# Coordenação de Serviços Web heterogéneos com tolerância a faltas

Filipe Campos, Miguel Matos e José Pereira

HASLab - High-Assurance Software Laboratory
INESC TEC e Universidade do Minho
Campus de Gualtar
4710-057 Braga, Portugal
{fcampos,miguelmatos,jop}@di.uminho.pt

**Resumo** A norma Devices Profile for Web Services (DPWS) permite a descoberta, a configuração e a interoperabilidade de dispositivos heterogéneos ligados em rede com grande disparidade em termos de capacidade de processamento, desde pequenos eletrodomésticos inteligentes ou controladores em máquinas industriais, até servidores em centros de dados. Os mecanismos de notificação de eventos e de configuração automática previstos pelo DPWS são especialmente adequados a cenários Machine-to-Machine (M2M) devido à sua simplicidade e flexibilidade. No entanto, a escalabilidade em cenários com um elevado número de nós, nomeadamente, em grandes infraestruturas, é limitada. A coerência dos dados armazenados e manipulados pelos mecanismos de autodescoberta também não é adequada a aplicações críticas. Neste artigo, abordamos este problema com uma proposta assente na norma DPWS com os conceitos de difusão epidémica e de consenso, mostrando que é possível compor serviços adequados a diferentes aplicações assegurando garantias de tolerância a faltas e maior escalabilidade.

## 1 Introdução

As arquiteturas orientadas a serviços (SOA) são um importante pilar para a computação empresarial e há um crescente interesse na sua utilização em dispositivos localizados nos mais variados ambientes, desde máquinas altamente especializadas para produção industrial até aos eletrodomésticos inteligentes. Isto sucede devido à melhoria na relação custo-benefício para equipar dispositivos com capacidades de processamento e de comunicação consideráveis, mas também devido à flexibilidade na interoperabilidade, combinação e composição de serviços.

Neste contexto, a possibilidade de expor as capacidades de dispositivos como serviços é atraente, mas levanta problemas como a disseminação escalável e tolerante a faltas [**?**], a manutenção da filiação no serviço [**?**] ou a cadência de transmissão face a destinatários de capacidade limitada [**?**].

Existem duas abordagens para a disseminação eficiente e fiável de informação na computação orientada a serviços. A primeira encapsula a lógica de disseminação em middleware, através de tecnologias como Enterprise Service Bus

(ESB), Java Messaging Service (JMS) ou Extensible Messaging and Presence Protocol (XMPP) [**?**,**?**], introduzindo uma dependência de uma determinada pilha de software e, frequentemente, requer servidores centralizados, ainda que tornados redundantes através de replicação. Para além disso, este middleware não está normalmente disponível para a variedade de dispositivos que têm que ser suportados. Finalmente, ao fornecer capacidades de comunicação como uma caixa negra, os padrões de troca de mensagens são bastante limitados. A segunda abordagem consiste na disseminação de informação ao nível do serviço, através de especificações que podem ser combinadas para expor diferentes padrões de troca de mensagens com variados níveis de *QoS*. Um exemplo desta abordagem é a norma WS-Eventing, que apesar de não suportar *brokers*, incorpora de raiz um mecanismo flexível de filtragem de mensagens, favorecendo implementações eficientes e disseminação de um para muitos. É de notar, contudo, que ambas as abordagens enfatizam a comunicação de um para muitos através de uma infraestrutura centralizada. A tolerância a faltas do servidor central assenta na replicação, enquanto a fiabilidade ponto a ponto e a atomicidade depende do facto de os participantes terem memória suficiente e armazenamento persistente para o registo transacional. Resumidamente, este pressuposto de uma infraestrutura pesada e centralizada é partilhada pela maioria das soluções existentes para a disseminação de informação na computação orientada a serviços.

Devices Profile for Web Services (DPWS) define um conjunto de protocolos e especificações que os dispositivos com recursos limitados devem implementar para interagir transparentemente através de Serviços Web. Cada dispositivo fornece a funcionalidade básica e expõe um ou mais serviços que fornecem as funcionalidades específicas do dispositivo. Entre outras, esta norma inclui especificações como WS-Eventing, SOAP-over-UDP, que permite a utilização de UDP como transporte para as mensagens SOAP e de comunicação multi-ponto, e WS-Discovery, WS-MetadataExchange e WS-Policy, que possibilitam a descoberta dinâmica de dispositivos na rede e respetiva caracterização ao nível dos serviços e recursos. Ao focar-se em sistemas de pequena escala, como domótica, a especificação DPWS apresenta limitações de escalabilidade, impedindo o seu uso com um grande número de componentes. Desde logo, a utilização de WS-Eventing impõe um fardo ao publicador de informação, ao obrigá-lo a notificar todos os subscritores e, por não suportar mecanismos de coordenação transacional, pode levar a estados do sistema incorretos com múltiplos intervenientes.

A alternativa seria recorrer a middleware de coordenação desenvolvido para infraestruturas de grande dimensão como o Apache ZooKeeper [**?**], que contudo requer Java SE 1.6, não sendo compatível com o sistema operativo Android ou Java Micro Edition, que são normalmente disponibilizados em plataformas com poucos recursos. Por outro lado, a norma DPWS fornece uma plataforma flexível para a implementação de serviços e disseminação de eventos, mas sem tolerância a faltas. De facto, embora seja possível adaptar protocolos de coordenação existentes para Serviços Web, como WS-Coordination e WS-AtomicTransaction, os modelos de faltas consideram infraestruturas empresariais com muitos recursos.

Portanto, um protocolo de coordenação leve e escalável, que se encaixe nas suposições gerais da norma DPWS, é necessário.

Neste artigo, propomos uma infraestrutura para a coordenação de serviços com tolerância a faltas que se baseia em especificações atuais para Serviços Web bem como em protocolos epidémicos e protocolos de consenso.

Os protocolos epidémicos são uma abordagem escalável e frugal para a disseminação de mensagens. Apesar de se poder utilizar um middleware já existente (ex. NeEM[1]), teríamos muitas das limitações já apontadas à primeira abordagem. Alternativamente, e em linha com a segunda abordagem, fornecemos serviços que podem ser combinados para reproduzir uma variedade de interações epidémicas, permitindo que qualquer arquitetura orientada a serviços utilize este tipo de protocolos para múltiplas aplicações de disseminação de informação.

O serviço de consenso fornecido na infraestrutura apresentada oferece a funcionalidade do serviço de coordenação ZooKeeper, não fornecida pela norma DPWS. Como a notificação de eventos se encontra incorporada em DPWS, esta traduz-se na capacidade de replicar dados de forma tolerante a faltas. A nossa abordagem para este serviço assentou na implementação do protocolo de consenso Raft [**?**] sobre DPWS de forma a fornecer aos serviços replicados a capacidade de tolerar falhas de comunicação bem como de servidores.

O resto do artigo apresenta a seguinte estrutura. A Secção **??** contextualiza e motiva o trabalho. Na Secção **??** é apresentada a arquitetura geral, discutindo-se a disseminação na Secção **??** e a coordenação na Secção **??**. A Secção **??** descreve a avaliação conduzida, a Secção **??** discute o trabalho relacionado e por fim a Secção **??** conclui o artigo.

## 2 Contexto

A coordenação de serviços envolvendo disseminação de informação preocupa-se com dois desafios principais: a) como disseminar informação de forma eficiente para todos os alvos e como manter as listas de alvos e b) que garantias são dadas de que a informação é realmente disseminada na presença de faltas.

No remanescente desta secção, serão descritas as normas para Serviços Web focados nestes desafios, bem como os protocolos epidémicos e de consenso.

Desde os anos 90 que a comunicação confiável tem sido vista como a solução para estes desafios, e assim, várias tecnologias de filas de mensagens transacionais têm sido utilizadas, tais como WebSphereMQ da IBM e MSMQ da Microsoft, para além de tecnologias de publicação/subscrição, como TIBCO Rendezvous. Num esforço para interligar estas diferentes tecnologias, a API Java Message Service (JMS) foi definida, normalizando a utilização de ambas entre aplicações diversas. Algumas destas tecnologias foram adaptadas aos serviços Web, mas, devido à dependência de protocolos proprietários, a interoperabilidade requer *gateways* que façam a tradução entre ambientes diferentes.

[1] http://neem.sf.net

Com a emergência dos Serviços Web como solução de integração preferida para sistemas distribuídos, WS-ReliableMessaging (WS-RM) é a norma atualmente adotada para realizar trocas confiáveis de mensagens em aplicações distribuídas na presença de falhas de software, de máquinas ou de rede. Assegurando a interoperabilidade entre serviços heterogéneos, em termos da confiabilidade da comunicação, o desenvolvimento de serviços baseados nesta norma é simplificado, minimizando o número de erros na lógica de negócio [**?**].

O termo coordenação refere-se por vezes a um tipo de orquestração definido pela norma WS-Coordination, que especifica uma infraestrutura extensível para gestão de contexto para a coordenação de ações em aplicações distribuídas. Esta coordenação é alcançada através de protocolos que estendem esta norma para, por exemplo, alcançar um acordo no resultado de transações distribuídas, como é o caso de WS-AtomicTransaction (WS-AT), para transações atómicas e de curta duração, e de WS-BusinessActivity (WS-BA), para atividades de negócio de longa duração. A norma WS-AT fornece uma adaptação do mecanismo clássico *2PC* para serviços Web, apesar de se considerar que não funciona bem com esta tecnologia [**?**]. No entanto, mostra-se adequado para interações curtas entre serviços que se encontrem próximos e que necessitem de resultados coerentes para uma transação.

Outras normas, como WS-Eventing, WS-AT ou WS-BA, podem utilizar WS-RM para garantir comunicação confiável entre os intervenientes. Apesar de ser capaz de garantir a entrega de mensagens ponto a ponto de forma confiável, a sua utilização é ineficiente e provoca uma sobrecarga no emissor no caso de haver muitos destinatários ou muitos erros de comunicação. Para garantir a entrega atómica para todos os destinos, a norma WS-RM teria que recorrer a WS-AT ou um protocolo de coordenação semelhante, o que aumentaria o consumo de recursos tanto de processamento como de comunicação. Assim sendo, não é capaz de lidar por si própria com falhas para oferecer um serviço replicado, enfatizando a necessidade de um algoritmo capaz e eficiente para tal cenário.

Nas comunicações entre computadores, a comunicação epidémica descreve o processo em que um participante, que pretende disseminar alguma informação, seleciona aleatoriamente um pequeno conjunto de outros participantes e lhes envia essa informação. Cada um destes destinos, repete este procedimento, sendo também frequentemente conhecida por rumor (*gossip*) por se assemelhar à propagação de boatos [**?**]. Um aspeto interessante destes protocolos é que não necessitam de um mecanismo reativo a falhas, nomeadamente, de armazenamento, confirmação ou retransmissão de mensagens que constituem a maior parte da complexidade dos protocolos de comunicação mais comuns. Em vez disso, a confiabilidade é assegurada pela redundância e aleatoriedade inerentes aos protocolos, permitindo lidar com falhas tanto de processos como de comunicação. A probabilidade esperada para a entrega de uma mensagem para cada destino e para todos os destinos pode ser derivada diretamente dos parâmetros $f$, número de alvos selecionados localmente por cada processo, e $r$, número máximo de vezes que uma mensagem deve ser reencaminhada antes de ser descartada. Ajustando $f$ e $r$ consoante as dimensões do sistema e número de faltas esperadas, a entrega

da mensagem a qualquer número médio desejado de destinos, ou até atómica, pode ser garantida [**?**]. A chave para a escalabilidade é que o valor do parâmetro $f$ é na pior das hipóteses logaritmicamente proporcional à dimensão do sistema.

O consenso é uma abstração do problema em que todos os processos de um sistema distribuído devem acordar no mesmo valor, apesar de terem iniciado com opiniões diferentes e independentemente de alguns falharem [**?**].

O Raft [**?**] é um protocolo de consenso que segue a abordagem da máquina de estados replicada. Nele são dissociados elementos chave dos algoritmos de consenso, como a eleição de líder, a replicação do registo persistente e a coerência, enquanto simplifica a concretização através da redução do número de estados possíveis. O Raft suporta modificações à constituição do grupo, mantendo-se em normal funcionamento durante tais transições. Comparativamente com o ZooKeeper, que também utiliza a noção de líder, o Raft é um protocolo mais simples pois requer a implementação de um menor número de operações distintas, e minimiza a funcionalidade das réplicas. Por exemplo, as entradas de registo persistente fluem num único sentido, do líder para as réplicas, enquanto no ZooKeeper, estas entradas fluem em ambos os sentidos.

## 3 Framework

A proposta para ultrapassar os desafios identificados é uma infraestrutura composta pelos serviços WS-Gossip e Raft4WS, cuja arquitetura está ilustrada na Figura **??**, assentando sobre os protocolos contidos na norma DPWS implementada pela Java Multi Edition DPWS Stack (JMEDS) do projeto Web Services for Devices (WS4D).

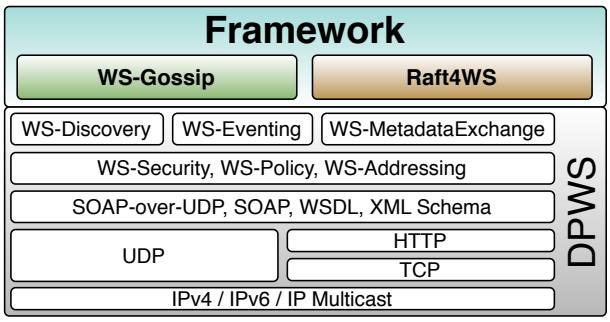


**Figura 1.** Visão global da arquitetura proposta no contexto da norma DPWS.

### 3.1 WS-Gossip

Nesta secção é descrito o serviço WS-Gossip, incluindo as operações disponibilizadas e a forma como se integra com aplicações DPWS existentes.

Quando instalado num dispositivo, o WS-Gossip analisa os serviços nele alojados, criando um serviço sombra para cada um deles. Tanto o serviço original como o seu serviço sombra são anunciados aos clientes que podem utilizar cada um deles independentemente. Um cliente com capacidades WS-Gossip pode examinar as anotações de políticas em ambos os serviços e descobrir que estão relacionados. Sendo possível enviar pedidos diretamente para o serviço original, é assegurada a compatibilidade com clientes que não tenham capacidades WS-Gossip.

Assumindo uma operação *one-way* ou de notificação e *push*, descreveremos o funcionamento básico do WS-Gossip. A disseminação é iniciada quando um cliente envia uma mensagem SOAP para uma operação do serviço sombra. Após a sua receção, a mensagem é inspecionada para determinar se contém um cabeçalho WS-Gossip. Caso não contenha tal cabeçalho, este é construído utilizando os parâmetros predefinidos, como por exemplo, a variante ou o número de alvos ($f$). A disseminação é então iniciada através da inclusão deste cabeçalho na mensagem, que é reencaminhada para os $f$ alvos obtidos do serviço de filiação e para o serviço original. Quando um participante recebe uma mensagem do WS-Gossip, decrementa o contador presente na mensagem enviando-a, de seguida, aos seus alvos para que a disseminação prossiga.

A utilização em operações *one-way* ou de notificação é tratada como foi descrito anteriormente. Em operações de pedido-resposta ou de *call-back*, a mensagem recebida é propagada e posteriormente todas as respostas recebidas são devolvidas até ao iniciador da disseminação. Isto requer que o seu endereço seja incluído no cabeçalho SOAP, nomeadamente no campo **ReplyTo** de WS-Addressing, para além do identificador da mensagem utilizado para a deteção de duplicados. A alternativa é utilizar um filtro que pode omitir ou agregar respostas de acordo com a regra especificada no parâmetro **Filter** do cabeçalho WS-Gossip aquando do início da disseminação.

O serviço sombra oferece as seguintes operações para além das que replica do serviço original:

**Push** Alternativa à invocação direta da operação replicada, permitindo também o envio de um conjunto de mensagens numa única interação.

**PushIds** Informa o alvo dos identificadores das mensagens que o dispositivo invocador possui. O alvo deve depois invocar a operação **Fetch** indicando os identificadores das mensagens que pretende receber.

**Pull** Retorna as mensagens recebidas e armazenadas durante um período de tempo especificado como parâmetro.

**PullIds** Variante da operação anterior, mas que retorna os identificadores em vez das mensagens, que podem ser obtidas através da invocação da operação **Fetch** indicando os seus identificadores.

**Fetch** Devolve as mensagens armazenadas cujos identificadores foram passados como parâmetro.

O WS-Gossip suporta várias variantes gossip que podem ser *lazy* ou *eager*, relativamente à prontidão de envio, e *pull* ou *push*, relativamente à preferência na receção ou envio destas mensagens, sendo realizadas através da composição

das operações descritas anteriormente. Nomeadamente, *lazy push* resulta da invocação de **PushIds**, em vez de **Push** como em *eager push*, aguardando por invocações a **Fetch** para enviar as mensagens identificadas. *Eager pull* é obtida pela invocação periódica da operação **Pull**. A variante *lazy pull* resulta da invocação periódica de **PullIds** para obter identificadores de mensagens, invocando **Fetch** para obter as mensagens pretendidas. A variante selecionada para a disseminação de determinada mensagem depende da configuração do serviço.

### 3.2 Raft4WS

Nesta secção descreve-se o serviço Raft4WS, que possui duas entidades: o Servidor, que aloja uma instância do serviço, e o Cliente, que efetua pedidos aos Servidores.

De acordo com o protocolo Raft, um Servidor pode assumir um de três estados diferentes, *seguidor*, *candidato* ou *líder*, e arranca sempre como seguidor. O serviço Raft4WS inclui 3 operações que todas as suas instâncias fornecem e que têm correspondência com os RPCs definidos no protocolo Raft: **InsertCommand**, que é invocada pelos clientes para inserir novos comandos no registo persistente replicado pelo grupo; **AppendEntries**, invocada pelo líder como *heartbeat*, quando não houver novas entradas no registo, ou para replicar tais entradas nos seguidores; **RequestVote**, que é invocada pelos candidatos para angariar os votos de outros Servidores. O objeto representativo de cada estado inclui as tarefas associadas e implementa uma interface comum para tratamento de disparos do temporizador, caso o servidor não seja contactado, e de invocações às operações do serviço.

A principal tarefa do Cliente é a deteção de dispositivos Raft, através da escuta de mensagens multicast WS-Discovery, ou através da procura ativa por tais dispositivos. O primeiro dispositivo Raft detetado será considerado como líder pelo Cliente, tornando-se o alvo das suas invocações. O Cliente invoca a operação **InsertCommand** com o comando e os parâmetros, bem como um identificador único como parâmetros. Se o Servidor contactado for o líder atual, a resposta conterá o resultado da criação da entrada no registo correspondente ao pedido do Cliente e da sua aplicação na máquina de estados replicada. Caso contrário, o Servidor responde ao Cliente indicando o insucesso do pedido e o endereço do líder atual. Neste caso, o Cliente pode utilizar este endereço para reenviar o pedido ao líder, para que este seja processado, bem como os seguintes.

## 4 Avaliação de Desempenho

### 4.1 Raft4WS

Avaliámos a nossa implementação do protocolo Raft, Raft4WS, comparando-a com o Apache ZooKeeper 3.4.5 em cenários com 1, 3 e 5 servidores, para responder aos pedidos de 1 a 25 clientes. Cada cenário foi testado 5 vezes num grupo de 6 máquinas, correspondendo os resultados apresentados à sua média.