware Baseados em Componentes

As soluções baseadas em componentes foram desenvolvidas para superar uma série de limitações que foram observadas no desenvolvimento de aplicações que trabalham com *middleware* de objetos distribuídos, como por exemplo, dependências implícitas de objetos, complexidade de programação, falta de separação de interesses de distribuição e falta suporte para a implantação de configurações de objetos (COULOURIS et al., 2011).

Segundo Coulouris et al. (2011), a principal característica dos objetos distribuídos é que eles permitem a adoção de um modelo de programação orientado a objetos para o desenvolvimento de sistemas distribuídos e, com isso, ocultam a complexidade subjacente da programação distribuída. Nesta abordagem as entidades que se comunicam são representadas por objetos distribuídos. Os objetos se comunicam através de invocação remota de métodos, eventos distribuídos ou outro paradigma de comunicação, como por exemplo, serviços *web*.

Diversas plataformas de *middleware* que permitem a implementação de sistemas baseados em componentes estão disponíveis, como por exemplo, o EJB (*Enterprise JavaBeans*),

o CORBA *Component Model*, o COM (*Component Object Model*), o .NET *Framework*, o OpenCOM e o Fractal *Component Model*. Dentre as plataformas a serem suportadas pelo *framework* proposto estão as tecnologias OpenCOM e Fractal, apresentadas nas próximas subseções.

O Modelo OpenCOM

O OpenCOM é um modelo de componentes de baixo peso, projetado para o desenvolvimento de *middlewares* em dispositivos de computação com poucos recursos (processamento, memória, armazenamento) (COULSON et al., 2008). Além de ser um modelo de baixo peso, o OpenCOM provê a capacidade de reconfiguração dinâmica de *middlewares* tanto no domínio estrutural quanto no comportamental (ROCHA, 2008).

O OpenCOM é fundamentado em três tecnologias: *(i)* **Componentes** - O modelo permite a especificação da estrutura de sistemas através do uso de componentes e conexões entre componentes (ROCHA, 2008). Cada componente pode possuir um conjunto de interfaces (ou interfaces providas) - através das quais o componente exporta seus serviços - e um conjunto de receptáculos (ou interfaces requeridas) - através dos quais o componente requisita serviços de outros componentes, *(ii)* **Reflexão computacional** - O modelo foi projetado para dar apoio à reflexão computacional - que é a capacidade que um sistema tem de observar sua própria representação/estrutura e até mesmo modificá-la e *(iii)* ***Frameworks*** **de componentes** - O modelo também permite a criação de *frameworks* de componentes. Um *framework* de componentes é definido no OpenCOM como uma subarquitetura de componentes sobre a qual pode ser checado um conjunto de propriedades arquiteturais desejadas (ROCHA, 2008). Essa subarquitetura também pode ser vista como um componente complexo que é composto de outros subcomponentes interconectados.

Uma visão de alto nível de um exemplo dos elementos do modelo de programação baseado em componentes OpenCOM é ilustrado na Figura 2. Ela mostra um componente (*Client*) que possui um receptáculo conectado à interface provida de outro componente (*Server*).

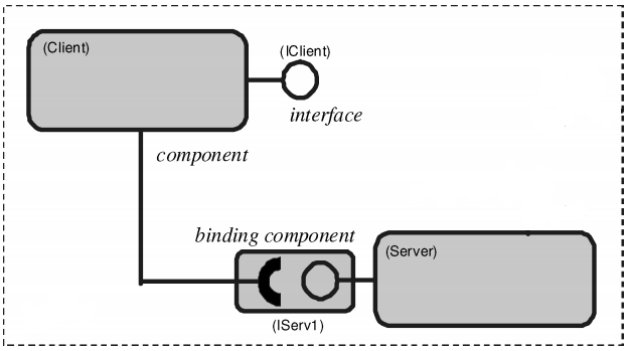


Figura 2: Elementos do modelo de programação no nível do *kernel*.
Fonte: Coulson et al. (2008).

O ambiente de execução do OpenCOM permite a execução de sistemas baseados em componentes localizados na mesma máquina (ROCHA, 2008). Esse ambiente é composto pelo OpenCOM *Runtime* e o *System Graph*. OpenCOM implanta um *kernel* de tempo de execução (OpenCOM *Runtime*) padrão que gerencia a criação e a exclusão de componentes, e age de acordo com solicitações para conexão e desconexão dos componentes (GRACE, 2007), processando e armazenando informações sobre sistemas implementados com componentes OpenCOM (ROCHA, 2008). Além disso, um grafo (*System Graph*) dos componentes atualmente em uso é mantido para apoiar a introspecção de uma estrutura da plataforma (GRACE, 2007), onde os componentes são mantidos e executados (ROCHA, 2008).

As funcionalidades do ambiente de execução do OpenCOM são acessadas através da interface IOpenCOM. As principais operações em tempo de execução são: criar uma nova instância de um componente e inseri-la em tempo de execução; excluir uma instância de um componente que foi criada anteriormente; conectar um receptáculo de um componente em uma interface requerida de um componente de origem, desconectar um receptáculo de uma interface; devolver os metadados armazenados no metamodelo de arquitetura sobre um componente individual; e retornar o nome do componente registrado exclusivamente para uma referência de um determinado componente.

O Modelo Fractal

O Fractal é apresentado em (OW2, 2004) como um modelo de componentes modular e extensível, que pode ser utilizado para projetar, implementar, implantar e reconfigurar sistemas e aplicações (OW2, 2004). O objetivo geral do Fractal é reduzir os custos de implantação, desenvolvimento e manutenção de *softwares*, e em particular os projetos do consórcio ObjectWeb (OW2, 2004). Segundo Bruneton et al. (2006), o Fractal é um modelo de componentes de baixo peso, hierárquico e reflexivo, com compartilhamento de componentes, proporcionando os benefícios da programação baseada em componentes para o desenvolvimento de sistemas distribuídos. Este modelo tem sido utilizado na construção de uma ampla gama de plataformas de *middleware* (COULOURIS et al., 2011). O modelo de componentes Fractal suporta várias linguagens de programação, como por exemplo, Java e C, e de forma experimental .NET, SmallTalk e Python.

Um componente Fractal é uma entidade de execução que está encapsulada, com identificação distinta e que suporta uma ou mais interfaces (BRUNETON et al., 2004). A interface é um ponto de acesso a um componente, que implementa um tipo de interface que representa as operações suportadas (BRUNETON et al., 2006). As interfaces podem ser de dois tipos: *(i)* **interfaces servidoras** - que correspondem a pontos de acesso que aceitam solicitações de serviços de outro componente, e *(ii)* **interfaces cliente** - que correspondem a pontos de acesso que invocam os serviços de outro componente.

Internamente, um componente Fractal é composto por duas partes (OW2, 2004): *(i)* uma **camada de controle** ou **membrana** - que possui as interfaces externas que permitem a introspecção para monitoramento e controle dos componentes em execução, bem como a capacidade de reconfiguração estática e dinâmica dos componentes, e *(ii)* a **camada de conteúdo** - que é composta por outros componentes também conhecidos como subcomponentes, que estão sob o controle do componente de controle envolvente. As interfaces da membrana podem ser externas, sendo acessíveis por outros componentes ou internas, que são acessíveis somente pelos subcomponentes. A Figura 3 mostra a estrutura básica de um componente Fractal.

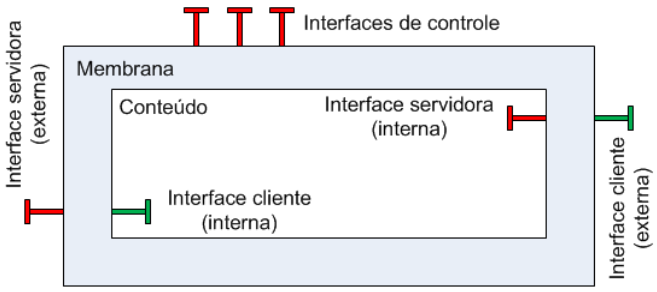


Figura 3: Estrutura de um componente Fractal.
Fonte: adaptado de Bruneton et al. (2006).

METODOLOGIA

A metodologia empregada no desenvolvimento deste trabalho foi a pesquisa exploratória com vistas à implementação e teste do *framework* proposto, mediante uso dos procedimentos técnicos: pesquisa bibliográfica e experimentos como método empírico para validação da proposta.

Solução Proposta: InteropFrame

O papel do InteropFrame é prover uma solução de interoperabilidade transparente entre os sistemas baseados em componentes distribuídos de modelos distintos, possibilitando que os componentes envolvidos na construção de aplicações distribuídas interajam através dos mecanismos automáticos de interoperabilidade providos. A abordagem utilizada para isso é a geração automática de *proxies* que intermediam a comunicação entre as partes heterogêneas distribuídas.

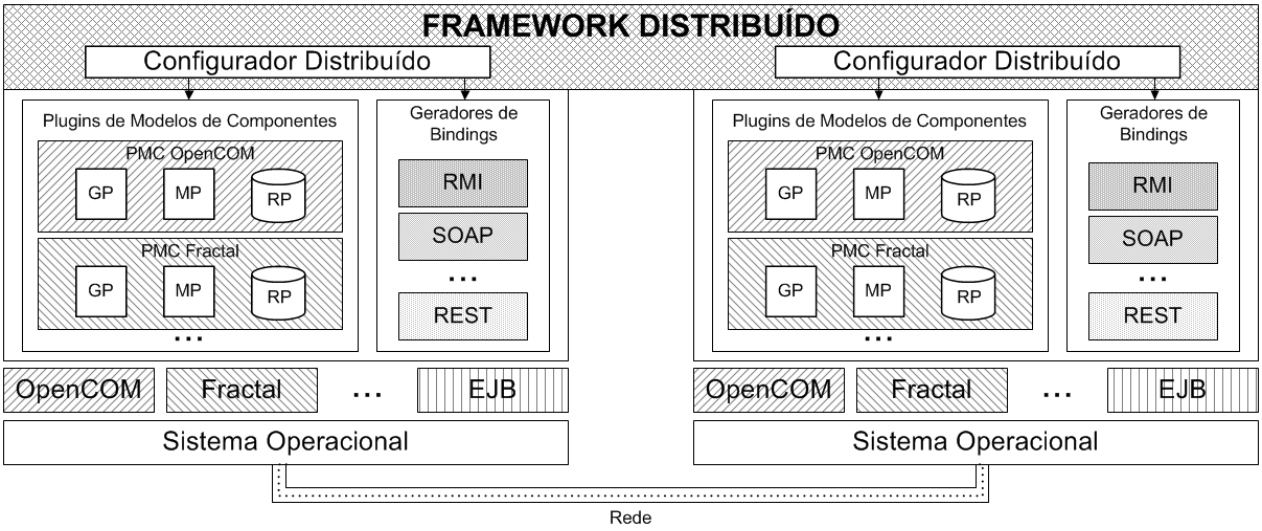


Figura 4: Arquitetura do *framework* distribuído InteropFrame.

A Figura 4 apresenta a arquitetura detalhada do *framework* distribuído InteropFrame. Os seus principais módulos são: *(i)* **Configurador Distribuído (CD)** - responsável pelo gerenciamento do serviço de interoperabilidade entre componentes distribuídos. Este módulo coordena as operações dos demais módulos do *framework* distribuído, *(ii)* ***Plugins* de Modelos de Componentes (PMC)** - cada *plugin* dá ao *framework* o suporte a um modelo de componentes diferente e *(iii)* ***Plugins* de Geradores de *Bindings* (GB)** - cada *plugin* é responsável pela geração automática do código-fonte de um tipo diferente de *binding* entre componentes remotos. O PMC é composto pelos seguintes submódulos: o **Gerador de *Proxies* (GP)** - responsável pela geração automática dos *proxies* que possibilitam a interoperabilidade entre os componentes distribuídos heterogêneos; o **Montador de *Proxies* (MP)** - responsável pelo carregamento dos *proxies* criados pelo GP sob demanda e disponibilização do serviço de interoperabilidade entre os componentes distribuídos; e o **Repositório de *Proxies* (RP)** - para gerenciamento do catálogo de *proxies* gerados pelo GP. O Gerador de *Proxies* pode usar um tipo específico de Gerador de *Binding* para enxertar no *proxy* o código que promove a interconexão remota.

Atualmente, o InteropFrame suporta a interoperabilidade entre componentes distribuídos dos modelos OpenCOM e Fractal, e suporta o *binding* entre componentes remotos usando os mecanismos RMI Java ou *Web Service* SOAP. Visando a extensibilidade deste *framework*, os módulos de *Plugins* de Modelos de Componentes (PMC) e *Plugins* de Geradores de *Bindings* (GB) foram projetados para suportar diversos tipos de modelos e diversos tipos de *bindings* na forma de *plugins* totalmente independentes. Dessa forma, outros modelos de componentes (ex: OSGi, EJB) e outros tipos de *bindings* remotos (*Web Services* RESTful, XML-RPC) podem vir a ser apoiados pelo *framework* na forma de *plugins* desenvolvidos à parte.

Na prática, isso significa que, se for desenvolvido mais um *plugin* para o modelo OSGi, por exemplo, o InteropFrame passaria a suportar a interoperabilidade não somente entre sistemas distribuídos heterogêneos OpenCOM/Fractal, mas também entre sistemas Open-COM/OSGi e

Fractal/OSGi, sem qualquer alteração nos *plugins* já existentes do OpenCOM e do Fractal. Além disso, a forma de *binding* remoto utilizada para interligar os sistemas heterogêneos distribuídos pode ser qualquer uma dentre as disponibilizadas pelos *plugins* de *bindings*, independentemente dos modelos de componentes desses sistemas. Esse total desacoplamento entre modelos de componentes suportados e tipos de *bindings* suportados é uma das contribuições mais relevantes dessa arquitetura como uma solução para os desafios da interoperabilidade.

Cenário Proposto para Avaliação

Para avaliação do *framework* proposto, foi implementado um cenário de heterogeneidade baseado na composição de um Servidor *Web Comanche* (ROUVOY; MERLE, 2009) com um conjunto de características desejadas.

O Comanche é um servidor *web* simples que oferece os recursos básicos de um servidor *web*, como por exemplo, devolução de páginas estáticas HTML. A seguir, são descritos os sete componentes que compõem *Comanche Web Server*: *(i)* **Receiver** - Componente OpenCOM responsável pelo recebimento das requisições HTTP de entrada, *(ii)* **Scheduler** - Componente OpenCOM responsável pelo escalonamento das requisições HTTP para análise, *(iii)* **Analyzer** - Componente Fractal que efetua a análise das requisições HTTP, *(iv)* **Logger** - Componente Fractal que registra as requisições HTTP de entrada, *(v)* **Dispatcher** - Componente Fractal responsável pela interpretação das requisições HTTP de entrada; *(vi)* **FileHandler** - Componente Fractal que realiza a manipulação da solicitação de um arquivo e *(vii)* **ErrorHandler** - Componente Fractal responsável pelo tratamento de erros (se existirem).

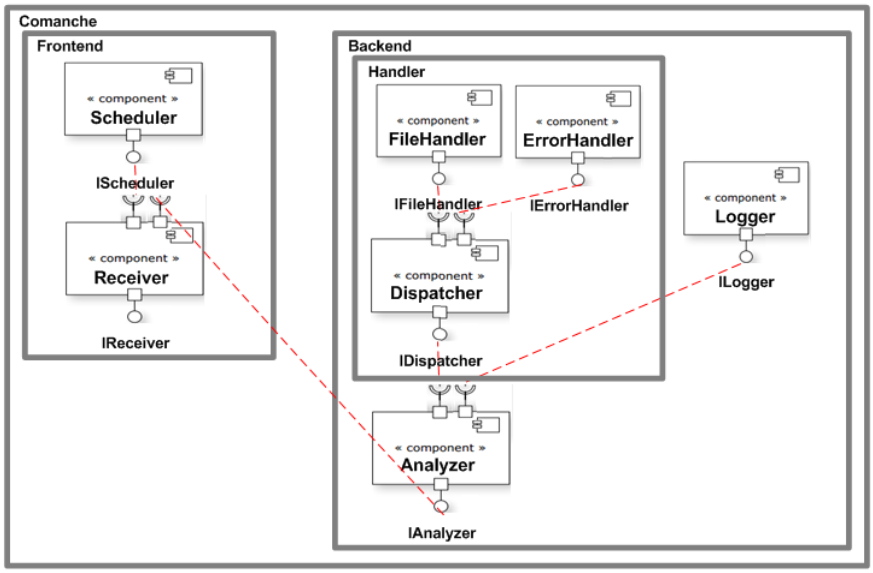


Figura 5: Arquitetura do Servidor *Web Comanche*.
Fonte: adaptado de Rouvoy e Merle (2009).

A Figura 5 apresenta a arquitetura do Servidor *Web Comanche*. O processo ocorre da seguinte forma: No *Frontend*, as requisições HTTP de entrada são recebidas pelo *Receiver* e sua análise é agendada pelo *Scheduler*. Em seguida, no *Backend*, a análise é tratada pelo *Analyzer* e o registro das requisições é feito pelo *Logger*. A seguir, o *Dispatcher* delega a manipulação das requisições de arquivos para o *FileHandler* e, finalmente, se ocorrer alguma falha, o *ErrorHandler* gera uma página *web* de erro.

O papel do *Frontend* é o recebimento e escalonamento das requisições HTTP de entrada enquanto que o *Backend* é responsável pela análise e interpretação das requisições HTTP. Dessa forma, a distribuição de parte dos serviços do Comanche para outros *hosts* como balanceamento de carga pode ser empregada com o objetivo de promover uma distribuição mais homogênea da carga de trabalho e evitar a sobrecarga do mesmo, melhorando o desempenho do Servidor *Web Comanche*. Ver a Figura 6.

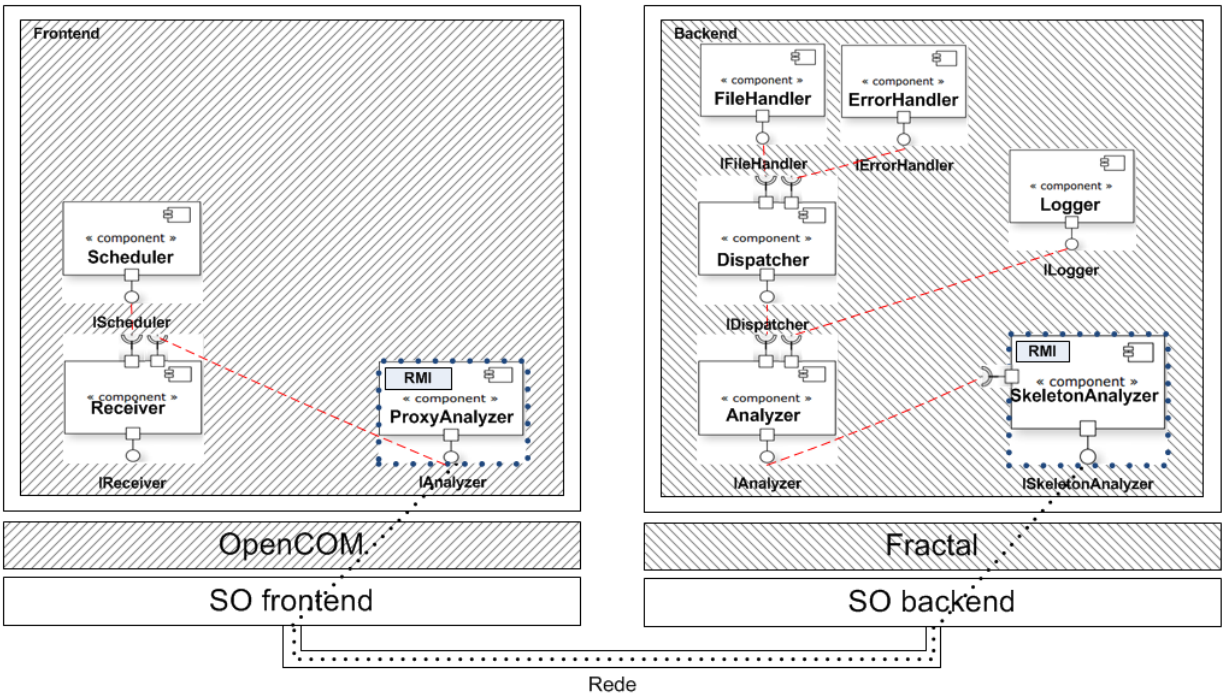


Figura 6: Arquitetura do Servidor *Web Comanche* com componentes distribuídos.

A Figura 6 mostra os componentes *ProxyAnalyzer* e *SkeletonAnalyzer* gerados automaticamente pelo InteropFrame para promover a interconexão entre as partes distribuídas do Servidor *Web Comanche*. Considerando que o *frontend* foi desenvolvido em OpenCOM e o *backend* em Fractal, a tarefa do *framework* foi interconectar esses dois subsistemas heterogêneos formando um sistema único funcional.

Para possibilitar a interconexão entre o componente OpenCOM *Receiver* localizado no *frontend* e o componente Fractal *Analyzer* localizado no *backend*, bastou informar ao InteropFrame os identificadores desses dois componentes, a interface usada na interconexão e a forma de *binding*. Depois disso, o InteropFrame gerou automaticamente os *proxies* que

promoverão a ligação transparente entre os componentes remotos, sendo eles: **ProxyAnalyzer** - Componente OpenCOM que representa localmente o componente remoto *Analyzer* e **SkeletonAnalyzer** - Componente Fractal que define um receptáculo para requisitar serviços do componente *Analyzer*. O componente *ProxyAnalyzer* implementa a mesma interface *IAnalyzer* do componente *Analyzer* remoto. Isso possibilita que o componente *Receiver* se conecte ao *ProxyAnalyzer* através dessa interface. O *ProxyAnalyzer* utiliza RMI para comunicação remota. Quando uma operação do *ProxyAnalyzer* é chamada através de sua interface *IAnalyzer*, o *ProxyAnalyzer* repassa essa chamada para o objeto remoto *SkeletonAnalyzer* através da invocação de uma operação equivalente na interface remota *ISkeletonAnalyzer*. O *SkeletonAnalyzer* também representa o objeto remoto Java esqueleto do lado servidor, o qual implementa os métodos da interface remota RMI *ISkeletonAnalyzer*. Ao receber uma invocação remota, ele intercepta o argumento na mensagem de requisição usando-o para invocar o método correspondente no componente *Analyzer* através do seu receptáculo. Em seguida, o *SkeletonAnalyzer* recebe o resultado do *Analyzer* e envia como resultado para o método do *ProxyAnalyzer* que fez a requisição.

Neste cenário, foi utilizado RMI como mecanismo intermediário de comunicação remota entre os componentes distribuídos heterogêneos. Na prática, o papel da RMI foi o de transmitir os parâmetros das operações chamadas no *ProxyAnalyzer* para o *SkeletonAnalyzer* (objeto remoto RMI). Por sua vez, o *SkeletonAnalyzer* ficou responsável por chamar a operação correspondente no *Analyzer* passando os parâmetros recebidos. A resposta recebida a essa chamada foi repassada no sentido inverso ao *ProxyAnalyzer* e em seguida ao *Receiver*. No exemplo apresentado anteriormente, os componentes para o Servidor *Web Comanche* estão localizados em uma rede local. Por este motivo, utilizou-se o gerador de *binding* que faz o uso do mecanismo do RMI Java para comunicação remota. Considerando o mesmo exemplo para redes distintas foi utilizada uma abordagem semelhante com a adoção do gerador de *binding* que usa *Web Services* via SOAP.

Ambiente de Desenvolvimento

Para realizar o desenvolvimento e a avaliação de desempenho do *framework*, foram utilizados dois computadores: (*i*) *Intel(R) Core(TM) i5 CPU M450 2.40GHz* com 4GB de RAM, com o Sistema Operacional de 64 Bits *Microsoft Windows 7 Professional Service Pack 1* instalado e (*ii*) *Intel(R) Core(TM) i5 CPU M2450 2.50GHz* com 6GB de RAM, com o Sistema Operacional de 64 Bits *Microsoft Windows 8 Professional*. Para conectar as máquinas cliente e servidora foi utilizado um *cross cable*, de modo que o tráfego de rede entre as máquinas foi isolado de qualquer outro tráfego para evitar interferência nos resultados. Além disso, à medida do possível, foram desabilitados outros processos/aplicações nas máquinas concorrendo por recursos de rede ou executando em paralelo aos testes, com exceção daqueles necessários ao correto funcionamento

do Sistema Operacional. Para a realização dos testes, todo o código-fonte foi compilado e executado a partir do Eclipse Juno, ambiente de desenvolvimento para aplicações Java, utilizando a máquina virtual Java(TM) SE *Runtime Environment* (*build* 1.7.0_11-b21).

RESULTADOS

Nesta seção, serão apresentados os resultados da avaliação do *framework* distribuído InteropFrame no cenário proposto, nos aspectos de usabilidade, desempenho e extensibilidade.

Avaliação de Usabilidade

O *framework* InteropFrame é leve e sua instalação é rápida e simples não requerendo muitas etapas, sendo necessário apenas baixar e instalar o *framework*.

O InteropFrame está disponível através do arquivo *InteropFrame.zip*, que contém um arquivo *IFrame.jar*, além das bibliotecas necessárias na pasta *lib* para o seu funcionamento. O *download* do InteropFrame com o seu manual de instalação e uso pode ser feito em http://computacao.ufs.br/pagina/interopframe-9682.html.

A instalação do *InteropFrame.zip* é simples. Para utilizar o *framework* é necessário extrair o conteúdo de *InteropFrame.zip* na pasta raiz do sistema desenvolvido em algum modelo de componente suportado (OpenCOM ou Fractal). Para configurar o InteropFrame para o uso, os arquivos da pasta *lib* bem como o arquivo *IFrame.jar* devem ser adicionados ao *Build Path* do projeto.

O esforço gasto em termos do número de instruções fontes (linhas de código correto), sem erros, para integrar o *proxy* e o *skeleton* gerados aos seus respectivos pares no cenário proposto (Servidor *Web Comanche* com componentes distribuídos) também é aceitável, uma vez que menos de dez linhas (somando o lado cliente ao lado servidor) são gastas para a execução da solução implementada com o uso do InteropFrame.

Foi possível constatar que o processo de geração automática dos *proxies* para interconectar os componentes OpenCOM *Receiver* e Fractal *Analyzer* no cenário proposto para avaliação foi simplificado através dos mecanismos automáticos de interoperabilidade providos pelo *framework* proposto. Este nível de automatização permite aos desenvolvedores diminuir tempo e esforço de codificação para promover a interconexão entre os componentes distribuídos heterogêneos.

Avaliação de Desempenho

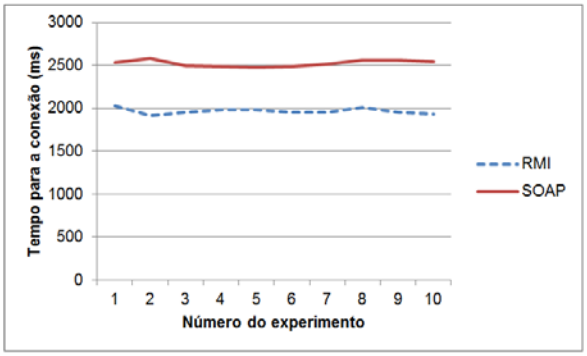
Para realização da avaliação de desempenho foi medido, de forma sistemática, o tempo que o *framework* levou para: *(i)* conectar um componente a outro componente remoto e *(ii)*

estabelecer a comunicação entre os componentes, variando os modelos entre OpenCOM e Fractal, bem como a variação do *binding* entre RMI Java e *Web Service* via SOAP entre tais modelos de componentes.

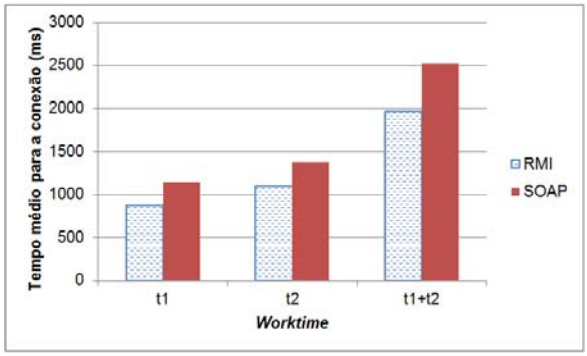
Para testar a influência dos *bindings* no desempenho, foram implementados dois experimentos para cada cenário avaliado, variando o mecanismo intermediário de comunicação remota entre os componentes distribuídos heterogêneos, ora o RMI Java e ora o *Web Service* via SOAP.

Nos experimentos, para cada variação de parâmetros na avaliação de desempenho, cada teste foi executado 11 vezes a fim de possibilitar uma análise dos tempos de execução. Foi observado que a primeira execução apresentou um *outlier*, um valor atípico, podendo seu tempo ser descartado, pelo fato do sistema não estar estável. Logo, este primeiro valor foi excluído de todas as amostras. O tempo foi medido utilizando dois métodos estáticos da classe *java.lang.System*: *currentTimeMillis* e *nanoTime*.

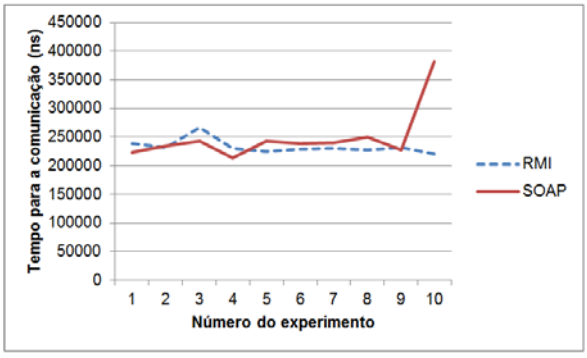
Para testar o desempenho, foram feitos três testes: *(i)* o primeiro teste de desempenho utilizou um cenário com os componentes homogêneos no *frontend* e no *backend* no modelo Fractal, *(ii)* o segundo teste de desempenho também foi utilizado um cenário homogêneo com todos os componentes implementados no modelo OpenCOM e *(iii)* para o terceiro teste de desempenho foi utilizado um cenário heterogêneo com os componentes OpenCOM no *frontend* e componentes Fractal no *backend*. Os resultados dos experimentos do terceiro teste são apresentados nos gráficos da Figura 7, que mostram os mesmos padrões observados nos experimentos para cada cenário avaliado.



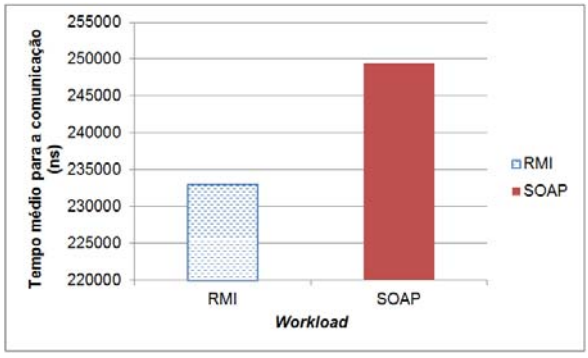
(a) Resultados do *worktime*



(b) Médias do *worktime*



(c) Resultados do *workload*

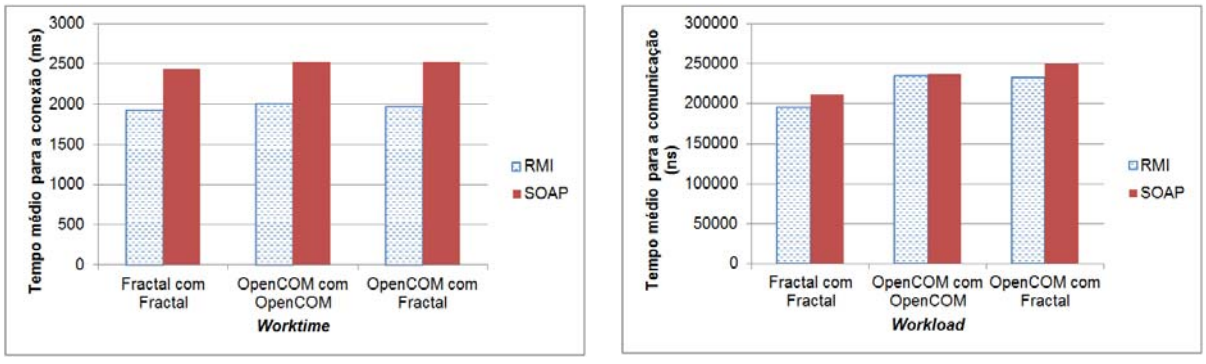


(d) Médias do *workload*

Figura 7: Resultados dos testes relacionado a OpenCOM com Fractal.

No gráfico 7(a) são apresentados os resultados obtidos nos experimentos executados na avaliação de desempenho da conexão entre dois componentes distribuídos. Neste gráfico, o eixo X indica o número do experimento executado e o eixo Y indica o tempo necessário para a conexão entre os componentes remotos. Esse experimento inclui o tempo total para geração, compilação e carregamento do *proxy*. Foram calculados os valores t1 e t2 que representam as médias dos tempos computacionais totais necessários em milisegundo (ms) para cada *worktime* no lado servidor e no lado cliente, respectivamente. Esses resultados são mostrados no gráfico 7(b). O gráfico 7(c) mostra os resultados obtidos nos experimentos executados na avaliação de desempenho da comunicação entre os componentes remotos. Neste gráfico, o eixo X representa o número do teste executado e no eixo Y representa o tempo necessário para a comunicação. Os tempos médios para a comunicação dos *workloads* relacionados aos testes de desempenho para cada *binding* são comparados no gráfico 7(d).

A Figura 8 apresenta a um comparativo das médias dos testes de desempenho. O gráfico 8(a) mostra um comparativo entre os tempos médios para a conexão entre os componentes remotos relacionados aos três experimentos. Observa-se um melhor desempenho geral dos experimentos usando o *binding* com RMI do que com o SOAP. Isso ocorre porque esse processo envolve a instanciação e a ativação dos *proxies* que, por sua vez, é mais custoso com o uso de *Web Services* SOAP do que com RMI Java. Outro aspecto que pode ser observado é que os valores das médias não se diferenciaram muito de um *worktime* para outro. O gráfico 8(b) mostra um comparativo entre os tempos médios para a comunicação entre os componentes remotos relacionados aos três experimentos. Observa-se um melhor desempenho dos testes usando o experimento com todos os componentes implementados no modelo Fractal. O desempenho da comunicação entre os componentes remotos usando o *binding* com RMI foi sempre superior com relação ao SOAP.



(a) Médias do *worktime* (b) Médias do *workload*

Figura 8: Comparativo das médias dos experimentos.

Avaliação de Extensibilidade

O InteropFrame permite a extensão para novos modelos de componentes, através do desenvolvimento de *plugins* à parte. O desenvolvedor que necessite estender o InteropFrame necessita conhecer em detalhes o funcionamento do modelo de componentes preterido, além de seguir alguns padrões estabelecidos na criação do *framework* InteropFrame, como a ferramenta de geração de código a partir de *templates* (*Apache Velocity*) e as outras tecnologias utilizadas.

O suporte à extensibilidade é feito de forma nativa na arquitetura do *framework*, permitindo que novos modelos de componentes e novos tipos de *bindings* possam ser suportados pelo mesmo através do desenvolvimento à parte de novos *plugins*. É possível observar que a extensão ou modificação de alguma funcionalidade do InteropFrame, é uma tarefa que requer conhecimento da arquitetura de implementação e tecnologias utilizadas.

CONCLUSÕES

Apesar da evolução das tecnologias de *middleware* com a adoção de modelos de componentes, constata-se que a área de sistemas distribuídos apresenta várias barreiras para interoperabilidade entre tais tecnologias, e a heterogeneidade de plataformas de *middleware* é um desafio para programação distribuída baseada em componentes de diferentes modelos de componentes.

Desenvolver sistemas de *software* para ambientes distribuídos que combinem diferentes modelos de componentes é uma tarefa difícil. O problema de interoperabilidade de plataformas heterogêneas é muito complexo, pois envolve o tratamento de questões relacionadas às diferenças de modelos de distribuição (ex: uma plataforma pode ser Orientada a Objeto e outra Orientada a Mensagem), protocolos de sessão (ex: um protocolo *request-reply* e outro *General Inter-ORB Protocol*), diversidade de serviços, entre outras. Neste cenário, a principal contribuição deste artigo é a proposta do InteropFrame, que através de sua arquitetura de *plugins* para os modelos de componentes e tipos de *binding* disponibiliza um *framework* extensível para interoperabilidade dinâmica entre componentes distribuídos heterogêneos.

O *framework* distribuído implementado pode servir de base para novas pesquisas mais abrangentes sobre interoperabilidade entre modelos de componentes, bem como para o desenvolvimento de outros mecanismos de interoperabilidade entre plataformas distintas.

A seguir são destacados alguns pontos relevantes do ambiente de execução implementado: *(i)* demonstra que a solução para interoperabilidade entre componentes distribuídos heterogêneos pode ser automatizada na prática (como demonstrado pelo InteropFrame), *(ii)* permite que o desenvolvedor escolha a forma de *binding* remoto mais apropriada para a comunicação distribuída, *(iii)* apresenta uma arquitetura para a solução que é flexível para suportar novos modelos de componentes e novos tipos de *binding* na forma de

plugins independentes, e *(iv)* viabiliza a reutilização de componentes heterogêneos na composição de sistemas distribuídos complexos.

Como trabalhos futuros, podem-se identificar algumas oportunidades, entre elas: *(i)* estender o InteropFrame para suportar componentes desenvolvidos com diferentes linguagens (a solução atual é para componentes Java), desenvolver, *plugins* para outros modelos (como EJB e o OSGi) e *plugins* para outros tipos de *bindings* (como *Web Services* RESTFul e XML-RPC), *(ii)* migrar a plataforma InteropFrame para o OSGi e avaliar o desempenho do *framework* causado pelo uso do OSGi, e *(iii)* projetar e implementar uma versão do InteropFrame para dispositivos com capacidade de processamento e recursos de computação limitados e providos de comunicação sem fio, como por exemplo, os celulares.

REFERÊNCIAS

BLAIR, G.; PAOLUCCI, M.; GRACE, P.; GEORGANTAS, N.. Interoperability in Complex Distributed Systems. In: Formal Methods for Eternal Networked Software Systems, ser. Lecture Notes in Computer Science, M. Bernardo and V. Issarn. **Springer**, v. 6659, p.1–26. Berlin Heidelberg, 2011. **DOI**: http://dx.doi.org/10.1007/978-3-642-21455-4_1

BROMBERG, Y.-D.; GRACE , P.; RÉVEILLÈRE , L.; BLAIR, G. S.. Bridging the Interoperability Gap: Overcoming Combined Application and Middleware Heterogeneity. In: Middleware 2011, ser. Lecture Notes in Computer Science, F. Kon and A.-M. Kermarrec. **Springer**, v.7049, p.390–409. Berlin Heidelberg, 2011. **DOI**: http://dx.doi.org/10.1007/978-3-642-25821-3_20

BRUNETON, E. et al.. An Open Component Model and Its Support in Java. Component-Based Software Engineering. **Springer**, p.7–22, 2004. **DOI**: http://dx.doi.org/10.1007/978-3-540-24774-6_3

BRUNETON, E.; COUPAYE, T.; LECLERCQ, M.; QUÉMA, V.; STEFANI, J-B.. The FRACTAL Component Model and its Support in Java. **Software: Practice and Experience**, v.36, n.11-12, p.1257–1284, 2006. **DOI**: http://dx.doi.org/10.1002/spe.767

COULOURIS, G. et al.. **DISTRIBUTED SYSTEMS**: Concepts and Design. 5 ed.. ed. Boston: Addison-Wesley, 2011.

COULSON, G.; BLAIR, G.; GRACE, P.; TAIANI, F.; JOOLIA, A.; LEE, K.; UEYAMA, J.; SIVAHARAN, T.. A Generic Component Model for Building Systems Software. **ACM Trans**. Comput. Syst., v.26, n.1, p.1:1–1:42, 2008. **DOI**: http://doi.acm.org/10.1145/1328671.1328672

CRNKOVIC, I.; STAFFORD, J.; SZYPERSKI, C.. Software Components beyond Programming: From Routines to Services. **IEEE Software**, v.28, p.22–26, 2011. **DOI**: http://dx.doi.org/10.1109/MS.2011.62

DEITEL, P.; DEITEL, H.. **Java**: como Programar. 8.ed. São Paulo: Pearson Prentice Hall, 2010.

GRACE, P.. **The OpemCOMJ Handbook**. Technical report, Distributed Multimedia Research Group. Lancaster University, Lancaster. 2007.

HAROLD, E. R.. **Java Network Programming**. 3rd. ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2004.

INVERARDI, P.; ISSARNY, V.; SPALAZZESE, R.. A Theory of Mediators for Eternal Connectors. In: Leveraging Applications of Formal Methods, Verification, and Validation, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen. **Springer**, v.6416, p.236–250. Berlin Heidelberg, 2010. **DOI**: http://dx.doi.org/10.1007/978-3-642-16561-0_25

OASIS, C.. **OASIS UDDI Specification TC**. 2012. Disponível: <http://www.oasis-open.org/committees/uddi-spec>. Acesso: 22 Out 2012.

OW2, C.. **The Fractal Component Model - Specification**. 2004. Disponível: <http://fractal.ow2.org/>. Acesso: 18 Out 2012.

PETERSON, L. L.; DAVIE, B. S.. **Computer Networks**: A Systems Approach, 4rd Edition. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

PRESSMAN, R. S.. **Engenharia de Software**. 6 ed. Porto Alegre: AMGH, 2010.

REILLY, D.; REILLY, M.. **Java Network Programming and Distributed Computing**. Boston, MA, USA: Addison-Wesley, 2002.

ROCHA, T.. **Serviços de Transação Abertos para Ambientes Dinâmicos**. Tese (Doutorado em Ciência da Computação) Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2008.

ROUVOY, R.; MERLE, P.. Leveraging Component-Based Software Engineering with Fraclet. Annals of Telecommunications, Special Issue on Software Components – The Fractal Initiative. **Springer Paris**, v.64, p.65–79, 2009. **DOI**: http://dx.doi.org/10.1007/s12243-008-0072-z

RUIXIAN, B.. **Distributed Computing via RMI and CORBA**. 2000. Disponível: <http://europepmc.org/abstract/CIT/497529>. Acesso: 02 Fev 2013.

SZYPERSKI, C.. **Component Software**: Beyond Object-Oriented Programming. [S.l.]: Addison-Wesley Professional, 1997.

W3C – World Wide Web Corsortium, 2012. Disponível: <http://www.w3.org/>. Acesso: 18 Jan 2013.