Sistema Distribuído de Imagens Médicas: Parte II – Avaliação de Desempenho de Diferentes Cenários

I. T. Pisa, E. E. S. Ruiz, M. Santos, A. J. Holanda
ImagCom – Computação de Imagem Médica, Departamento de Física e Matemática,
FFCLRP - USP, Ribeirão Preto, SP, Brasil, 14040-901, (16) 602-3774,
ivanpisa@dfm.ffclrp.usp.br

Resumo. Este artigo tem como objetivo apresentar uma avaliação de diferentes cenários de tecnologias *Object Web* que podem ser utilizados como base para a arquitetura de um sistema distribuído de imagens médicas. Os resultados indicam que a arquitetura CORBA apresenta o melhor desempenho. Foram considerados os seguintes requisitos: nível de abstração, integração com Java, suporte a diferentes plataformas, facilidade de configuração, uso de métodos distribuídos, capacidade de manter o estado das requisições, localização e evocação dinâmicas, tempo de acesso, uso de linguagem neutra, escalabilidade, padrão aberto etc.
Palavras-chave. Imagem Médica, Informática em Saúde, Sistemas Distribuídos, CORBA, Processamento de Imagem.

Introdução

A arquitetura de distribuição de objetos CORBA (*Common Object Request Broker Architecture*) apresenta uma solução para o desenvolvimento de sistemas distribuídos baseada num barramento conceitual de software que permite uma comunicação simples e robusta entre diferentes aplicativos independentemente de fabricante, plataforma, linguagem de desenvolvimento e da localização dos computadores. CORBA desempenha um importante papel na facilitação do desenvolvimento de aplicativos de última geração, incluindo interoperabilidade em qualquer tipo de rede de computadores. CORBA é provavelmente o projeto de interfaces mais importante e ambicioso desenvolvido pelas indústrias de computação para resolver paradigmas de distribuição de objetos[1,2,3]. A questão quanto à utilização de CORBA é: como CORBA compete com outras tecnologias cliente/servidor? Ou melhor, como CORBA se insere em diferentes cenários que incluem outras tecnologias cliente/servidor? O objetivo deste artigo é apresentar comentários e avaliação de desempenho de um objeto de requisição cliente/servidor de dados médicos em diferentes cenários.

Material e Método

A metodologia utilizada para implementar uma avaliação de desempenho em diferentes cenários foi desenvolver um objeto *query*/*retrieve* para uma base de imagens médicas. O objetivo inicial foi desenvolver um objeto que implementasse uma funcionalidade similar ao programa cliente/servidor descrito por Orfali[4] para avaliação de desempenho de diferentes cenários, porém, considerando um sistema distribuído de imagens médicas. O componente desenvolvido é bastante simples: consulta uma base de dados de imagens médicas buscando o nome de um possível paciente e retorna o nome do arquivo DICOM referente ao paciente.

Foi possível utilizar esse objeto para avaliar o desempenho médio de uma resposta dentro da arquitetura CORBA dividindo o tempo total transcorrido pelo número de métodos de requisição realizados. Foram comparadas diferentes configurações CORBA/Java cliente/servidor usando técnicas de requisição remota. O objeto foi utilizado por outros programas para comparar CORBA com outras tecnologias competidoras, incluindo Java RMI, Servlets, programação HTTP/CGI, Microsoft DCOM/COM+ e comunicação direta com Sockets. Testes locais foram realizados como processamento interno entre processos e testes remotos como processos sobre uma rede Ethernet 100Mbps. Utilizou-se como servidor um computador PIII 200MHz com Windows NT4 e como cliente um computador K6-II 500MHz com Windows 98 SE. Os testes locais foram realizados no computador servidor.

Resultados e Discussão

A. Evocação Estática CORBA

O programa foi executado em diferentes situações para avaliar o desempenho da evocação estática de métodos de CORBA. O programa cliente executou 1000 chamadas a métodos durante o tempo transcorrido. O programa calculou a média do tempo de resposta em milisegundos dividindo o tempo transcorrido pelo número de chamadas. Os ORBs utilizados foram Borland Visibroker for C++ 4.1 e Borland Visibroker for Java 3.3. Os compiladores utilizados foram JavaSoft JDK 1.1.5, Borland C++ 5.5 e Symantec Visual Café JIT. Os programas Netscape Communicator 4.x e Microsoft AppletViewer foram utilizados na execução das *applets*.

Cliente	Servidor	Local	Remoto
C++	Aplicativo Java Compilado	3,6 ms	3,4 ms
Aplicativo Java Compilado	Aplicativo Java Compilado	3,9 ms	3,6 ms
C++	C++	2,3 ms	3,9 ms
Aplicativo Java Compilado	C++	5,7 ms	4.3 ms
Aplicativo Java Interpretado	Aplicativo Java Interpretado	6.7 ms	5.6 ms
Java Applet Compilado	Aplicativo Java Compilado	16.1 ms	16.1 ms
Java Applet Interpretado	Aplicativo Java Interpretado	30.2 ms	23.3 ms

Tabela 1 – Desempenho de evocações estáticas.

A Tabela 1 demonstra que chamadas locais em C++ são as mais rápidas. O contrário ocorre com chamadas em Java, o que demonstra que a Máquina Virtual Java (JVM) utiliza intensivamente a CPU. O destaque é que os ORBs Java são tão rápidos quanto os ORBs C++. Nota-se também que o melhor desempenho remoto foi encontrado na situação servidor em Java compilado e um programa cliente em C++. A surpresa, em todos os casos, é que a execução remota foi mais rápida que a execução local. Uma explicação para esses resultados é que a execução do cliente e servidor na mesma máquina causa uma demanda maior de tempo pelo fato do sistema estar enviando e recebendo as requisições pelo mesmo ORB.

B. Evocação Dinâmica CORBA

O destaque da evocação dinâmica é que a localização do objeto é feita sob demanda, entretanto, o aumento na comunicação CORBA é bastante significativo. A avaliação do desempenho está na Tabela 2.

Essa diferença de desempenho ocorre porque antes de evocar dinamicamente um método de um objeto, primeiramente é necessário encontrar o objeto e obter sua referência. A partir de sua referência pode-se utilizá-la para recuperar a interface do objeto e dinamicamente construir a evocação do método. Deve-se, nesse caso, especificar na requisição o método que se deseja executar e seus parâmetros.

	Dinâmico Local	Dinâmico Remoto
Apenas Requisição	3,9 ms	3,6 ms
Preparação e Requisição	61,4 ms	57,8 ms

Tabela 2 – Desempenho de evocações dinâmicas.

C. Comparação com Competidores

A Tabela 3 apresenta uma comparação das mais importantes características de cada tecnologia *Object Web* avaliada, incluindo seu desempenho. Os indicadores utilizados na Tabela 3 são:

(a) nível de abstração: quanto maior esse nível, menos trabalho seu aplicativo fará. Os detalhes sobre requisição de nomes e parâmetros ficam por conta da tecnologia; Sockets apresentam o pior nível de abstração pois o programador deve criar por si mesmo suas convenções para passagem de nomes e seus argumentos;

(b) integração perfeita com Java: CORBA e RMI possibilitam a melhor integração em programação distribuída para Java. Ambos utilizam-se de interfaces Java para expor seus serviços. DCOM/COM+ introduz seu próprio IDL, que pode eventualmente integrar-se com objetos Java, mas não estendem seu modelo de objeto.

(c) suporte a diferentes plataformas: CORBA, HTTP/CGI e Sockets rodam em praticamente todos os sistemas operacionais cliente/servidor. RMI é escrito em Java, por isso roda em todas as plataformas que suportam Java. DCOM/COM+ é nativo somente para Windows NT/9x. Há maneiras de portá-lo para Unix através do uso de MTS. DCOM/Java está intimamente condicionado ao *Microsoft Virtual Machine*.

(d) parâmetros com tipos: todos os 3 ORBs suportam tipos na passagem de pa-

râmetros. DCOM e CORBA permitem *in*, *out* e *in/out*. RMI suporta apenas *in* mas permite passar classes como argumentos. HTTP/CGI e Servlets suportam funcionalidade mínima através de MIME para descrever o conteúdo da mensagem. Sockets não permite nenhuma funcionalidade. No entanto, Java permite associar Sockets com o tipo *streams*;

Função	CORBA/IIOP	DCOM/COM+	RMI/RMP	HTTP/CGI	Servlets	Sockets
Nível de Abstração	4	4	4	2	2	1
Integração Perfeita com Java	4	4	4	2	2	2
Diferentes Plataformas	4	3	4	4	4	4
Passagem de Parâmetros com Tipos	4	4	4	2	4	4
Facilidade de Configuração	3	3	3	3	3	3
Evocação de Métodos Distribuídos	4	3	3	0	0	0
Mantém Estado das Evocações	4	3	3	0	2	2
Descoberta Dinâmica e Meta-dados	4	3	2	0	1	0
Evocação Dinâmica	4	4	1	0	0	0
Tempo de Acesso Ping	3 3.5 ms	3 3.8 ms	3 3.3 ms	1 827 ms	1 55 ms	4 2.1 ms
Camada de Segurança	4	4	3	3	3	3
Camada de Transação	4	4	0	0	0	0
Persistência na Referência	4	3	4	0	0	0
Endereçamento URL	4	2	2	4	4	3
Uso de Linguagem Neutra	4	4	0	3	3	0
Escalabilidade	4	2	1	2	2	4
Padrão Aberto	4	2	2	4	2	4

Tabela 3 – Avaliação das características das 6 tecnologias *Object Web* mais importantes. Nota entre 0 (inexistente) e 4 (ótima).

(e) facilidade de configuração: todas as tecnologias obtiveram nota 3 pois não é uma tarefa simples configurar um sistema cliente/servidor;

(f) evocação de métodos distribuídos: CORBA teve nota máxima porque suporta referência única a objetos, permitindo reativação sob demanda;

(g) mantém estado das evocações: a falta dessa funcionalidade inviabiliza o uso de HTTP/CGI como sistema distribuído. Sockets mantém o estado entre evocações mas não há noção de estado de objeto. Os 3 ORBs mantém estado da manipulação de objetos. CORBA pode manter estados entre seções. CORBA permite reconectar ao mesmo objeto com o mesmo estado. Servlets permite reconectar à mesma instância em uma seção;

(h) descoberta dinâmica e suporte a meta-dados: somente CORBA e DCOM/COM+ suportam introspecção de objeto, ou seja, ambos permite que se descubra a interface do objeto dinamicamente. CORBA ainda suporta repositório de interfaces entre ORBs. DCOM/COM+ suporta apenas repositórios locais. RMI e Servlets suportam reflexão Java local. Entretanto, não é possível descobrir uma classe RMI remotamente. Porém, pode-se baixar uma descrição para uma classe RMI que permite introspecção. Mas isso não é substituto para um repositório de interfaces;

(i) evocação dinâmica: somente CORBA e DCOM/COM+ suportam evocação dinâmica, ou seja, construir o acesso a método em tempo de execução;

(j) tempo de acesso *ping*: Sockets possui o melhor desempenho, seguido pelos 3 ORBs;

(k) camada de segurança: Sockets permite o uso de SSL (*Secure Socket Layer*). HTTP suporta segurança via S-HTTP (*Secure-HTTP*) e SSL. DCOM/COM+ incorpora a segurança do Windows NT. CORBA define um serviço completo de segurança construído no próprio ORB. CORBA também pode utilizar SSL. A segurança em RMI é baseada em SSL mas também prevê o controle de segurança ao baixar classes;

(l) camada de transação: somente CORBA e DCOM/COM+ suportam transações distribuídas;

(m) persistência na referência de objetos: CORBA permite persistência de objetos e também persistência na referência de objetos. DCOM/COM+ utiliza-se de *moniker* para manter um objeto sem estado associado a algum contexto. RMI suporta persistência a referências desde JDK 1.2;

(n) endereçamento URL: é uma funcionalidade bastante útil em Internet e *intranets*. URL foi pioneiro com HTTP/CGI. As

portas de Sockets são integradas ao esquema URL. RMI e CORBA possuem esquemas URL que permitem associar endereço e objeto, porém, o endereço URL de RMI não é persistente mas o de CORBA é. DCOM/COM+ suporta endereço URL indiretamente;

(o) uso de linguagem neutra: DCOM/COM+ e CORBA suportam um formato canônico nas mensagens. HTTP também suporta auto-descrição via MIME. Entretanto, possui tipos muito limitados de dados. Servlets suportam HTTP;

(p) escalabilidade: Sockets via TCP/IP suporta qualquer tamanho de rede, porém, possui baixo nível de abstração. Em contraste, CORBA suporta qualquer rede de ORBs;

(q) padrão aberto: a tecnologia deve ser aberta e estar sujeita a evoluir de acordo com discussões e sugestões da comunidade. CORBA, Sockets e HTTP/CGI são padrões abertos. DCOM/COM+ é controlado pela Microsoft. JavaSoft controla RMI e Servlets mas são ratificados pelo ISO.

Conclusões

Através da avaliação de desempenho e das características existentes nas 6 tecnologias *Object Web* mais significativas conclui-se claramente que CORBA é a melhor tecnologia. O único competidor real de CORBA é o padrão DCOM/COM+. RMI é atualmente parte da tecnologia ORB até que IIOP assuma completamente sua funcionalidade. A utilização de Sockets representa apenas um protocolo de baixo nível. A combinação HTTP/CGI é largamente empregada, porém, não é uma solução de fato para a distribuição de objetos na rede pois fundamentalmente não armazena o estado da comunicação e nem permite funcionalidade dinâmica. Portanto, a avaliação demonstra que é CORBA versus DCOM/COM+.

Nos últimos anos diversas empresas da área de informática se juntaram a OMG com a intenção de utilizar CORBA para desenvolver produtos, não apenas um padrão, entre eles Netscape, Oracle e JavaSoft. Outras empresas significativas já fazem parte da OMG há bastante tempo, como IBM, Novell, Borland/Visigenic, SunSoft e Iona. Juntas essas empresas formam um excelente bloco de desenvolvimento. A arquitetura CORBA já tem sido utilizada em áreas como engenharia aeronáutica, automação bancária e comercial, gerenciamento de empresas de aviação civil etc. Dentro das áreas de Informática em Saúde, Bibliotecas Digitais e Sistemas Distribuídos de Imagens Médicas pode-se constatar uma tendência de utilização de tecnologias *Object Web*, em especial, a arquitetura CORBA[5].

Esse trabalho faz parte das atividades do grupo ImagCom cujos objetivos incluem produzir um sistema funcional e distribuído de imagens médicas com visualizadores de diferentes formatos de imagens, sistema de busca inteligente de imagens e serviços distribuídos de processamento de imagens.

Abstract. The objective of this article is to present an Object Web technologies multi-scenario benchmark that can be used to implement architecture for medical images distributed system. The results show that CORBA presents the best performance. The requisites considered are abstraction level, seamless Java integration, OS platform support, ease of configuration, distributed method invocations, state across invocations, dynamic discovery and metadata support, language-neutral wire protocol, scaling, open standard etc.

Referências

[1]S. Vinoski, “CORBA: Integrating diverse applications within distributed heterogeneous environments,” in IEEE Communications, vol. 14, no. 2, Feb. 1997.

[2]T. J. Mowbray, W. A. Ruth, Inside CORBA: Distributed Object Standards and Applications, Addison Wesley, 1997.

[3]K. Seetharaman, “The CORBA Connection,” in Communications of the ACM, vol. 41, no. 10, October 1998.

[4]R. Orfali, D. Harkey, Client/Server Programming with Java and CORBA, 2nd. edition, 1998, ISBN 0-471-24578-X, pp. 3-379.

[5] R. D. Cox, C. J. Henry, R. K. Rubin, “Transparent Image Access in a Distributed Picture Archiving and Communications System: The Master Database Broker,” in Journal of Digital Imaging, vol 12, no 2, suppl 1 (May), 1999: pp 175-177.