6ª Conferência da Associação Portuguesa de Sistemas de Informação
26, 27 e 28 de Outubro de 2005
Bragança – Portugal

Aumento da resiliência dos Web services com uma Infra-estrutura *Peer-to-Peer*

Martinho Correia
Universidade da Madeira, Funchal, Portugal
martinho@netmadeira.com

Jorge Cardoso
Universidade da Madeira, Funchal, Portugal
jcardoso@uma.pt

Resumo

Actualmente muitas organizações sustentam uma parte significativa dos seus processos de negócio nas Tecnologias da Informação (TIs). Sensivelmente no início do novo milénio verificou-se que a ubiquidade da *Web* é um obstáculo aos requisitos de rápida integração de aplicações. Como solução para este problema surge a tecnologia *Web Services*. Esta tecnologia simplifica o desenvolvimento de aplicações distribuídas com base num conjunto de protocolos, contudo, nas suas especificações não fornece nenhuma forma para cobrir eventuais falhas que possam ocorrer no fornecimento de serviços ao nível do sistema sob o qual os *Web services* são desenvolvidos. Este trabalho pretende explorar como uma infra-estrutura *Peer-to-Peer* pode ser usada para aumentar a disponibilidade de *Web services*. Foi feita uma implementação desta infra-estrutura usando tecnologias *state-of-the-art* das áreas *Web services* e *Peer-to-Peer*.

Palavras chave: *Web Services*; *Peer-to-Peer*; resiliência a falhas; JXTA.

1 INTRODUÇÃO

Os *Web Services* são uma componente importante de muitos sistemas *Business-to-Business* (B2B) e *Business-to-Customer* (B2C). Em muitos casos os sistemas B2B e B2C assumem uma importância bastante significativa para as organizações. A quebra de serviços pontuais pode facilmente comprometer a conclusão de um processo de negócio em curso e períodos de indisponibilidade em tais sistemas podem ter um impacto financeiro significativo. Por isso, é frequente a definição de mínimos de disponibilidade ao nível da qualidade de serviço de sistemas B2B e B2C. A disponibilidade e escalabilidade nos sistemas distribuídos são normalmente obtidas através de replicação (por exemplo *clusters* de servidores). Estes mecanismos podem representar investimentos consideráveis em arquitecturas distribuídas convencionais. Nas tecnologias *Peer-to-Peer* a disponibilidade é um ponto forte uma vez que todos os nós da rede são potenciais fornecedores de serviços e também porque podem ser acrescentados nós à rede de uma forma dinâmica e natural. O nosso objectivo é propor um método transparente que aumente a disponibilidade dos *Web services*. A partir deste propósito implementámos uma arquitectura que combina Web services e uma insfra-estrutura *Peer-to-Peer*. Apresentamos neste documento detalhes do desenho e implementação do nosso método de resiliência a falhas e demonstramos a sua utilidade através de um cenário típico de utilização.

Este artigo está estruturado da seguinte forma. Na Secção 2 introduzimos a Tecnologia Web Services e a arquitectura *Peer-to-Peer*. Ainda nesta secção apresentamos uma comparação entre estas duas tecnologias. Na secção 3 introduzimos o conceito de tolerância a falhas em sistemas distribuídos, e em particular na tecnologia Web services. Na secção 4 apresentamos a nossa arquitectura e questões relacionadas com a integração entre Web Services e a tecnologia Peer-to-Peer usada na implementação: JXTA (juxta). Na secção 4 são ainda apresentados os conceitos técnicos da tecnologia JXTA, assim como questões relevantes da implementação da nossa

arquitectura. Na secção 5 abordamos o trabalho relacionado e finalmente na Secção 6 apresentamos conclusões.

2 WEB SERVICES E ARQUIECTURAS PEER-TO-PEER

Esta secção introduz a tecnologia Web Services e a arquitectura Peer-to-Peer.

2.1 Web Services

Web Services é uma tecnologia que tem vindo a ganhar cada vez mais popularidade no meio empresarial devido ao aumento de eficiência e redução de custos resultantes da integração de aplicações. Integração de aplicações da própria organização, ou integração de aplicações entre parceiros de negócio. Por exemplo, os Web services permitem que se possa integrar facilmente diferentes aplicações, de diferentes parceiros, quando um parceiro numa dada área deixou de fornecer as melhores condições. A tecnologia Web Services tem por objectivo principal a integração e interoperabilidade. Para atingir integração e interoperabilidade esta tecnologia usa como base para as comunicações o XML (eXtensible Markup Language). A partir do XML define um padrão para descrição de serviços, WSDL – Web Service Description Language [W3c WSDL spec], e um protocolo de invocação de serviços, Simple Object Access Protocol (SOAP) [W3c SOAP standards]. Através de uma interface WSDL, um serviço pode ser registado, e posteriormente encontrado, num directório de serviços usando UDDI [UDDI standards] – Universal Description for Discovery and Integration.

2.2 Arquitecturas peer-to-peer

Peer services são serviços fornecidos por pares, i. é, iguais entre si, num sistema distribuído. Em oposição a este tipo de arquitecturas distribuídas temos as arquitecturas cliente/servidor, usadas actualmente na maioria dos sistemas distribuídos. A arquitectura *Peer-to-Peer* [Dejan S. et al, 2002] promete ser alternativa à arquitectura Cliente/Servidor. Uma razão é que esta arquitectura traduz a essência de como as interacções acontecem na Web. Um utilizador quer encontrar um recurso, ou um serviço, e quer usar esse recurso ou serviço ligando-se directamente ao nó de rede que o disponibiliza. Na maioria das arquitecturas actuais, esse nó de rede, com o serviço procurado, é um servidor. Nas arquitecturas Peer-to-Peer qualquer nó de rede pode disponibilizar serviços. Os sistemas distribuídos peer-to-peer são escaláveis, provêem maior facilidade na implementação de mecanismos de tolerância a falhas e equilíbrio de carga da rede, etc.

A tecnologia *peer-to-peer* tem estado em observação desde 2001, devido à explosão dos sistemas de partilha de ficheiros como Napster [Shawn Fanning], Gnutella [Justin Frankel]. No entanto as potencialidades *peer-to-peer* vão além da partilha de ficheiros, como mostram desenvolvimentos noutras áreas como por exemplo: distribuição do poder de computação (redes Grid, SETI@Home - Search for Extra Terrestrial Inteligence at Home); *instant messaging*, *groupware*/CSCW (Computer Suported Cooperative Work); integração *peer-to-peer*/Web Services. Contudo as redes Peer-to-Peer apresentam ainda limitações a nível da segurança e largura de banda. Segurança devido ao aumento de possíveis pontos fracos. Um nó de rede pode ligar-se a qualquer outro nó de rede, havendo mais complexidade no desenvolvimento de sistemas de segurança. Por outro lado a largura de banda é em parte ocupada por mensagens de anúncios de serviços e mensagens de procura destes mesmos serviços, que têm de ser lançadas a vários nós de rede. Nestes últimos anos estas limitações das arquitecturas *Peer-to-Peer* têm vindo a desaparecer com o aumento da largura de banda comum, e com evolução em segurança como encriptação e chaves de identificação. Isto faz com que a tecnologia *Peer-to-Peer* seja um dos temas mais quentes e actuais da computação distribuída.

2.3 Comparação entre Web services e arquitectura Peer-to-Peer

As tecnologias *Peer-to-Peer* são baseadas em modelos descentralizados, cujo foco principal é o fornecimento de poder de computação, conteúdos, ou aplicações a nós de rede de uma forma distribuída. Um foco secundário destas tecnologias é a instituição de protocolos de comunicação e formato das mensagens que circulam num sistema distribuído. Por outro lado, a tecnologia Web Services, é baseada num modelo centralizado e é focada principalmente na padronização de formatos de mensagens e protocolos de comunicação.

Interoperabilidade - a um nível alto ambas as tecnologias apresentam soluções de interoperabilidade para problemas em domínios diferentes. Web Services procura ser solução para a heterogeneidade na Web, "escondendo" diferenças de aplicações e plataformas por detrás de serviços descritos e invocados de uma forma padrão. JXTA procura ser uma plataforma padrão para desenvolvimento de aplicações *peer-to-peer* e a sua arquitectura foi pensada para interoperabilidade. Por exemplo, os Web services e os *peer services* ultrapassam *firewalls* e NAT (Network Address Translation) de forma semelhante. Nos Web services o protocolo de rede usado é o TCP/IP e o protocolo de transporte é o HTTP. Tanto a tecnologia Web Services como a implementação Java da tecnologia JXTA usam o protocolo HTTP e o porto 80 para ultrapassar *firewalls* e NAT (Network Address Translation).

Descoberta e anúncio de serviços - em Web Services o anúncio de serviços é feito através de um repositório central usando a tecnologia UDDI. Nas tecnologias *Peer-to-Peer* os serviços são anunciados por difusão na rede. Anúncios de serviços podem ser encontrados em *peers* que estejam próximos ou em peers que de algum modo são comparáveis a directórios UDDI de serviços, *rendezvous peers*.

3 TOLERÂNCIA A FALHAS

Nesta secção introduzimos o tema da tolerância a falhas nos sistemas distribuídos e na tecnologia Web Services em particular.

A tolerância a falhas é a capacidade de um sistema continuar a disponibilizar serviços na presença de falhas, que possam ocorrer devido a erros internos ao sistema, ou por influência do ambiente envolvente do sistema. Estas falhas podem surgir, por exemplo, devido a indisponibilidade de comunicações, *crash* de um processador, *bugs* etc. Os mecanismos de tolerância a falhas devem ser transparentes, introduzir pouca complexidade de gestão, ser portáveis, e escaláveis. A transparência significa que existe um mecanismo tal que as aplicações que o implementem podem, de uma forma significativa, ignorar falhas e subsequentes recuperações de processos, uma vez que estas questões são tratadas pelo próprio mecanismo. Normalmente a tolerância a falhas em sistemas distribuídos é feita com replicação. Vários processos com o mesmo estado são replicados de forma a substituírem processos que falhem.

3.1 Mecanismos para tratamento de falhas em Web Services

A nível conceptual a tecnologia Web Services dispõe de alguns tipos de tratamento de erros. Na camada da descrição de serviços, o WSDL provê um mecanismo através de <wsdl:fault> para as aplicações poderem especificar características de erros. É semelhante ao tratamento de excepções na interface Java. Igualmente, a camada de troca de mensagens SOAP, provê <soap:fault> para aplicações comunicarem informações sobre erros. Os mecanismos de tratamento de erros dos protocolos SOAP e WSDL, ajudam a gerir os erros gerados por uma aplicação mas não permitem o tratamento de erros ocorridos no sistema, ou na plataforma da qual a aplicação depende para funcionar.

4 INTEGRAÇÃO WEB SERVICES/ INFRAESTRUTURA *PEER-TO-PEER*

4.1 Arquitectura

A arquitectura do nosso sistema na figura 1 é composta por três secções. A secção **A** é a secção de clientes de Web services. Clientes usam a tecnologia Web Services para procurarem serviços (através de UDDI). Após saberem que fornecedores existem para um determinado serviço podem ligar-se directamente a um servidor com o serviço, usando uma qualquer implementação da tecnologia Web Services. Um fornecedor deste serviço pode ter sido implementado sobre a nossa infra-estrutura, como é caso dos Web Services na secção **B**. Vários serviços podem ser disponibilizados pelo Web Server (Service1, Service2, …). O Web Server disponibiliza uma interface WSDL do serviço, o serviço em sí é fornecido por um grupo de *peers* na secção **C**. A secção **C** é uma rede *peer-to-peer* que disponibiliza os serviços definidos pelas interfaces WSDL na secção **B**. Cada *peer*, na rede da secção **B**, disponibiliza o "mesmo" serviço, ou pelo menos o resultado desse serviço é satisfatoriamente o mesmo. Ainda na secção **B,** é iniciado um proxy, "dentro" da interface do Web service, que invoca a execução do serviço JXTA. Entre as duas tecnologias, o *proxy* vai se ligar ao grupo ao qual está implicitamente associado programaticamente, ou, alternativamente pode procurar um qualquer grupo que forneça o serviço pretendido, por exemplo, através de um par atributo/valor.

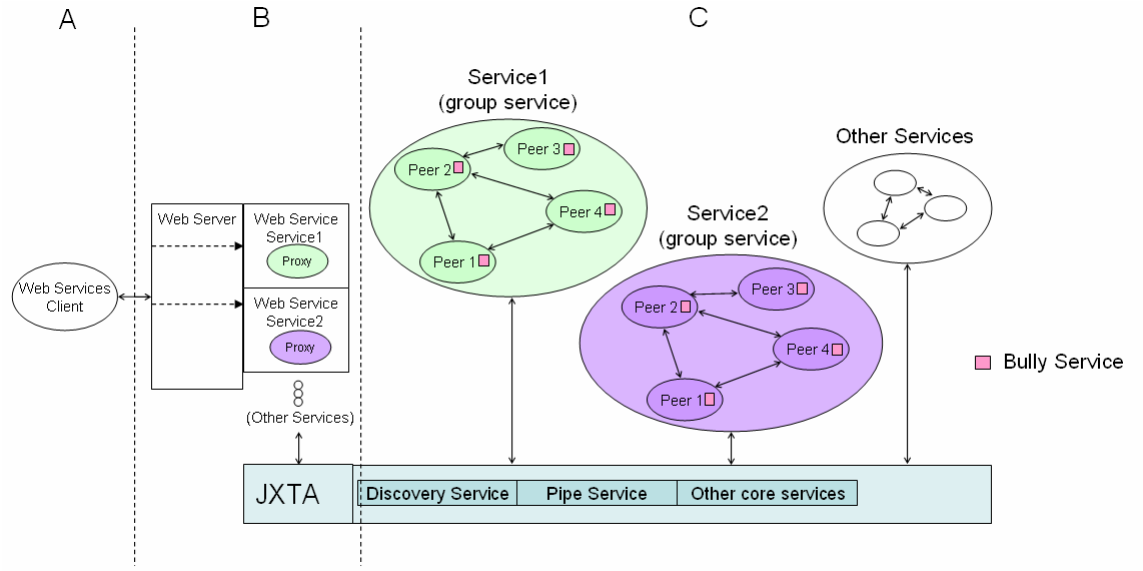

Figura 1 Arquitectura

Mais detalhadamente os elementos desta arquitectura são:

Proxy: O *proxy* é uma aplicação iniciada pelo Web service que "fala" JXTA. Permite relegar a execução do Web service para a infra-estrutura JXTA e receber um resultado. O *proxy* faz assim o mapeamento entre serviços descritos usando WSDL da tecnologia Web Services e serviços JXTA. Esta questão é apresentada mais detalhadamente na secção 4.5 Questões de implementação.

Peer: um *peer* é um nó de uma rede *peer-to-peer*. Na nossa arquitectura temos dois tipos de peers: o *proxy* e *peer* de grupo. Este último fornece serviços replicados. Cada *peer* pode conter uma implementação diferente do mesmo serviço. Por Exemplo, dois *peers* podem fornecer o mesmo serviço de acesso a uma base de dados relacional usando tecnologias diferentes, C++/ODBC ou Java/JDBC.

Group service: um grupo é composto de *peers* que disponibilizam um dos serviços definidos no servidor de Web services. A cada Web service corresponde pelo menos um grupo.

Bully Service: bullyservice é um serviço que todos os grupos da infra-estrutura implementam. Trata-se do algoritmo Bully para eleição de um coordenador e é executado no âmbito de um grupo. Cada *peer* tem um Id associado. O algoritmo garante simplesmente que o *peer* com maior Id será o *peer* fornecedor de serviços, se este *peer* falhar o algoritmo garante que o *peer* activo com Id imediatamente inferior será o novo fornecedor do serviço.

DiscoveryService: DiscoveryService é um *core service* de qualquer implementação dos protocolos JXTA. É especificado pelo Peer Discovery Protocol. Todos os grupos dispõem deste serviço para que os seus *peers* possam descobrir outros *peers***,** *groups, pipes,* sob a forma de anúncios XML.

PipeService: PipeService é *core service* JXTA que permite que dois *peers* estabeleçam um canal de comunicação.

Web Services Client: é uma aplicação que invoca um Web service. É uma aplicação que usa a tecnologia Web Services para aceder a um fornecedor de um Web service. A aplicação cliente pode conhecer o endereço e interface do serviço ou procurar o serviço através de um registo UDDI.

Web Server: É um servidor de JSPs, Servlets ou Web services. Na nossa implementação foi usado Tomcat 5.x e Axis 1.x.

Usamos JXTA por esta tecnologia, para além de ser uma grande aposta de padronização, oferecer também os mecanismos necessários para a implementação da infra-estrutura. O conceito de grupo é um conceito bem estudado na tecnologia JXTA. O âmbito de um grupo é útil para implementação de uma camada de QoS como a procura de serviços por par atributo/valor, mecanismos de autenticação, segurança, agregação de serviços, etc. A disponibilização de serviços de grupo, *group services*, permite que cada *peer* que se junte a esse grupo possa fornecer os serviços definidos para esse grupo. A qualquer altura um *peer* pode ser introduzido, ou retirado, da rede para fornecer mais ou menos disponibilidade de serviços. Os serviços que esse grupo implementa podem ser também implementados por um novo *peer*, ainda que este não conheça a implementação do serviço. Estas ideias são exemplificadas na secção 4.5 Questões de implementação.

Características da Arquitectura

A arquitectura apresentada na subsecção anterior possui como principais características: interoperabilidade, que herda da tecnologia Web Services, sendo facilmente integrável com outros Web services; escalabilidade por ser fácil acrescentar *peers* num grupo de um qualquer serviço; transparência porque o tratamento de falhas é feito independentemente do cliente do Web service; e apresenta um *low overhead* na medida em que a tecnologia JXTA usada possui um conjunto de características que tornam fácil a implementação do sistema. Discutimos em mais detalhe esta última característica na secção que se segue.

4.2 JXTA Core building blocks

JXTA, pronunciado *juxta*, de *juxtapose* - justaposto, é uma plataforma de programação de redes, introduzida em 2001 pela SUN Microsystems, e desenhada para resolver uma série de problemas da computação distribuída moderna, especialmente na área das redes *peer-to-peer*. A tecnologia JXTA tem como principal objectivo fornecer uma plataforma com as funções básicas para o desenvolvimento de redes *peer-to-peer*. Além disso pretende ter como principais características: interoperabilidade entre diferentes *peers* que fornecem serviços *peer-to-peer*; independência da plataforma em relação a linguagens de programação, protocolos de transporte ou plataformas de distribuição; ubiquidade, permitindo que qualquer dispositivo com um *heartbeat* digital possa ser acessível. Os conceitos JXTA incluem *peers*, *peergroups*, *pipes*, *messages*, *endpoints*, serviços, *advertisements*, *modules*, *rendezvous* e segurança. Um *peer* é qualquer dispositivo na rede capaz de fazer computações.

Um *peergroup* é um conjunto de *peers* que partilham o mesmo conjunto de serviços e recursos. Tipicamente estes grupos são criados com base no interesse mútuo dos *peers*. *Peers* interessados num certo conjunto de serviços vão se agrupar, formando um âmbito de procura. Um *peergroup* pode ainda servir de âmbito para autenticação e monitorização.

No protocolo de transporte de rede temos os conceitos de *endpoints*, *pipes* e *messages*. *Endpoints* são interfaces de rede que indicam a fonte/destinatário de qualquer peça de informação trocada. *Pipes* (dutos) são canais virtuais de comunicação unidireccionais e assíncronos entre dois *endpoints*. *Messages* são *containers* para dados transmitidos num *pipe* de um *endpoint* para outro.

Serviços são funcionalidades que um *peer* pode invocar remotamente para obter um resultado útil. Podemos dividir serviços JXTA em duas categorias: *peer services* e *group services. Peer services* são serviços associados exclusivamente a um *peer*. Quando esse *peer* sai da rede os serviços desse *peer* deixam de estar disponíveis se nenhuma replicação explícita for feita. *Groups services* são funcionalidades que um grupo oferece a qualquer membro desse grupo. Esta funcionalidade pode ser disponibilizada por vários *peers* permitindo redundância. Desde que um *peer* esteja num *peer group* os serviços desse grupo estão disponíveis. A plataforma JXTA fornece um conjunto de serviços fundamentais para a gestão do grupo como: *Discovery Service, Membership Service, Access Service, Pipe Service, Resolver Service, Monitoring Service*. O utilizador pode definir serviços específicos num grupo.

Advertisements são anúncios XML de uma determinada entidade na rede. *Peers*, *peer groups*, *pipes* e *services* são anunciados na rede JXTA usando *advertisements*.

Um *Module* é um pedaço de código (funcionalidade) que pode ser carregado e instanciado num *peer* dinamicamente (em *runtime*). Os *modules* são publicados num *peer group* usando *advertisements* que definem o seu comportamento geral, uma especificação e uma implementação. Um *peer* pode carregar e instanciar uma implementação já existente noutro *peer,* ou diferentes implementações da mesma funcionalidade podem ser feitas em diferentes linguagens, em diferentes contextos.

Rendezvous peers são peers que agem como um ponto de encontro para os anúncios de serviços e recursos ou simples presença de *peers* comuns. *Relay peers* são *peers* que permitem a rede alargar-se para além de *firewalls* e NAT (Network Adress Translaction).

A segurança na tecnologia JXTA é fornecida pela possibilidade de encriptação de mensagens à saída de cada *endpoint,* permitindo integridade, autenticação e confidencialidade.

4.3 Bully: Algoritmo de eleição de coordenador

Por vezes é necessário haver uma entidade coordenadora em sistemas distribuídos. Isto implica o uso de algoritmos distribuídos, i. é, algoritmos onde a decisão é descentralizada. O Algortimo *Bully* [Garcia Molina 1982] é um algoritmo clássico da computação distribuída que aborda o problema da eleição de um coordenador num sistema distribuído. Foram propostas algumas alterações desde então [Mamum e tal 2004, Stoller 1997]. O algoritmo *Bully* define para cada *peer* uma prioridade (peso ou simples Id). O *peer* com Id mais elevado deve ser o coordenador. Se um *peer* detectar que o coordenador falhou deve iniciar uma eleição. Se não houver nenhum nó na rede com maior Id, este *peer* é o segundo nó com maior Id, e é eleito coordenador. Se não, espera que um nó com maior Id se eleja como coordenador. Cada nó sabe o ID de todos os outros nós no sistema distribuído. Todos os *peers* podem iniciar uma eleição se detectarem que o coordenador não responde a um pedido de *acknowledge*. Um nó de rede pode estar num de três estados: recém recuperado – desconhece o coordenador, em estado de eleição e em estado normal. Em certas situações poderá ser acrescentado o estado de recuperação, onde o sistema redistribui tarefas pelos *peers*. Cada nó no estado normal (e de recuperação se for o caso) sabe o ID do coordenador corrente. Este algoritmo é síncrono na medida em que tempos de espera são definidos dentro dos quais respostas têm de ser recebidas.

No nosso sistema o algoritmo Bully foi usado para eleger o *peer* fornecedor de serviços activo. Na presença de erros que originem a falha de disponibilidade do *peer* fornecedor, a execução do algoritmo garante que outro *peer* ficará responsável pelo fornecimento do mesmo serviço.

4.4 Questões de implementação/Protótipo

A implementação foi feita usando a linguagem Java, a plataforma JXTA, e a tecnologia Web Services através de Tomcat/Axis. A linguagem Java e a plataforma JXTA de desenvolvimento *Peer-to-Peer* foram escolhidas devido às características de interoperabilidade e independência da plataforma, importantes em aplicações Web. Tomcat foi excolhido por ser um servidor simples e leve, juntamente com Axis, a extensão de Tomcat para Web services.

Interacção Web Services/JXTA

No servidor Tomcat/Axis é colocado o Web service. O Web service engloba um *proxy* que faz a "tradução" entre Web services e JXTA. Na implementação do *proxy* deparámo-nos primeiro com uma questão ligada a decisões de arquitectura da interface Java da tecnologia JXTA. O protocolo define NetPeerGroup como o PeerGroup global que permite descoberta de *peers* e troca de mensagens na rede JXTA global. O NetPeerGroup é uma instância única (*singleton)* da classe PeerGroup. Por outro lado o netPeerGroup tem de ser instanciado no *bootstrap* de cada *peer*. Como cada vez que o *Web service* é invocado, o *proxy* também o é, há simplesmente que garantir que apenas uma instância de netPeerGroup possa ser criada usando o modificador *static* juntamente com o método seguinte para criar o NetPeerGroup.

```
private synchronized void init() throws PeerGroupException {
    try {
        if (netPeerGroup == null)
                netPeerGroup = PeerGroupFactory.newNetPeerGroup();
    } catch (Exception e) { …}
}
```

Descrição de Web services e de Peer services.

Na tecnologia Web Services define-se um serviço e os seus detalhes de invocação com uma interface WSDL. A implementação deste serviço é feita introduzindo código com a lógica do serviço, usando por exemplo ferramentas como java2WSDL que permitem compor facilmente o Web service.

Na tecnologia JXTA os serviços são definidos geralmente através de três tipos de anúncios na rede: o *module class advertisement*, *module specification advertisement* e o *module implementation advertisement*. Esta estrutura foi desenhada para a implementação de funcionalidade de uma forma extensível na rede JXTA. O *module class advertisement* não fornece informação sobre a implementação do serviço, a sua função é apenas anunciar a existência de uma classe que é um módulo de funcionalidade. Entre a informação que contém destaca-se o MCID (module *class ID*), um nome e uma descrição do modulo. O *module specification advertisement* existe como camada intermédia para especificar compatibilidade entre um conjunto de módulos. Entre os atributos do *module specification advertisement* destacam-se o MSID, que contém o MCID já referido acima, informação sobre a versão da especificação e parâmetros da especificação. Finalmente, o *module implementation advertisement* contém informação sobre a implementação. Para isso este documento XML contém como elementos o MSID da especificação implementada, contém também informação necessária para correr o código, podendo até fornecer o próprio código, ou informação sobre um local de onde código pode ser descarregado. O código pode ser um ficheiro JAR ou um ficheiro class. A especificação JXTA para a descrição de serviços foi feita em três níveis para facilitar a interoperabilidade entre plataformas distintas. Uma determinada funcionalidade anunciada em abstracto pelo *module class advertisement*, poderá ter várias implementações anunciadas na rede pelo *module implementation advertisement*. Num nível intermédio o *module specification*

advertisement contém informação que garante compatibilidade entre estas diferentes implementações, que podem ser efectuadas em linguagens e plataformas diferentes e por vezes incompatíveis.

Implementação do Algoritmo Bully

O algoritmo Bully tem por objectivo a eleição de um *peer*, de entre os *peers de grupo*, que deverá responder ao pedido do serviço. Todos os *peers* que fornecem serviços implementam e participam no algoritmo no âmbito de um grupo. O *proxy* quando invocado deve saber que *peer* está a responder a pedidos. O *proxy* não participa no algoritmo e pode não ter informação actualizada sobre alterações que tenham ocorrido nos períodos em que nenhum serviço é invocado. Perante isto duas soluções surgiram. i) A implementação de um *listener* que acompanhe o início do servidor do Web service e que receba mensagens de alterações do coordenador, ou ii) o armazenamento de informações sobre o coordenador o pedido, por exemplo num ficheiro. As informações são obtidas sempre que um serviço é invocado, e um pedido de serviço ao coordenador corrente é feito. Se o coordenador não responde o *proxy* envia a todos os *peers* na rede um pedido de informação até receber uma resposta. Se não recebe uma resposta, entra em espera e volta a tentar após um período de tempo até que haja um coordenador. A solução i) é melhor pois promove a performance do sistema uma vez que o proxy não precisa de perguntar por uma eventual mudança de *peer* fornecedor do serviço. A solução ii) é de mais fácil implementação.

As redes *Peer-to-Peer* caracterizam-se pelo seu dinamismo. Um nó de rede pode entrar e sair da rede de uma forma imprevisível. Um *peer* pode sair da rede originando alterações em caminhos de rede e alterando a conectividade da rede. Para abordar o problema da eleição de um coordenador em redes Peer-to-Peer, foi proposta a plataforma ACE (Adaptative Cordinator Election) [Yoshinaga et al 2004]. Esta plataforma usa um algoritmo de eleição que tem em consideração métricas que para a escolha de um coordenador. Além de um Id, um nó de rede pode ser avaliado por condições ambientais como largura de banda, características próprias como capacidade de processamento, memória etc, ou histórico, como número de *crashes* num determinado período.

Benchmarking

Nesta secção apresentamos conclusões das medições da escalabilidade do algoritmo e dos tempos de latência de invocação do Web service

O *Benchmarking* para este tipo de sistemas estuda normalmente características como a performance e escalabilidade. Para a tecnologia JXTA são apresentadas análises de performance e escalabilidade em [JXTA bench]. A nossa infra-estrutura JXTA implementa um algoritmo distribuído de troca de mensagens pelo que interessa saber a linearidade na variação do número de mensagens com o número de *peers* no algoritmo. Para a medição do número de mensagens introduzidas pelo algoritmo usámos uma rede Ethernet 100Mb/s com 15 computadores 1GHz/128Mb e o sistema operativo Windows XP. Foi usada a versão 2.3.2 da implementação Java da tecnologia JXTA para desenvolver o sistema. Na figura 2 é apresentada uma representação gráfica do número de mensagens com o número de *peers* de grupo. Foram injectadas falhas para diminuir o número de *peers* sistema de uma forma controlada.

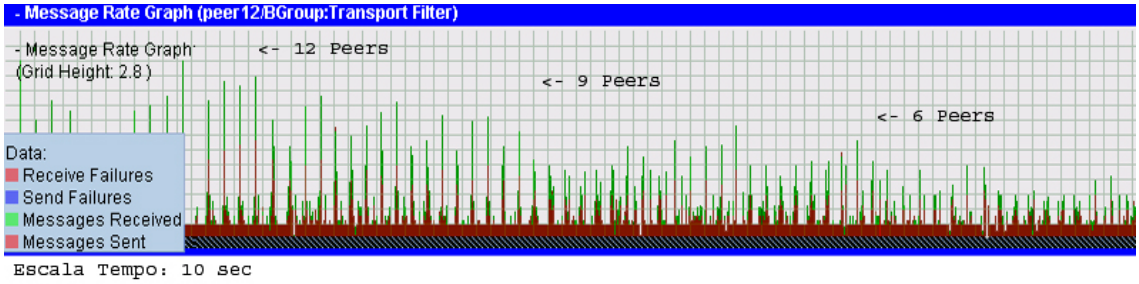

Figura 2 – Variação do número de mensagens com o número de *peers* de grupo.

Verificou-se que a taxa de crescimento de mensagens (mensagens de funcionamento da tecnologia JXTA + mensagens do sistema) é linear com o aumento de *peers* fornecedores.

Para medir os tempos de latência de invocação do Web service usámos *Round-Trip Time* (RTT). O RTT pode ser definido simplesmente como o tempo entre o envio de um pedido e recepção de um resultado. Verificámos que o tempo de latência médio é da ordem dos 0,5 segundos. Contudo na pior situação de alterações do algoritmo o RTT pode chegar às várias unidades de segundo. Na pior situação há uma falha no fornecedor corrente do serviço que é detectada pelo *peer* com menor Id, o qual envia mensagens de eleição para todos o *peers* com Id superior. A fraca performance para este caso deve-se, por um lado ao tempo necessário para a eleição de um novo *peer* fornecedor serviços, e por outro à operação de *binding* para criação de um novo *pipe* entre o *proxy* e o novo fornecedor.

5 TRABALHO RELACIONADO

Esta secção apresenta referências a trabalhos de investigação sobre mecanismos de tolerância a falhas em Web services.

[Dialani et al 2002] propõe uma arquitectura padrão para o desenvolvimento de Web services tolerantes a falhas. O mecanismo de tolerância a falhas é apresentado como uma faixa vertical na arquitectura dos Web services e criadores de um Web service devem implementar uma interface de *checkpoint* e *rollback*. *Checkpoint* para armazenar um último estado correcto do sistema, e *rollback* para, em caso de falha, repor o sistema nesse último estado correcto. O mecanismo estende dinamicamente a interface de definição de serviço (WSDL) com métodos para tolerância a falhas e os Web services devem declarar as suas interdependências para que o mecanismo possa controlar a recuperação de falhas de composições de Web services.

Em [Looker e tal, 2005] é proposto o WS-FTM (*Web Service-Fault Tolerance Mechanism)* que é uma implementação, para Web Services, do modelo N-version para tolerância a falhas em sistemas distribuidos. O modelo N-version é um padrão de desenho que permite que as falhas sejam tratadas por redundância. Como na maioria dos sistemas de tolerância a falhas, no modelo N-version é usada replicação, mas, cada réplica é implementada numa versão diferente para evitar erros comuns, resultantes de características comuns a uma dada implementação. Todas as implementações correm em paralelo e existe um mecanismo de votação para garantir um nível de integridade nos resultados obtidos de cada uma das N versões de implementação do serviço.

Outro artigo [Tartanoglu 2003] apresenta uma recolha sobre a forma como implementar sistemas de tolerância a falhas em Web services. Este artigo aborda os mecanismos *backward error recovery* e *forward error recovery*. Usando *backward error recovery* o sistema é reposto num último estado correcto. Na *forward error recovery* o sistema é conduzido de um estado erróneo para um estado válido. Neste artigo é concluído que os mecanismos de *forward error recovery* são mais adequados para implementação de mecanismos de tolerância a falhas em Web services num contexto de composição de Web services. A *forward error recovery* seria mais adequada aos mecanismos de tolerância a falhas em sistemas distribuídos fechados.

Como método de resiliência a falhas este trabalho situa-se a montante da problemática da tolerância a falhas de Web services. O contributo do nosso método é explorar as características das redes *Peer-to-Peer* como base para o desenvolvimento um método transparente, simples e escalável de resiliência a falhas para Web services no sistema sob o qual os Web services são desenvolvidos.

6 CONCLUSÕES

Os *Web services* são hoje uma tecnologia reconhecida para o desenvolvimento de sistemas *B2B* e integração de aplicações Web.

A computação *Peer-to-Peer* representa a próxima revolução na era da computação. Este novo método de desenvolvimento de sistemas distribuídos vai alterar dramaticamente a forma como as aplicações comunicam, colaboram e trocam dados pela Internet. Contudo a computação *Peer-to-Peer* está ainda na sua infância e muito trabalho será necessário até à sua adopção por um número significativo de organizações, assim como será necessária a consolidação de padrões e especificações que sejam uniformemente seguidas.

As duas tecnologias exploradas, *Web Services* e *Peer-to-Peer*, podem efectivamente ser integradas e usadas para aumentar a resiliência de sistemas distribuídos.

7 REFERÊNCIAS

W3c SOAP standards, 2001

W3C wsdl sepcification, http://www.w3c.org/TR/wsdl

UDDI standards, http://www.uddi.org

Projecto JXTA, http://www.jxta.org

Wilson, J., "Inside JXTA: *Programming peer-to-peer Using the JXTA Platform*", New Riders, 2002.

JXTA Programmers Guide, http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

JXTA Bench, http://bench.jxta.org/

O'Hearne, B. S., artigo opinião: *Web Services and JXTA: Companion Technologies*, 2001

Mamun Q., Masun S., Mustafa M., *Modified Bully Algorithm for Election Coordination in Distributed Systems*, 2004

Stoller S., *Leader Election in Distributed Systems with Crash Failures*, 1997

Garcia-Molina, H., "Elections in a Distributed Computing System, *IEEE Trans. on Computers*, Vol. C-31, No. 1, Janeiro de 1982, páginas 48-59

H. Yoshinaga, T. Tsuchiya, K. Koyanagi, "*A Coordinator Election Using the Object Model in* Peer-to-Peer *Networks*", Third International Workshop on Agents and *Peer-to-Peer* Computing, Nova Iorque, Julho, 2004

V. Dialani, S. Miles, L. Moreau, D. D. Roure, M. Luck. *Transparent fault tolerance for web services based architectures*. *In* Eighth International Europar Conference (EUROPAR '02), Padeborn, Germany, Agosto de 2002. Springer-Verlag.

Looker N., Munro M.,WS-FTM: *A Fault Tolerance Mechanism for Web Services*. http://www.dur.ac.uk/computer.science/research/technical-reports/2005/A Fault Tolerance Mechanism.pdf.

F. Tartanoglu, V. Issarny, N. Levy, A. Romanovsky. *Dependability in the Web Service Architecture*. In Architecting Dependable Systems, LNCS 2677, páginas 89-108. SpringerVerlag, 2003.

Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. *Peer-to-Peer Computing*. Relatório técnico HPL-2002-57, HP Labs, Março de 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf