Universidade do Minho
Escola de Engenharia

Paulo Bruno Lima de Abreu

Coordenação de RESTful Web Services

Outubro de 2010

Universidade do Minho
Escola de Engenharia

Paulo Bruno Lima de Abreu

Coordenação de RESTful Web Services

Mestrado em Informática

Trabalho realizado sob a orientação do
Prof. Doutor José Orlando Pereira

Outubro de 2010

Abstract

The use of Web Services has been playing an important role in both enterprise application integration and service availability in the internet. In particular, the services planned and implemented in accordance with the *REpresentational State Transfer* (REST) paradigm have an increased adoption in many environments due to its simplicity, efficiency and reliability.

However, unlike what happens with Web Services and SOAP-based standardized by W3C, coordination of REST services lacks structured solutions, reducing its applicability in more complex scenarios.

This project has as objective the study, implementation and evaluation of the coordination mechanisms for RESTful Web Services, with particular emphasis on the use of epidemic protocols or rumors dissemination, that offer interesting features of scalability and reliability.

This dissertation presents results that allow us to conclude that is possible to obtain significant gains in the use of epidemic protocols in distributed systems, when compared to centralized models, reaching higher response performances with gains between 44% and 51% for the dissemination of information in all network servers.

Keywords: Web Services, RESTFul, Gossip Dissemination, Distributed System.

Resumo

A utilização de *Web Services* tem vindo a desempenhar um papel importante tanto na integração de aplicações empresariais como na disponibilização de serviços na Internet. Em particular, os serviços projectados e concretizados de acordo com o paradigma *REpresentational State Transfer* (REST) têm uma adopção crescente nos mais diversos ambientes devido à sua simplicidade, eficiência e fiabilidade.

No entanto, ao contrário do que acontece com os *Web Services* baseados em SOAP e normalizados pela W3C, a coordenação de serviços REST carece de soluções estruturadas para a sua coordenação, reduzindo assim a sua aplicabilidade em aplicações mais complexas.

Este projecto tem pois como objectivo o estudo, concretização e avaliação de mecanismos de coordenação de *RESTful Web Services*, com particular ênfase na utilização de protocolos epidémicos ou de disseminação de rumores, que oferecem características interessantes de escalabilidade e fiabilidade.

Esta dissertação apresenta resultados que permitem concluir que, na utilização de protocolos epidémicos em sistemas distribuídos, é possível obter ganhos significativos quando comparados com modelos centralizados, chegando a atingir performances de resposta muito elevadas com ganhos entre 44% e 51% para a disseminação de informação em todos os servidores da rede.

Palavras Chave: Web Services, RESTFul, Gossip Dissemination, Distributed System.

Agradecimentos

Ao Prof. Doutor José Orlando Pereira, pela sua disponibilidade, pela generosidade com que sempre transmitiu o seu vasto conhecimento tendo contribuído para a minha formação e para a concretização deste projecto.

A todos os que me têm acompanhado o meu percurso pessoal e profissional, apoiando a construção do meu projecto pessoal.

O meu sincero agradecimento.

Conteúdo

Figuras

Acrónimos

REST - Representational State Transfer

XML - Extensible Markup Language

WSDL - Web Services Description Language

UDDI - Universal Description Discovery and Integration

SOAP - Simple Object Access Protocol

HTTP - Hypertext Transfer Protocol

DTD - Document Type Definition

URI - Uniform Resource Identifier

WS - Web Services

SGML - Standard Generalized Markup Language

JIT - Just-in-time compilation

Capítulo 1

Introdução

A tecnologia de Web Services [ACKM03], normalizada pelo W3C, tem como objectivo a integração de diferentes aplicações e sistemas. Em particular, permite ultrapassar as dificuldades de comunicação através de uma troca de dados num formato universal, o XML, um protocolo para troca de mensagens, o SOAP e um conjunto de normas para a coordenação de serviços em sistemas compostos. Esta abordagem tem, no entanto, sido criticada pela sua elevada complexidade, baixo desempenho e incompatibilidade com a utilização tradicional do protocolo HTTP.

Neste contexto, REpresentational State Transfer (REST) [RR07] é uma técnica de desenvolvimento de Web Services que permite aproveitar directamente a arquitectura HTTP evitando abstracções adicionais tais como o protocolo SOAP. Esta técnica assenta em quatro princípios base: Encapsulamento do estado da aplicação como recursos discretos; Endereçamento uniforme dos recursos; Utilização da interface do HTTP e de formatos padrão; E a compatibilidade com a estrutura de *caching* do HTTP. No entanto, ao contrário dos Web Services tradicionais, não existem mecanismos genéricos de coordenação. Torna-se assim interessante estudar a possibilidade de construir mecanismos de coordenação baseados em protocolos epidémicos.

Os protocolos epidémicos ou baseados em rumores [EGKM04][KS07][KSSV00] são protocolos fundamentais para os enormes sistemas actuais, disponibilizando simultaneamente escalabilidade e fiabilidade, sendo no entanto muito simples de perceber e implementar. Por exemplo, estes protocolos são largamente utilizados na implementação da infra-estrutura dos Amazon Web Services.

1.1 Problema

Os Web Services projectados e concretizados de acordo com paradigma REST, têm tido uma adesão crescente nos mais diversos ambientes devido à sua simplicidade, eficiência e fiabilidade. No entanto, ao contrário dos Web Services tradicionais, não existem mecanismos genéricos de coordenação, reduzindo assim a sua aplicabilidade em soluções mais complexas.

1.2 Objectivo

O objectivo desta dissertação baseia-se em estudar a viabilidade de uma coordenação com Gossip em Web Services, através da implementação e análise de performance de diferentes técnicas. Tendo em consideração a coordenação de infra-estruturas, tecnologias de comunicação (e.g. HTTP) e a paradigmas (e.g. REST). Resumindo esta dissertação tem como objectivos principais:

- Propor um mecanismo de coordenação de RESTful Web Services em Gossip.
- Avaliar este mecanismo através de uma aplicação protótipo e *benchmarks*.

1.3 Contribuições

Com esta dissertação apresenta-se uma arquitectura e um protótipo de coordenação de Web Services baseado no paradigma REST, com o uso do protocolo Gossip para uma disseminação de informação. Apresenta-se também uma avaliação comportamental através de *benchmarks*.

1.4 Estrutura do documento

A presente dissertação encontra-se organizada em 5 capítulos. No primeiro capítulo é feita uma pequena introdução apresentando o problema em questão e seus objectivos. Ainda neste capítulo apresentam-se quais as contribuições que este trabalho traz, acabando com uma estrutura do documento.

No segundo capítulo, é feita uma revisão teórica. Apresenta-se o estado de arte, com as diversas tecnologias e protocolos existentes actualmente.

No terceiro capítulo, apresenta-se a implementação seguida neste trabalho. Aqui encontra-se a sua arquitectura, passando também pelas escolhas que determinaram a sua implementação.

No quarto capítulo, apresenta-se os resultados obtidos no final da implementação, assim como a sua discussão.

No quinto capítulo, são apresentadas as conclusões, limitações do trabalho e sugestões futuras.

Capítulo 2

Estado da Arte

Este capítulo apresenta o estado da arte, focando pontos essenciais da coordenação de Web Services, fundamentais para a realização desta tese. Sendo o tema a coordenação de RESTful Web Services, é apresentada numa primeira fase, uma descrição geral de Web Services mostrando o porquê da sua existência e a sua evolução. Ainda nesta secção descreve-se uma arquitectura de Web Services e algumas tecnologias para a sua implementação, onde se insere o REST e Atom. Estes, são utilizados para a implementação da coordenação dos serviços. Numa segunda fase, aborda-se mais especificamente o tema sobre a coordenação de Web Services, apresentando protocolos que têm sido utilizados na sua concretização. Por último, apresenta-se o Gossip como forma de coordenação e disseminação de informação, que através de diferentes metodologias permite propagar informação em ambientes distribuídos.

2.1 Web Services

Com a evolução das tecnologias ao longo do tempo, diversas entidades espalhadas pelo mundo inteiro depararam-se com uma grande necessidade de trocar e partilhar informação. Para colmatar esta necessidade surgiram algumas alternativas que, tendo sido desenvolvidas, não resultaram em soluções práticas, pois as entidades necessitavam de trocar informação de forma simples e flexível contemplando conhecimento da linguagem em ambas as partes para garantir uma comunicação correcta. Surgiram então os Web Services que revolucionaram a forma de comunicação entre qualquer entidade.

Os Web Services vieram representar uma forma de comunicação onde cada entidade tem a possibilidade de disponibilizar os seus serviços internos para entidades externas através de uma rede de comunicações. Embora se relacione em grande parte com a exposição de serviços para o exterior, estes são também muito utilizados de forma interna ou em redes internas como comunicação de componentes, abstraindo alguns serviços tanto por questões de segurança como de eficiência ou organizacional. A troca de informação é estabelecida através de mensagens cujo formato é conhecido pelas entidades permitindo falarem a mesma linguagem. Estas mensagens designaram-se de SOAP Messages, um

protocolo que é descrito como Simple Object Access Protocol (SOAP) onde cada mensagem obedece a uma estrutura de XML.

Os serviços disponibilizados através de Web Services podem fornecer as mais diversas informações através das mensagens desde que obedeçam às regras estabelecidas. De tal forma que foi criada uma especificação que define o formato e a informação disponível pelos serviços através do WSDL (Web Service Description Language). Esta especificação pode ser consultada pelos clientes a qualquer altura, no sentido de adquirir todo o conhecimento disponibilizado pelo serviço, especificando os interfaces existentes, tipo de informação, operações suportadas, formato dos pedidos e das respostas.

Estabelecidos os protocolos de comunicação, foi criado um serviço de registo de Web Services denominado de UDDI (Universal Description Discovery and Integration), onde cada entidade publica e regista os seus serviços ficando estes disponíveis para os consumidores, podendo-se consultar ou pesquisar à semelhança um directório de serviços.

Porém esta abordagem de Web Services, através de SOAP, tem vindo a ser muito criticada, pois tem um grau de complexidade elevado, baixo desempenho e incompatibilidade com a utilização tradicional do protocolo HTTP. Até há bem pouco tempo apenas se ouvia falar nos Web Services SOAP, mas recentemente surgiu uma nova arquitectura, denominada REST, que veio revolucionar a forma como os Web Services são implementados, permitindo utilizar alternativas à estrutura XML no SOAP e utilizando de forma mais eficiente o protocolo HTTP.

2.1.1 Arquitectura dos Web Services

A arquitectura dos Web Services é definida segundo um conjunto de protocolos normalizados, tais como XML, SOAP, WSDL, UUID, HTTP, entre outros, que permitem interoperabilidade e extensibilidade de aplicações, plataformas e *frameworks* através da Web. Esta arquitectura define uma forma de troca de mensagens, descrição de serviços, publicação e pesquisa de serviços [W3C04]. A arquitectura básica de Web Services define um modelo de interacção de três perfis principais: *service provider*, *service discovery agency* e *service requestor* que se podem representar segundo a Figura 2.1.

Nesta, são também apresentadas as relações entre os diferentes actores, que comunicam através de XML segundo uma especificação SOAP. O *service provider* é detentor de um serviço disponibilizado por um componente de software, onde este define uma descrição do Web Service através de WSDL que publica no *service requestor* ou no *service discovery agency*. O *service requester* realiza uma pesquisa localmente ou no *service discovery agency* de forma a obter a descrição do serviço para que possa gerar uma mensagem SOAP ao destinatário.

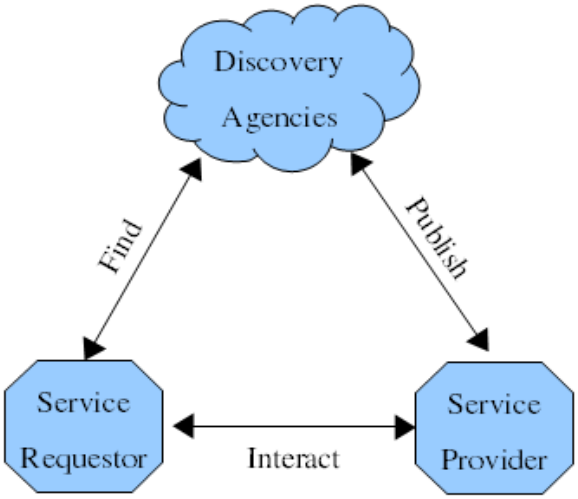

Figura 2.1: Service Oriented Architecture
(adaptado de Web Services Architecture [W3C04])

2.1.2 XML

Extensible Markup Language (XML) [W3C10] é uma linguagem que derivou do SGML onde define uma representação de dados estruturada de forma universal e padronizada, partilhando informação de forma mais simples na Internet. Entre os principais objectivos da criação desta linguagem encontra-se a simplicidade e legibilidade, tanto para pessoas como para computadores, de forma a criar aplicações que podem processar facilmente este tipo de documentos. É também possível utilizar técnicas de validação através de DTD ou XML Schema, sendo estes definições dos tipos de documentos onde é indicada qual a estrutura que o documento XML poderá assumir, assim como todo o tipo de dados suportados. Segue-se um exemplo que representa o conteúdo de um ficheiro XML que é composto por um conjunto de dados, onde existe uma listagem com duas entidades, os respectivos nomes e valores associados.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <entidades>
3    <entidade>
4      <nome>Entidade A</nome>
5      <valor>1000</valor>
6    </entidade>
7    <entidade>
8      <nome>Entidade B</nome>
9      <valor>500</valor>
10   </entidade>
11 </entidades>
```

2.1.3 SOAP

Nos Web Services tradicionais é utilizado um protocolo denominado de SOAP (Simple Object Access Protocol) [W3C07], que visa estabelecer uma comunicação estruturada (através de XML) entre as entidades que desejam trocar informação. Este protocolo permite uma disponibilização de serviços estruturados, sob a forma de WSDL (Web Services Description Language), tendo como vantagens uma fácil comunicação através de HTTP e Firewall, assim como a sua versatilidade e independência de plataformas e linguagens. Embora existam algumas limitações como a performance em grandes dimensões de dados na troca de mensagens, quando comparado com outros protocolos.

O SOAP troca informação, através de mensagens (SOAP Messages), sobre uma estrutura de envelopes onde na origem é construído um envelope devidamente estruturado e no destino é extraída a informação. Os envelopes são constituídos em duas partes principais designadas por *header* e o *body,* onde apenas o *body* é obrigatório. Ambas as partes são constituídas por um ou mais blocos que assumem uma estrutura definida pelo WSDL (ver Figura 2.2).

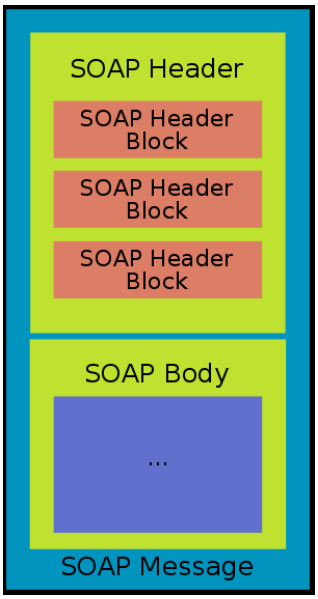

Figura 2.2: Estrutura de mensagens SOAP
(adaptado de Foundations And Future Directions Of Web Services [Haa05])

O SOAP assume a existência de uma entidade que envia o envelope e outra que recebe. Entre este processo podem existir intermediários que reenviam o envelope, onde podem processar o *header* caso exista. As mensagens SOAP apresentadas na Figura 2.2, são estruturadas através de documentos XML que contêm a informação necessária para as entidades que comunicam. No SOAP body é possível adicionar todo o tipo de informação que deverá estar descrita no XML Schema de forma a validar o conteúdo. Segue-se um exemplo (adaptado de Using WSDL in SOAP applications [Ogb00]), que representa um envelope SOAP tal como descrito anteriormente.

```
POST /EndorsementSearch HTTP/1.1
Host: www.snowboard-info.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 261
SOAPAction: "http://www.snowboard-info.com/EndorsementSearch"
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Body>
   <m:GetEndorsingBoarder xmlns:m="http://namespaces.snowboard-info.com">
     <manufacturer>K2</manufacturer>
     <model>Fatbob</model>
   </m:GetEndorsingBoarder>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.1.4 WSDL

Web Services Description Language (WSDL) [W3C01], define uma especificação baseada em documentos XML que descrevem os Web Services. Isto permite a troca de mensagens SOAP entre os prestadores de serviços e o cliente de forma estruturada. Assim é possível definir uma estrutura obrigatória de comunicação dos serviços fornecidos aos quais o cliente deve obedecer. Esta especificação é normalmente disponibilizada num directório de serviços para que os clientes possam conhecer quais os serviços disponibilizados por determinada entidade. De seguida apresenta-se um exemplo (adaptado de Web Services Essentials [Cer02]) que representa um WSDL no formato XML, onde estão declarados todos os serviços suportados e de que forma podem ser utilizados, indicando as operações, tipos de pedidos e respostas.

```
<?xml version="1.0" encoding="UTF-8"?>
        <definitions name="HelloService"
      targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

            <message name="SayHelloRequest">
               <part name="firstName" type="xsd:string"/>
            </message>
            <message name="SayHelloResponse">
               <part name="greeting" type="xsd:string"/>
            </message>

            <portType name="Hello_PortType">
               <operation name="sayHello">
                  <input message="tns:SayHelloRequest"/>
                  <output message="tns:SayHelloResponse"/>
```

```
20              </operation>
21          </portType>
22
23          <binding name="Hello_Binding" type="tns:Hello_PortType">
24              <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/
                    http"/>
25              <operation name="sayHello">
26                  <soap:operation soapAction="sayHello"/>
27                  <input>
28                      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                            encoding/"
29                          namespace="urn:examples:helloservice"
30                          use="encoded"/>
31                  </input>
32                  <output>
33                      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                            encoding/"
34                          namespace="urn:examples:helloservice"
35                          use="encoded"/>
36                  </output>
37              </operation>
38          </binding>
39
40          <service name="Hello_Service">
41              <documentation>WSDL File for HelloService</documentation>
42              <port binding="tns:Hello_Binding" name="Hello_Port">
43                  <soap:address location="http://localhost:8080/soap/servlet/
                        rpcrouter"/>
44              </port>
45          </service>
46      </definitions>
```

2.1.5 UDDI

Universal Description Discovery and Integration (UDDI) , é uma especificação criada para fornecer um serviço que permite a pesquisa de Web Services, comportando-se na realidade como um directório de serviços que permite a sua publicação com a descrição dos Web Services. Este serviço possibilita a pesquisa de forma simples e centralizada descrevendo a forma como os sistemas deverão interagir.

O registo de serviços no directórios divide-se em três componentes: White Pages, Yellow Pages e Green Pages.

White Pages – Armazena a informação sobre as entidade de negócio que disponibilizam os serviços, tal como, nome, morada ou outro tipo de informação relevante que se associe à descrição da entidade.

Yellow Pages – Neste componente encontram-se as possíveis classificações que se pode atribuir às entidades que prestam os serviços, dividindo-as em diferentes categorias de serviços ou negócio.

Green Pages – Todo o detalhe mais técnico encontra-se neste componente, onde se disponibiliza informação sobre os Web Services, como utilizar, de que forma e onde estão disponíveis, assim como os interfaces tal como os WSDL que especificam a forma de utilizar determinado serviço.

2.1.6 REST

REpresentational State Transfer (REST) é uma forma de arquitectura de software para sistemas distribuídos. O termo REST teve origem na tese de doutoramento de Roy Fielding (Architectural Styles and the Design of Network-based Software Architectures)[Fie00] e refere-se a um design de arquitectura para aplicações Web, com alguns princípios básicos:

- Atribuir um identificador a tudo. Tudo deverá ter um ID, URI, ou uma referência (http://di.uminho.pt/eventos/1234);
- Utilizar métodos standards do protocolo HTTP: GET, PUT, POST, DELETE;
- Conteúdos com múltiplas representações (Accept: text/x-vcard; Accept: application/xhtml+xml; ...);
- Comunicações Stateless. Aplicações não mantêm sessões/ligações ao servidor;

Actualmente, as pesquisas Web representam a utilização do protocolo REST. Por exemplo quando estamos a utilizar um endereço Web (http://di.uminho.pt/eventos/ 12345) estamos a fazer um pedido de um recurso disponibilizado pelo servidor, assim como toda a navegação de conteúdos.

O REST trata a Web como recursos disponíveis através do URL, cada endereço representa um recurso que pode ser tratado de diferentes formas. A sintaxe utilizada no URL pode ser facilmente interpretada, por exemplo `http://server/book/12345` significa que são devolvidos os detalhes de um determinado livro com o código 12345.

Recentemente foi adicionado o suporte de REST à especificação de WSDL 2.0, onde permite especificar os serviços como já vinha sendo conhecido com o SOAP [CMRW07]. O protocolo utilizado no REST é o HTTP, ao contrário do SOAP que pode ser utilizado em vários protocolos.

Os métodos utilizados no protocolo HTTP são mapeados para o REST da seguinte forma:

- POST – criar um recurso
- GET – devolver um recurso
- PUT – actualizar um recurso
- DELETE – remover um recurso

A Figura 2.3 representa uma tradução de serviços numa arquitectura não REST, numa arquitectura REST. Como se pode verificar, existem vários métodos para representar as várias acções no mesmo conteúdo: getOrder(), submitOrder(), updateOrder(), cancelOrder que são traduzidas para REST como um único URI /orders permitindo todas as operações do anterior através dos métodos REST já indicados.

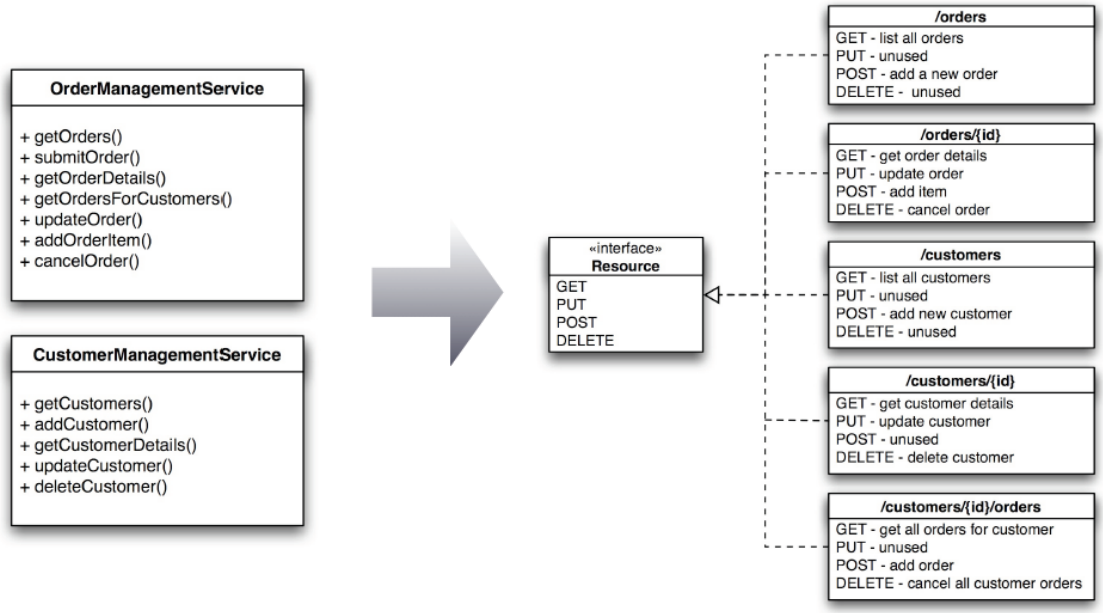

Figura 2.3: Tradução de serviços em arquitectura REST
(adaptado de A Brief Introduction to REST [Til07])

Devido à sua simplicidade, o protocolo REST poderá permitir uma redução do volume de informação das comunicações quando comparado com o protocolo SOAP. Isto é, os RESTful Web Services não necessitam de grandes estruturas em XML para fazer a troca de mensagens.

2.1.7 Atom

Atom é um protocolo ao nível da aplicação que permite publicar e editar conteúdos sob a forma de XML, como por exemplo, Blogues, Wikis, etc. Este é definido através de Atom Publishing Protocol, no qual representa métodos básicos como GET, POST, PUT e DELETE.

- GET - tem como funcionalidade devolver conteúdos ao cliente;
- POST - permite a criação de novos conteúdos;
- PUT - actualiza conteúdos existentes;
- DELETE - remove conteúdos;

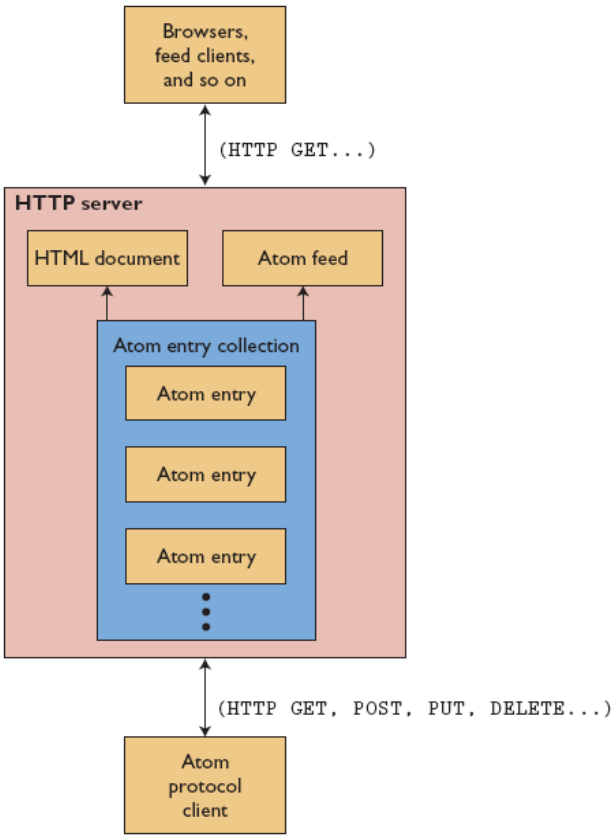

Figura 2.4: Arquitectura Atom
(adaptado de Atom The Standard in Syndication [HSAA05])

A Figura 2.4, representa uma arquitectura Atom num servidor HTTP. O Atom contém várias entradas ("Atom entry") que representam os vários conteúdos possíveis da aplicação. O protocolo Atom Publishing Protocol, permite uma gestão de conteúdos através

dos diferentes métodos já especificados, enquanto os Browsers, Feed clients consomem os conteúdos através do método GET.

2.2 Coordenação de Web Services

Com a evolução das tecnologias surgiram sistemas complexos e de grande escala, que levaram a criar serviços distribuídos pelas mais diversas necessidades. Para manter estes sistemas, foram utilizados Web Services para facilitar a sua comunicação, assim como para criar um standard de comunicações.

Surge então a necessidade de coordenar estes serviços de forma estável e coerente. Esta coordenação não é de todo fácil pois existem inúmeras variáveis na sincronização de dados e sua distribuição. Temos de ter em atenção que a informação deverá ser distribuída rapidamente para garantir informação coerente na rede, mas de forma consistente para evitar falhas ou informação redundante. De tal forma surgiram alguns protocolos tais como WS-Coordination, WS-ReliableMessaging e WS-AtomicTransaction utilizados em Web Services baseados em SOAP, pois em REST não existem especificações de protocolos de coordenação.

2.2.1 WS-Coordination

O WS-Coordination [WWC09] descreve uma especificação de protocolos que permite a coordenação de serviços entre aplicações distribuídas, esta, por si só, não define um modelo completo para a coordenação de Web Services sendo necessário conjugar-se com outros protocolos definidos, utilizando os modelos de SOAP e WSDL. Esta especificação define um modelo que se caracteriza pela existência de uma serviço de coordenação (coordenador), que é responsável pela criação de contextos de coordenação e pelo registo de aplicações.

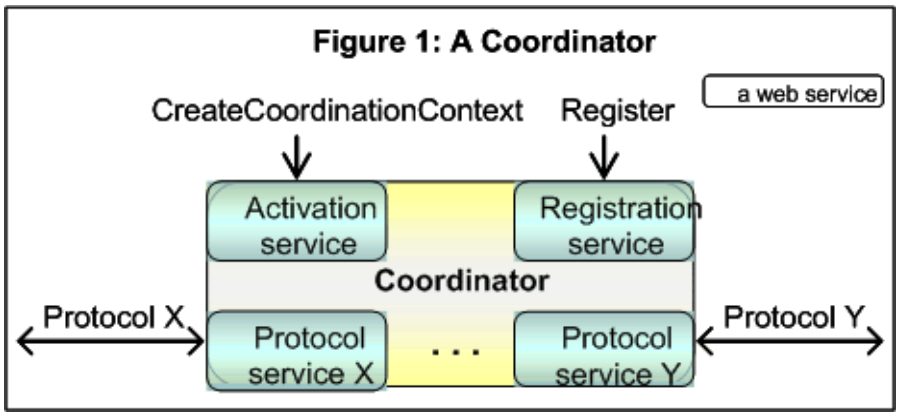

Figura 2.5: Modelo WS-Coodination
(adaptado de Web Services Coordination [WWC09])

A Figura 2.5 representa o modelo de coordenação de um Web Service, onde existe um

serviço de activação que permite criar um contexto de coordenação e um serviço de registo que permite a uma aplicação registar protocolos de coordenação.

2.2.2 WS-ReliableMessaging

Quando falamos de sistemas distribuídos está implícita a existência de comunicação entre diversos intervenientes, onde consideramos que para tal é necessário haver troca de mensagens. Contudo esta comunicação pode ser interrompida por diversas falhas de sistemas ou redes implicando a perca de mensagens, duplicação ou mesmo a ordem de entrega, podendo ter implicações graves mediante o contexto. Como tal, foi necessário garantir que a troca de mensagens era realizada de forma confiável entre os sistemas, onde o comunicador que inicia o envio de uma mensagem tenha garantias de que essa mesma foi entregue no destino, ou a confirmação de que houve um problema e não foi possível fazer a entrega.

Para que fosse possível dar garantias de fiabilidade, foi definido o WS-ReliableMessaging (Web Services Reliable Messaging), que descreve um protocolo para ambientes distribuídos onde garante fiabilidade na entrega de mensagens entre aplicações em caso de falhas [FL05]. O modelo de Reliable Messaging apresenta quarto tipos de entrega: AtMostOnce, AtLeastOnce, ExaclyOnce e InOrder.

AtMostOnce – a mensagem será entregue ao destinatário no máximo uma vez sem duplicações, caso contrário será gerado um erro.

AtLeastOnce – as mensagens serão entregues ao destinatário pelo menos uma vez, caso contrário será gerado um erro. É possível existir duplicação de mensagens visto poderem ser entregues mais do que uma vez.

ExaclyOnce – a mensagem será entregue apenas uma vez não existindo duplicação de mensagens, caso contrário será gerado um erro.

InOrder – as mensagens serão entregues pela ordem que são enviadas para garantir a sua entrega, pode-se combinar com uma das três regras já apresentadas.

Esta especificação tem como principal objectivo definir um modelo de entrega de mensagens com garantia de entrega, mas este não define por si só um modelo comunicação de mensagens completo, sendo necessário utilizar outras especificações de Web Services. O WS-ReliableMessaging utiliza extensibilidade dos modelos de SOAP e WSDL, mas este mecanismo é extensível podendo-se adicionar novas funcionalidades.

2.2.3 WS-AtomicTransaction

Na coordenação de Web Services podem existir inúmeras aplicações ou serviços distribuídos, onde muitas vezes é necessário garantir que a informação transmitida chega ao destino

ou é processada correctamente pelo destinatário. Nestas situações, por vezes, algumas das mensagens não chegam a ter sucesso na entrega ou no processamento resultando na corrupção do modelo distribuído, pois alguns dos destinatários perdem informação e não adquirem o mesmo conhecimento. Para evitar este tipo de problemas, é necessário garantir que na existência de uma falha a operação é cancelada e todos os destinatários são informados do problema. Isto foi o que a especificação WS-AtomicTransaction [LW09] definiu, um protocolo de transacções atómicas que em conjunto com a especificação WS-Coordination permitem criar modelos de coordenação com atomicidade nas transacções.

2.3 Gossip Dissemination

Actualmente, com o aparecimento de grandes redes e serviços em escala mundial a necessidade de distribuir informação é cada vez maior. Viu-se então uma necessidade de definir protocolos para propagar informação em sistemas distribuídos, surgindo então o Gossip Dissemination.

A disseminação de informação em Gossip é realizada por algoritmos epidémicos de informação, que se comportam tal como um vírus, contagiando todos os indivíduos ao seu alcance. Uma vez iniciado o contágio é difícil de o parar, pois são apenas necessárias poucas pessoas para iniciar o processo. Mesmo que algumas desapareçam antes de contagiar, o processo não para de contagiar o resto da população [EGKM04].

A primeira utilização de Gossip em sistemas distribuídos aconteceu à mais de vinte anos, onde foi utilizado para garantir a consistência de base de dados replicadas. Desde então foi utilizado para resolver diversos problemas [KS07][HSAA03].

O Gossip Dissemination fornece vários métodos de disseminação de informação através da troca de dados entre os diversos nós da rede, segundo alguns critérios desejados:

- Não congestionar a rede;
- A informação deverá ser consistente;
- Reduzir o tempo de disseminação da informação;
- Ser tolerante a falhas;

A forma como se realiza a troca de informação, baseia-se em algumas definições [KSSV00] de envio, recolha e envio/recolha:

- **Push** - para o envio de informação para outro nó na rede;
- **Pull** - recepção de informação de outro nó;
- **Push/Pull** - envio e recepção de informação entre nós.

Na implementação de um processo de disseminação é necessário ter em conta alguns aspectos tais como:

Gestão de grupos - É necessário garantir que todos os intervenientes pertencem a grupos coordenados, de forma a garantir sincronismo e uma boa capacidade de comunicação.

Estabilidade da rede - Os membros pertencentes à rede, devem estar organizados de forma a garantir que não existe uma degradação da rede ou dos serviços disponibilizados.

Gestão de buffer de mensagens - Cada servidor presente deve ser capaz de fazer uma boa gestão de armazenamento de mensagens, garantindo minimamente a recepção de dados limitado a um *buffer*.

Filtrar mensagens - Todas as mensagens devem ser filtradas caso exista duplicação de dados na rede, de forma a garantir a coerência dos dados recebidos.

2.3.1 Métodos

Em Gossip Dissemination, são vários os métodos que podem ser definidos, para a disseminação de informação. Seguem-se alguns dos mais importantes métodos utilizados.

Flat Gossip - Representa o método mais simples, enviando informação para todos os nós que conhece. Isto é realizado de forma periódica e aleatória, não existindo controlo para quais os nós que serão enviados [PGC06][KMG03].

Hierarchical Gossip - Para sistemas de grande escala criam-se hierarquias quando o custo entre os nós mais distantes é elevado, existindo assim um tipo de "nuvem", onde se agrupam vários nós em que algum ou alguns serão seleccionados para comunicar com outra nuvem. Existe então o conceito de hierarquia, pois a informação é propagada em primeira mão para os nós locais e só depois serão enviados para outra hierarquia [KMG03][GKG02].

Directional Gossip - Através do conhecimento da topologia da rede, este método escolhe os vizinhos com menos ligações para aumentar a qualidade de mensagens e reduzir a sua quantidade. Isto é, se enviarmos informação para um nó com poucas ligações, o congestionamento na rede de mensagens duplicadas será menor. Com este método garante-se uma redução do congestionamento na rede, embora reduza a velocidade de propagação [LEH03][LM].

Limite de saltos - Este método resume-se em propagar a informação até um limite de saltos, isto é, determinada informação será propagada até um limite de nós. Sendo assim se enviarmos determinada informação com um limite de 2 nós, esta deverá ser transmitida até ao segundo nó deixando este último de a propagar. Isto garante uma redução do congestionamento da rede [LM].

Utilidade - O método de "utilidade", é definido através de uma garantia de utilidade de determinado nó. Existe uma avaliação ou classificação de cada nó, permitindo que estes de maior utilidade façam a propagação de informação. Esta "utilidade" pode ser definida de várias formas mediante o contexto, por exemplo, localização, qualidade de rede, número de ligações [LM].

Conhecimento da topologia total - Com o conhecimento da topologia total da rede, é possível criar caminhos para a propagação de informação de forma a reduzir o número de comunicações. Isto permite evitar situações em que se envia informação para um nó quando este é atingível por outro conhecido. Assim é possível evitar o congestionamento da rede e garantir uma rápida distribuição da informação. Embora existam grandes vantagens, há um custo grande inerente, pois é necessário armazenar a informação da topologia nos nós assim como eleger os melhores [KMG03].

2.3.2 Gossip na prática

Olhando para casos reais, o tempo de propagação e a consistência de informação é muito importante. Numa situação em que é necessária informação muito frequente e actualizada, como por exemplo, o estado de um sensor (temperatura, ...), o envio de mensagens com novos valores do sensor será muito frequente o que, numa situação de grandes atrasos, teria como consequência a entrega de mensagens desactualizadas e um congestionamento da rede, visto não ter capacidade de propagação para determinado volume de informação. A utilização de Gossip na disseminação de informação, deve garantir que este tipo de problemas não ocorre, devendo-se escolher o método mais adequado para o caso em questão.

2.3.3 Número de servidores a vigiar

A utilização de Gossip, obriga à existência de um conjunto de servidores que partilham informação, como tal é necessário que estes comuniquem entre si. Uma possibilidade seria todos os servidores terem o conhecimento de todos os outros e enviar a informação quando necessário, mas isto resultaria no congestionamento da rede. Com o Gossip evita-se o congestionamento da rede definindo conjuntos de servidores que se conhecem e propagam a informação apenas para um conjunto restrito conhecido. O cálculo do número de servidores que cada um na rede deverá vigiar é de log(N), onde N é o número total de servidores

presentes na rede de forma a garantir uma rede estável e de boa capacidade de resposta [EGKM04][KSSV00].

2.3.4 Probabilidade de entrega

Numa rede de servidores onde são utilizados métodos de Gossip, é necessário considerar que existe uma probabilidade de entrega de mensagens relacionado com o número de servidores em causa, considerando como já referido que cada servidor necessita de propagar as mensagens para log(N) servidores, onde cada mensagem chega a todos os servidores com uma probabilidade muito elevada de log(N) iterações. A falha de comunicação de um ou vários servidores não afectará significativamente a propagação da mensagem pelos servidores restantes , neste processo o utilizador do sistema Gossip deverá estar confortável com esta probabilidade de entrega [EGKM04][KSSV00].

2.3.5 Membership

As redes baseadas em Gossip, são organizadas por conjuntos de servidores que trocam informação entre si segundo uma estrutura definida, de forma a reduzir o congestionamento da rede. Para tal, cada servidor tem conhecimento de determinados nós vizinhos para os quais deve propagar a informação, mas isto requer uma organização de rede que pode ser resolvida através de *membership*, isto é, uma rede sobreposta. O Gossip Membership pode ser definido de duas formas distintas, distribuído e centralizado.

Membership distribuído - Os modelos de *membership* distribuídos, são conseguidos através de um conjunto de servidores que disponibilizam um serviço de *membership*. Estes têm de garantir em conjunto a consistência do serviço mantendo a rede actualizada e disponibilizando a mesma informação entes eles. Estes modelos podem ser especificados de diversas formas, das quais se pode implementar recorrendo a um modelo de Gossip para a disseminação de informação.

Membership centralizado - O modelo de *membership* centralizado, é constituído apenas por um único ponto capaz de fornecer todas as funcionalidades, permitindo que todos os servidores entrem em contacto consigo para fornecer as regras de *membership*. O modelo centralizado é uma boa opção, existindo boas referências tal como por exemplo o Skype que utiliza este mecanismo para a autenticação dos utilizadores [S.A06].

Na comparação dos modelos de *membership*, é necessário ter em conta o custo do modelo distribuído, pois quando comparado com o modelo centralizado podem existir desvantagens desnecessárias se não for realmente necessário. No modelo distribuído pode ser necessário

fazer actualizações frequentes para os manter a rede sincronizada, sendo necessário avaliar se este custo é realmente necessário e compensa. Em todo caso se necessário seria fácil transformar o modelo centralizado num modelo distribuído, com técnicas já conhecidas, onde também seria fácil de implementar com Gossip.

2.4 Sumário

A coordenação de Web Services baseados em SOAP, é realizada segundo a especificação dos protocolos WS-Coordination, WS-ReliableMessaging, WS-AtomicTransaction, que não se adequam ao protocolo REST. Do que foi exposto, pode-se depreender que as propriedades do Gossip poderão aproximar essa coordenação, sem violar os princípios do REST.

Capítulo 3

Implementação

Este capítulo, descreve a implementação utilizada para a coordenação de Web Services utilizando um componente de *middleware*, que permite transformar um serviço Atom centralizado num serviço distribuído. Esta implementação recorre a técnicas de Gossip de forma a disseminar a informação num ambiente distribuído, com o uso do protocolo REST.

3.1 Descrição do Problema

Considerando um cenário típico de aplicação de Web Services REST, em que se pretende a coordenação de Web Services, foi escolhido um serviço Atom centralizado, que se deseja transformar num serviço distribuído. Surge, assim, o problema de coordenar vários servidores Atom, de forma a que qualquer alteração de informação seja propagada para todos os servidores existentes.

No serviço centralizado de Atom, apenas existe um único servidor responsável por receber todas as alterações e propaga-las aos restantes. Na Figura 3.1 pode-se ver uma representação, onde o servidor C1 publica informação no S1, sendo este responsável por propagar toda a informação para os restantes C2, C3, C4 e C5.

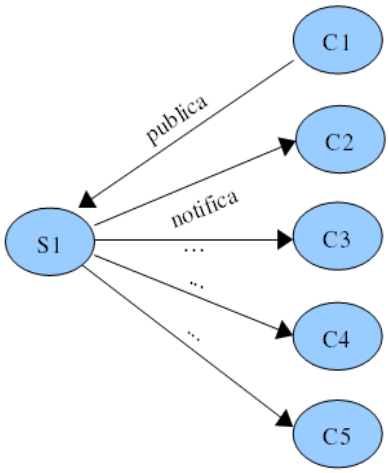

Figura 3.1: Serviço Atom centralizado

O desafio desta abordagem passa por criar um serviço de Atom distribuído, garantindo a existência de atomicidade na actualização de mensagens de todos os servidores, assim como a garantia que a performance não é afectada, mas também poderá ser melhorada, reduzindo possíveis congestionamentos de rede e garantindo a existência de escalabilidade do serviço sem afectar directamente a rede.

3.2 Fundamentação

Com o intuito de focar os objectivos na resolução do problema supracitado num ambiente distribuído, faz sentido usar o Gossip para o solucionar. Para tal foi utilizado um exemplo básico de disponibilização de conteúdos onde cada servidor é responsável por armazenar e disponibilizar todas as mensagens que lhe forem enviadas. O modelo de distribuição deve garantir que todos os servidores disponibilizam a mesma informação com a partilha de conhecimento entre os mesmos.

Cada servidor pertencente à rede de distribuição tem poder para armazenar mensagens recebidas de outros servidores ou da entidade que publica pela primeira vez. Cada um têm a responsabilidade de propagar para outros servidores, actualizando o seu conhecimento sobre a informação que circula na rede.

Para o modelo de gestão de conteúdos já existem *standards* disponíveis, tal como o Atom, que permite ter um modelo de disponibilização de informação de forma centralizada (ver Figura 3.2), possibilitando que vários serviços se liguem para publicar informação e os restantes para consumir os conteúdos publicados. O Atom foi escolhido por permitir uma fácil adaptação aos objectivos pretendidos uma vez que este utiliza a mesma semântica de REST para a gestão de conteúdos.

Figura 3.2: Atom Centralizado

De forma a atingir os objectivos definidos pretende-se utilizar as funcionalidades do serviço Atom, mas adaptadas a um modelo distribuído, dado que este, sendo baseado num modelo centralizado, apenas permite publicação de informação numa única localização. Pretende-se assim criar um serviço distribuído que permita ter toda a informação disponível por vários servidores onde cada um disponibiliza todas as funcionalidade de publicação e consulta de conteúdos.

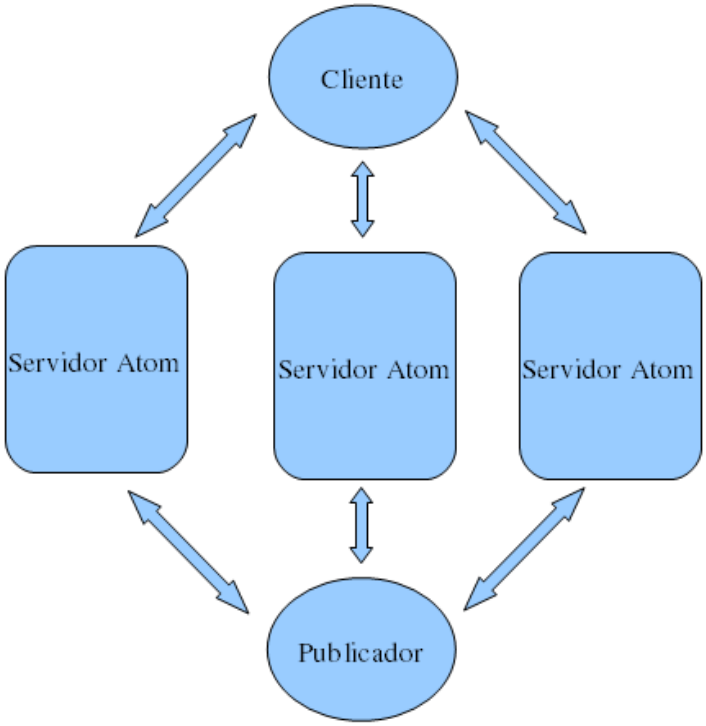

Figura 3.3: Atom Distribuído

A Figura 3.3 representa um sistema distribuído de serviços Atom, tal como pretendido, onde pode existir um ou mais clientes a consumir os conteúdos e um ou mais publicadores de conteúdos. Os vários servidores garantem a publicação e consulta de conteúdos através da ferramenta Atom, mas não têm capacidade para garantir a distribuição de informação nem a sua consistência. Para tal, foi necessário desenvolver um componente de *middleware* que garantisse a propagação da informação publicada para todos os outros servidores. A implementação deste componente passa pela sua inclusão no serviço Atom, onde é responsável por intersectar alterações dos conteúdos do serviço fazendo a propagação da informação para os restantes servidores. Todas as operações realizadas para publicação, consulta ou distribuição, baseiam-se no uso do protocolo REST através de operações de HTTP do tipo GET, POST, PUT e DELETE. As operações de leitura não deverão ser consideradas para a distribuição de informação pois não provocam alterações no sistema.

Todos os servidores existentes na rede são responsáveis por propagar os conteúdos recebidos para todos os outros através do novo componente de Software. Cada um deverá

garantir a consistência de informação verificando que não existe duplicação de dados. Isto porque numa rede onde os vários servidores se encontram ligados entre si, estes podem enviar dados para um servidor que já contém essa mesma informação. Este último deverá então ignorar a informação recebida, caso já tenha o seu conhecimento através de outro servidor. Esta duplicação de dados na rede e a forma como será distribuída depende do modelo de distribuição escolhido, que será referido em maior detalhe nos capítulos seguintes.

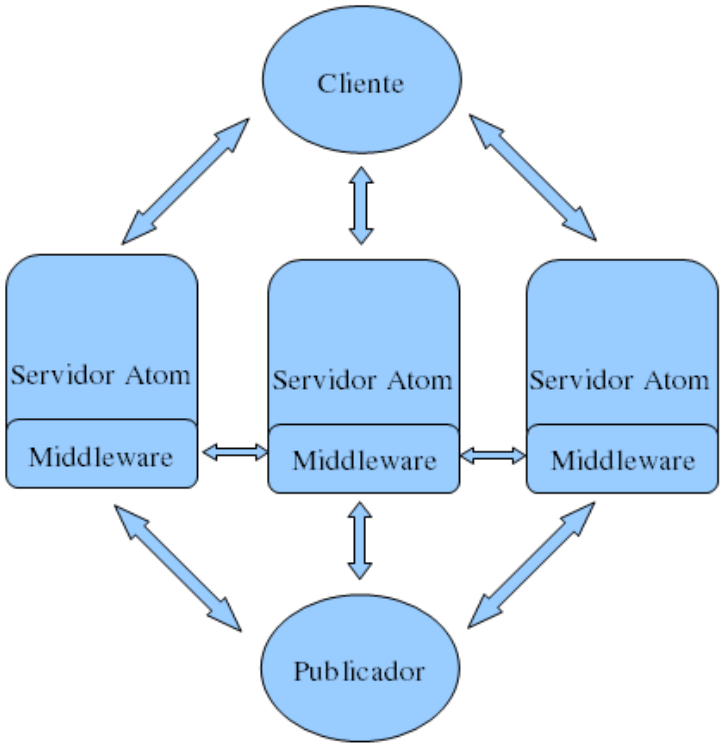

Figura 3.4: Atom distribuído com aplicação

A Figura 3.4 representa o modelo distribuído completo, constituído por vários servidores que disponibilizam o serviço Atom, incluindo o componente de Software responsável pela distribuição da informação por todos os servidores existentes na rede. Este componente é também responsável por implementar todos os conceitos dos modelos de distribuição utilizados, isto é, a distribuição de mensagens está sujeita a um modelo de distribuição definido no momento da execução em que esta deve respeitar o comportamento definido.

3.2.1 Modelo de distribuição

O modelo distribuído Gossip, baseia-se num conjunto de servidores que disponibiliza o serviço Atom e garante que todos comunicam entre si de forma a garantir que a informação que cada um contém é distribuída por todos, através de métodos Gossip.

O modelo Gossip permite disseminar informação a partir de diversos métodos, tais como foram supracitados no capítulo 2.3.1, onde é necessário garantir que a informação é distribuída de forma coerente e que chega a todos os servidores rapidamente e sem congestionar a rede. Nesse sentido, foi criado um modelo através do conhecimento total da tipologia da rede onde cada servidor tem o conhecimento do mesmo número de vizinhos.

A integração do modelo de Gossip com o protocolo REST, foi realizada facilmente através do uso de uma semântica semelhante, que permite associar os métodos do Gossip com o REST. Tal como foi referido no capítulo 2.1.6 o REST utiliza os métodos de HTTP (PUT, POST, DELETE, GET), como forma de realizar operações na sua comunicação. Utilizando um exemplo básico, onde um serviço disponibiliza operações de leitura e escrita de dados usando as respectivas operações de GET para leitura e PUT, POST e DELETE para realizar alterações nos dados.

No Gossip tal como referido no capítulo 2.3, a disseminação de informação é realizada através da utilização de operações de Push e Pull, isoladamente ou em conjunto, onde por exemplo na necessidade de disseminar informação é utilizado o método Push como forma de enviar dados para um conjunto de servidores e o Pull para recolher informação de determinados servidores. Analisando as operações destes dois modelos evidencia-se a semelhança da semântica, pois embora com nomes diferentes as operações são equivalentes:

- Push corresponde ao PUT, POST ou DELETE;
- Pull representa o GET;
- Push/Pull, será uma conjunção dos dois.

A utilização de um modelo de Gossip para implementar o serviço de Atom distribuído, foi realizado da seguinte forma: uma mensagem que vai ser publicada no serviço de Atom é efectuada através de um método de POST pelo cliente, de forma a enviar uma nova mensagem actualizando o serviço com novos dados. O servidor que detêm o serviço Atom procede à verificação da informação (verifica se é duplicada) e, caso seja válida, efectua a disseminação da informação para todos os servidores que estejam ao seu alcance, utilizando uma operação de Push em Gossip onde em REST corresponde a um POST para cada servidor. Os restantes servidores na rede que recebam um pedido semelhante vão actuar da mesma forma, até que a rede esteja toda actualizada com a nova informação.

Caso existisse a necessidade de a determinado momento um servidor consultar outro para se actualizar, seria utilizado o método de Pull de Gossip onde em REST temos uma correspondência de uma operação de GET. Assim podemos modular um sistema baseado em Gossip, olhando apenas às suas características e sem nos preocuparmos efectivamente com a sua implementação, pois o REST utiliza uma semântica semelhante que resolve todo os problemas que daí poderiam resultar.

3.2.2 Membership

Para a utilização dos modelos de distribuição, é necessário organizar todos os servidores de forma a definir uma estrutura de comunicação. Todos os servidores ao fim de determinado período de tempo têm de estar actualizados com a nova informação. Terá de existir uma ligação entre eles de forma a propagar a informação. Aqui surge o conceito de *membership*, que permite definir que determinado servidor tem conhecimento de um conjunto de servidores vizinhos. A quantidade ou quais os servidores que este deve conhecer apenas depende do modelo pretendido ou sua configuração.

A configuração de rede é disponibilizada por um serviço implementado por uma aplicação independente do contexto tratado. Este permite a consulta de estrutura da rede a qualquer momento. A forma como os servidores consultam este serviço, neste protótipo, estabelece-se na fase inicial do seu arranque. Como o objectivo é retirar métricas de tempos de execução, cada um apenas consulta uma única vez o serviço de forma a adquirir a listagem dos servidores vizinhos que conhece evitando a perda de tempo ou o congestionamento da rede.

Esta estratégia permite alterar a estrutura da rede quando necessário, sem alterações nos servidores de aplicações.

A utilização do modelo de *membership* centralizado é adequado, uma vez que se torna eficiente nesta aplicação. Esta solução é também utilizada em aplicações conhecidas mundialmente, tal como, o Skype, que adoptam a mesma escolha.

3.3 Arquitectura

A arquitectura é definida em duas camadas distintas, de software e infra-estrutura. A arquitectura de software consiste em dois componentes diferentes que têm como objectivo implementar as necessidades apresentadas nesta dissertação. Um é responsável por armazenar a organização de todos os nós existentes, disponibilizando, quando pedido, uma listagem de nós vizinhos para a comunicação (*membership*). O outro componente é responsável por todo o trabalho de gestão dos seus conteúdos, assim como fazer a propagação dos mesmos quando criados ou alterados. Este utiliza o primeiro componente para verificar quais os nós vizinhos que tem disponíveis.

3.3.1 Componentes de software

O componente de *middleware* desenvolvido foi dividido em dois componentes de Software, designadamente um responsável pela gestão de *membership* e outro por todo o funcionamento de troca de mensagens e gestão de conteúdos.

Todos os componentes dão uso ao protocolo REST como modo de comunicação entre si. Para tal foi utilizada uma implementação deste protocolo através do JAX-WS que

disponibiliza as bibliotecas de software em Java. Para o uso do serviço de Atom foi utilizada a ferramenta Apache Abadera que implementa os protocolos de Atom na linguagem Java, permitindo de forma flexível a inclusão de novos componentes ou a alteração de comportamentos.

Primeiro componente

O primeiro componente é responsável pelo armazenamento da organização dos nós existentes. Isto é possível através de uma parametrização inicial com o *layout* da rede onde para cada servidor existente está atribuída uma lista de servidores vizinhos conhecidos. Este componente disponibiliza Web Services baseados no protocolo REST como modo de comunicação entre os servidores, tentando assim reduzir ao máximo a latência de respostas ou mesmo o congestionamento da rede.

Durante o seu funcionamento cada servidor terá de entrar em contacto com o servidor de *membership* para consultar o *layout* que deverá conhecer, isto é, um servidor faz um pedido via Web Services fornecendo a sua identificação, onde obtém como resultado uma listagem de nós vizinhos para os quais ele deverá comunicar.

Segundo componente

Actualmente o serviço típico de Atom já permite a publicação de conteúdos e sua consulta de forma centralizada. Fazendo parte dos objectivos tornar o serviço centralizado num serviço distribuído, foi desenvolvido este segundo componente de software responsável por toda a lógica de distribuição e recepção de mensagens, garantindo que a informação esteja distribuída de igual forma por todos os servidores existentes.

Este componente de Software foi incluído no serviço Atom para garantir a intersecção de todos os pedidos de alteração de conteúdos, permitindo um controlo de informação antes do destino final (serviço Atom). A intersecção de pedidos é necessária perante duas situações extremamente importantes. Em primeiro lugar, quando o serviço recebe uma alteração de conteúdos necessita de distribuir essa mesma alteração para todos os vizinhos conhecidos. A segunda situação ocorre quando este serviço recebe informação repetida comportando-se de forma a ignorar o pedido pois, se já existe, não pode criar informação duplicada. Então, não distribui a informação pois sendo duplicada é porque já o fez anteriormente.

A implementação deste componente foi baseado em Web Services segundo o protocolo REST, tanto para intersectar alterações como para fazer a distribuição das novas alterações para o resto da rede, através de módulos com responsabilidades especificas, módulo de *membership*, módulo de distribuição e módulo de gestão de conteúdos.

Módulo de Membership

Utilizado aquando o arranque da aplicação, para verificar quais os nós vizinhos que tem conhecimento. Como o objectivo é retirar métricas da distribuição das mensagens, não foi muito relevante a necessidade de actualizar o seu conhecimento sobre a rede durante o processo. Se nos encontrássemos noutra situação, isto poderia ser ajustado com uma actualização, de forma periódica ou com outra componente decisiva.

Assim, quando utilizado, este módulo é responsável por entrar em contacto com o servidor de *membership* através de RESTful Web Services e fazer um pedido para obter uma listagem de servidores vizinhos. Esta é armazenada para evitar que para cada envio de mensagens seja necessário realizar um novo pedido.

Módulo de distribuição

Quando um conteúdo é criado ou alterado, este módulo é responsável por fazer a distribuição do mesmo para um conjunto de servidores conhecidos como vizinhos. Esta distribuição é realizada através RESTful Web *Services* e do protocolo de Atom de forma a publicar conteúdos nos servidores vizinhos. Estes depois são intersectados pela aplicação localizada nesse serviço que será responsável por tratar o pedido.

Módulo de gestão de conteúdos

Este módulo é responsável por manter a coerência dos conteúdos para cada servidor, disponibilizando operações que permitem verificar se o conteúdo em questão já existe, caso contrário o conteúdo é inserido no sistema de forma a ficar disponível para os clientes.

3.3.2 Modelo da arquitectura

Os componentes supracitados trabalham em conjunto e são representados segundo uma arquitectura apresentada na figura 3.5, que representa um serviço de Atom a ser utilizado num ambiente distribuído.

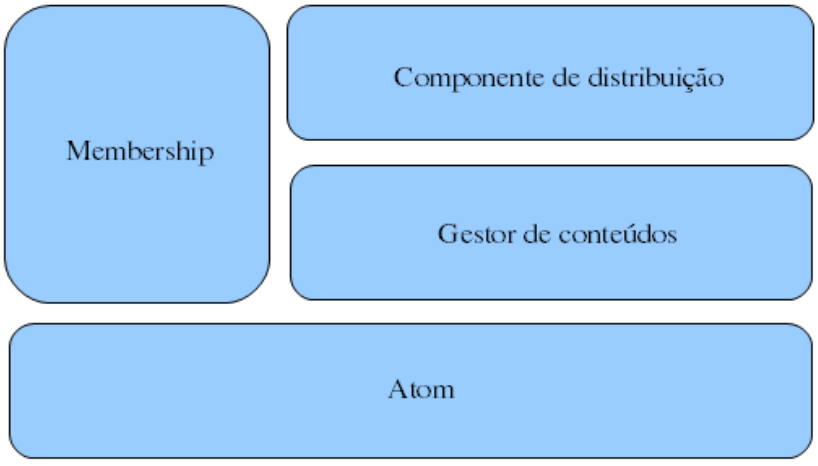

Figura 3.5: Arquitectura

Na implementação dos componentes com o protocolo REST, foi utilizado o JAX-RS que disponibiliza uma API [MH08] que permite criar Web Services baseados na arquitectura REST, através de um conjunto de anotações que permitem mapear classes Java em recursos Web. Algumas as anotações são representadas da seguinte forma:

- @Path – para indicar qual o caminho de determinado recurso que representa uma classe;
- @GET, @PUT, @POST, @DELETE – indicam quais os tipos de operações permitidas em cada recurso;
- @Produces, @Consumes – indicam quais os tipos de dados que são produzidos ou consumidos pelos recursos disponibilizados;
- @PathParam, @QueryParam, @HeaderParam, entro muitos outros oferecem um grande leque de propriedades aos recursos disponíveis.

Para o serviço Atom foi utilizada a ferramenta *Apache Abadera* que é um projecto da autoria da *Apache Software Foundation* e visa uma implementação funcional completa e de performance elevada do *IETF Atom Syndication Format* e do protocolo *Atom Publishing Protocol* [Fou10a]. Este projecto disponibiliza todas as funcionalidades dos serviços de Atom, através de bibliotecas de software que podem ser integradas em qualquer componente de tecnologia Java oferecendo uma independência da plataforma utilizada.

Todos estes componentes e ferramentas são executados por um servidor aplicacional designado por Jetty, que é uma ferramenta de software implementada Java, mais propriamente um servidor de HTTP com suporte de Servlets que permite disponibilizar aplicações tais como os Web Services ou muitas outras [Fou10b]. Este servidor aplicacional foi escolhido pela sua simplicidade, eficiência e facilidade de integração com Web Services. Todas as aplicações desenvolvidas foram executadas num servidor Jetty, mais propriamente cada servidor Atom era executado com uma instância independente do Jetty representado um servidor isolado, onde mantinha todas as propriedades dos serviços Atom.

Capítulo 4

Avaliação

Este capítulo, apresenta uma descrição dos resultados obtidos através da implementação de um componente de *middleware*, para a coordenação de Web Services. Os resultados apresentados têm como base a comparação de dois modelos centralizados com um modelo distribuído, finalizados com uma discussão onde estes são interpretados à luz do conhecimento actual.

4.1 Infra-estrutura

A infra-estrutura está organizada por quatro servidores, com as mesmas características e ligados em rede de 1GB de forma a tentar minimizar ao máximo a latência de rede. Cada servidor é responsável por executar uma série de componentes de software, que podem variar segundo o tipo de recolha de métricas a realizar para os modelos: centralizado com notificação, centralizado com *polling* e distribuído com Gossip.

4.1.1 Modelo centralizado (Notificação)

No modelo centralizado por notificação, apenas existe um servidor responsável por receber todas as alterações e fazer a distribuição para os restantes. Existe, pois, uma máquina que executa o componente de software de *membership* e um servidor Atom para fazer a distribuição. As restantes máquinas executam 40 servidores de Atom em conjunto, recebendo as actualizações do primeiro servidor.

A Figura 4.1 representa a infra-estrutura utilizada para o modelo centralizado por notificação. Aqui pode-se observar que existem quatro máquinas representadas pelas letras A, B, C e D que são responsáveis por executar os componentes de Software tal como mencionado.

Na máquina A temos o componente M1 que representa o componente de Software de *membership* e o componente S1 que representa um serviço Atom centralizado. Nas restantes máquinas existe um componente designado com a letra P que é responsável por

fazer envios de nova informação para disponibilizar no serviço Atom. Os componentes C são clientes que vão consumir os dados publicados pelo serviço.

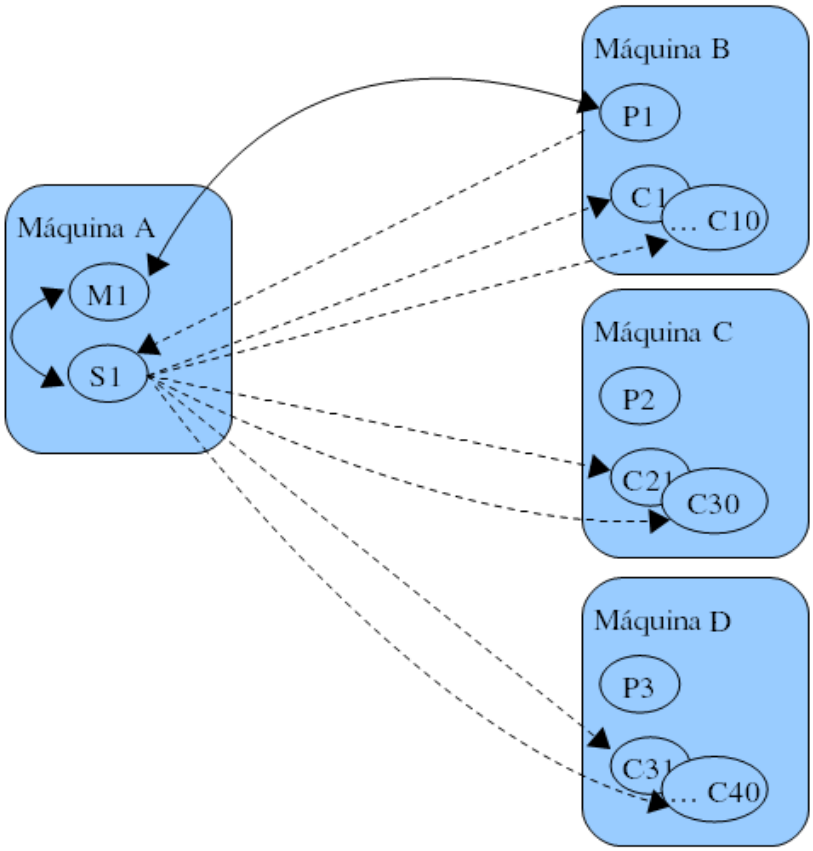

Figura 4.1: Infra-estrutura do modelo centralizado por notificação

O fluxo de informação resulta de uma primeira fase de arranque onde todos os componentes S, P e C consultam o componente M para verificar quais os servidores conhecidos. Numa segunda fase os componentes P publicam informação no servidor S e este é responsável pelo envio a todos os servidores C.

4.1.2 Modelo centralizado (Polling)

À semelhança do modelo de notificação, neste, apenas existe um servidor responsável pela distribuição de alterações, mas ao contrário do modelo de notificação, em vez de enviar as alterações este fica à espera que algum servidor o consulte sobre alterações.

A Figura 4.2 representa a infra-estrutura utilizada para o modelo centralizado por Polling. Esta é semelhante à infra-estrutura do modelo centralizado por notificação mantendo-se o mesmo número de máquinas e instâncias de servidores, à excepção do comportamento dos componentes designados pela letra S e C. O componente S, à semelhança do modelo anterior, é responsável pela recepção de novas alterações a conteúdos, mas ao contrário

deste, não distribui a informação pelos restantes componentes C, apenas disponibiliza para consulta e os componentes C é que são responsáveis por consultar alterações de forma frequente.

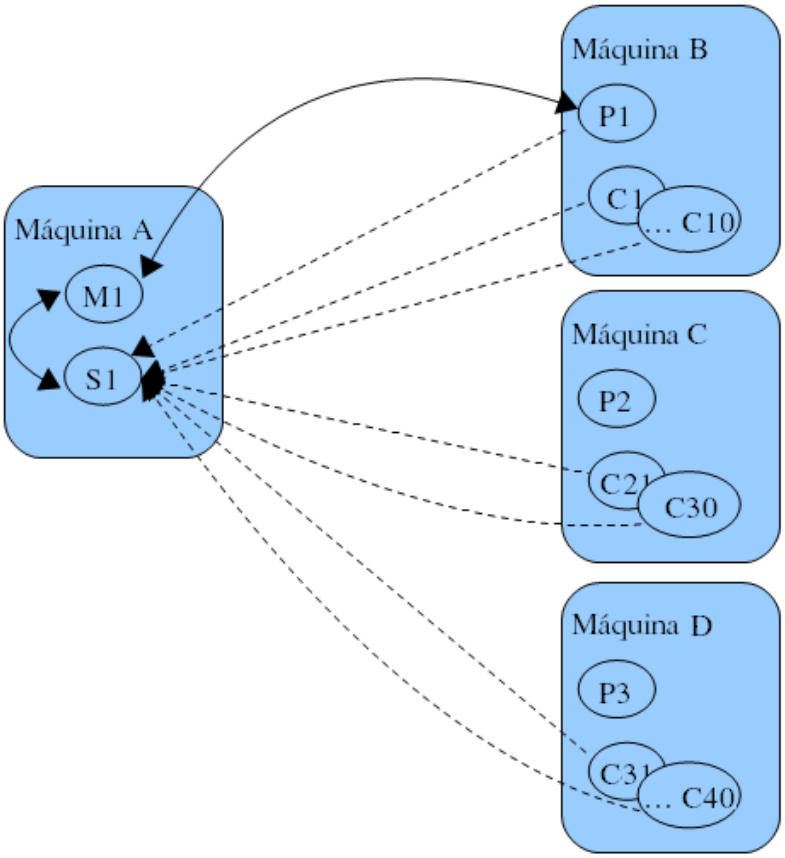

Figura 4.2: Infra-estrutura do modelo centralizado por Polling

4.1.3 Modelo Gossip

No modelo de Gossip, a quantidade de máquinas continua a ser a mesma face aos modelos anteriores, mas o comportamento dos componentes altera-se de forma significativa. Este modelo continua a ter uma máquina com um componente de *membership*, e as restantes com os componentes de Software do serviço de Atom. Os componentes que disponibilizam o serviço Atom passam a ter um comportamento diferente pois agora estes comunicam todos entre si baseados no modelo Gossip, fazendo a troca de mensagens.

A Figura 4.3 representa a infra-estrutura do modelo de Gossip, que, à semelhança dos modelos anteriores, mantém as mesmas máquinas designadas pelas letras A, B, C e D onde os componentes M e P mantêm todas as suas características. Os componentes P passam agora a publicar informação nos servidores C que forem indicados pelo serviço de *membership* e os componentes C são agora responsáveis por gerir conteúdos distribuindo al-

terações para outros servidores e disponibilizando a sua consulta através dos serviços Atom. O fluxo de informação varia segundo a topologia da rede, ordem e atraso da distribuição.

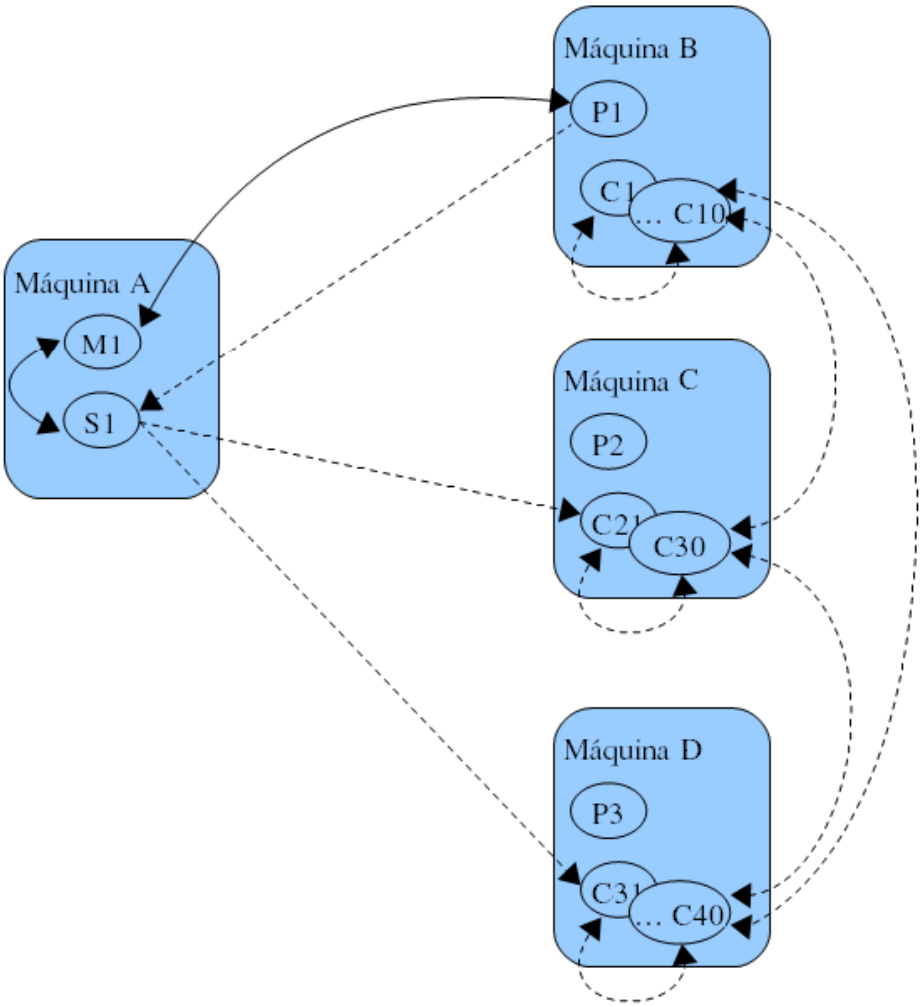

Figura 4.3: Infra-estrutura do modelo de Gossip

4.2 Recolha de métricas

Para recolher métricas sobre as operações realizadas foi necessário utilizar o relógio interno de cada servidor, aqui surge um problema de sincronização dos relógios entre todas as máquinas. De forma a evitar este problema de sincronização, foi definido que todas as medições temporais apenas se realizam nas próprias máquinas, isto é, a medição da propagação da mensagem desde o publicador até ao servidor que a recebe é realizado na mesma máquina garantindo que o relógio interno utilizado é o mesmo. Isto garante que o tempo utilizado nos *logs* será sempre do mesmo relógio da mesma máquina, descartando possíveis erros de medições por motivos de sincronismo.

Resumindo, cada máquina tem, exceptuando a responsável pelo *membership*, um conjunto de servidores Atom e um publicador de mensagens de forma a garantir que cada servidor apenas monitoriza os tempos de propagação caso a mensagem tenha origem do publicador do mesmo servidor. Para tornar o sistema real, dados transmitidos de diferentes máquinas serão apenas propagados, onde não constarão nas medições de tempos.

4.3 Resultados

Esta secção apresenta os resultados obtidos nos três modelos referidos anteriormente, onde para cada modelo são representados gráficos com a análise do atraso ao longo do tempo, da probabilidade de atraso e do atraso médio por servidor. Para todos os valores temporais é utilizada a unidade de medida em milissegundos.

4.3.1 Modelo Centralizado (Notificação)

No modelo centralizado por notificação foi recolhida uma amostra de resultados que se representa segundo a Figura 4.4, onde para um determinado período de tempo foi determinado para cada mensagem qual o atraso na sua propagação. O tempo apresentado na figura, varia entre o período inicial de zero até aos cinco minutos e quarenta segundos, onde o atraso atinge valores máximos entre os 12000 e 14000 milissegundos.

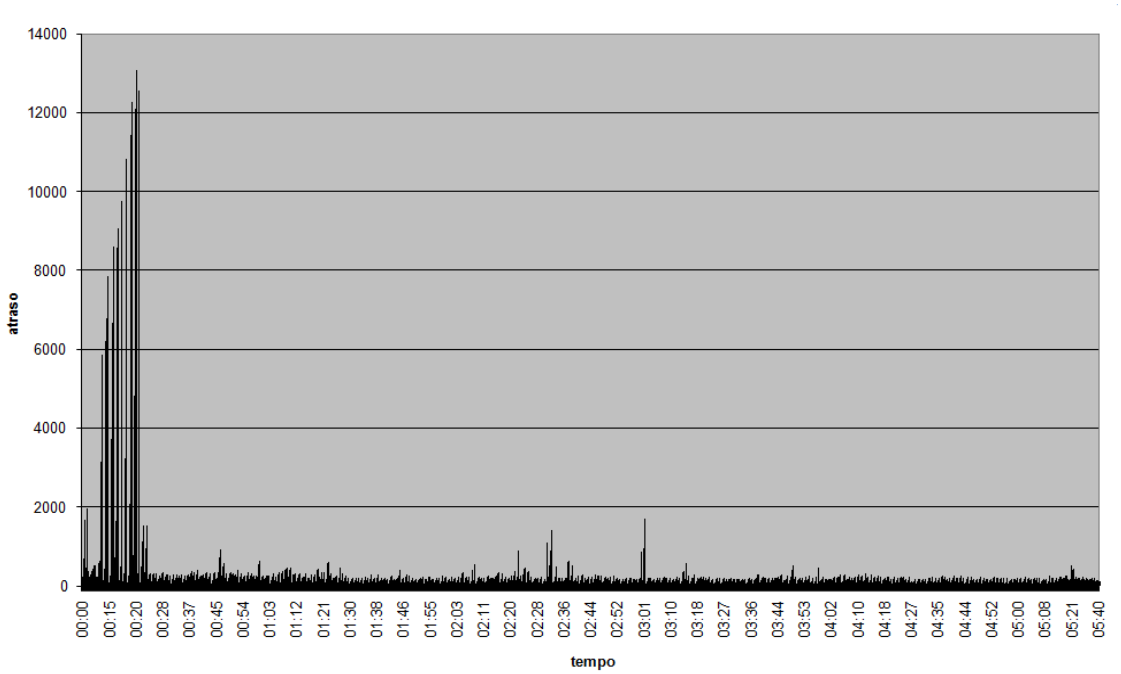

Figura 4.4: Modelo centralizado por notificação total

Observando o gráfico da Figura 4.4 em maior detalhe, observa-se que no período inicial existem grandes picos de atraso com valores muito superiores à média, embora maior parte

dos valores fica muito abaixo do 2000 milissegundos com poucas excepções por volta do período "02:30"e "03:00".

Para se obter uma maior percepção dos valores fora dos picos excepcionais, foi retirada a amostra inicial entre o período zero e os trinta segundos. Ficando assim apenas uma amostra de valores a partir do período "00:30", representado na Figura 4.5.

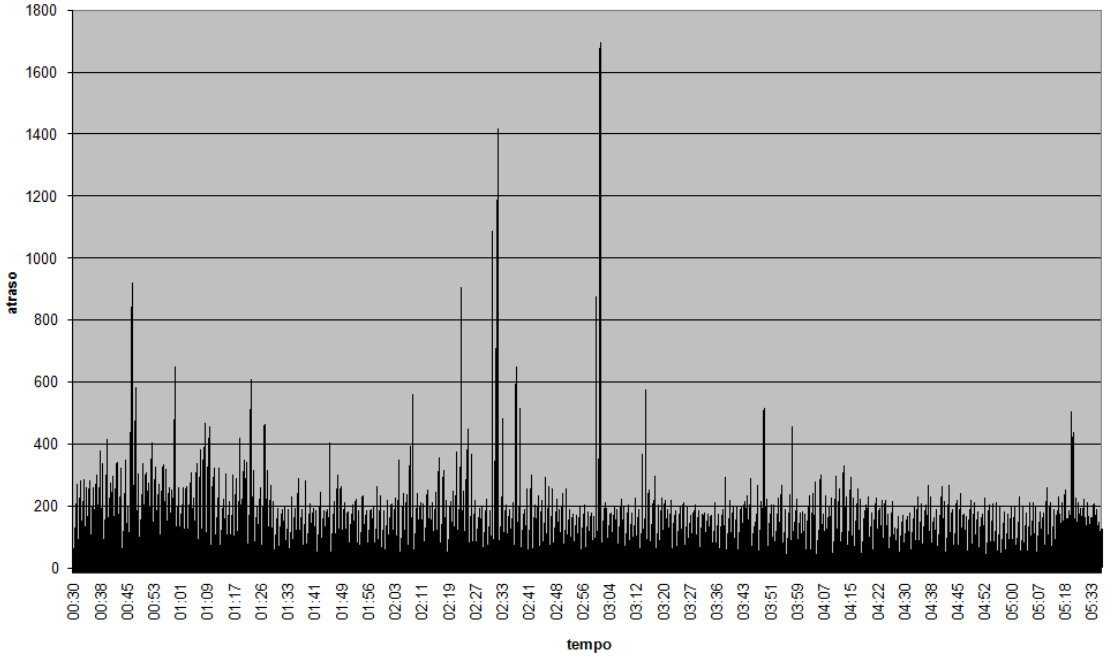

Figura 4.5: Modelo centralizado por notificação filtrado

Na Figura 4.5 observa-se que a maioria dos valores está abaixo de 200 milissegundos, existindo alguns casos que ultrapassam este limite. Na parte inicial entre o período "00:30"e "01:26", é onde existe mais valores acima dos 200, nos restantes existem alguns casos de onde se deve realçar os grandes picos. Estes encontram-se com valores muito elevados nos períodos "02:32", "03:02", onde chegam a atingir que ultrapassam os 1400 milissegundos e os 1600 respectivamente.

De forma a obter valores que possam traduzir da melhor forma o atraso da distribuição de mensagens, foi gerado um gráfico (ver Figura 4.6) que apresenta a probabilidade de atraso. Este indica a probabilidade de atraso na entrega das mensagens para os servidores existentes na rede.

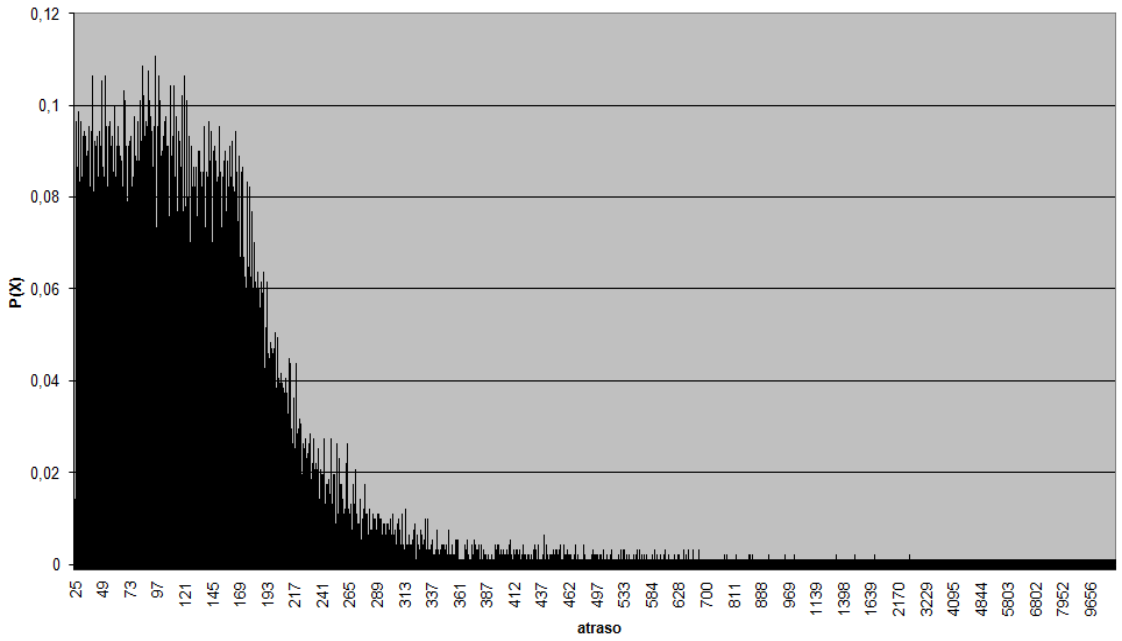

Figura 4.6: Modelo centralizado por notificação - probabilidade de atraso

Observando em detalhe a Figura 4.6, onde o eixo vertical representa a probabilidade de atraso e o eixo horizontal o respectivo atraso, verifica-se que existe uma grande probabilidade de atraso entre os valores [25,193]. Onde os valores da probabilidade chegam a ultrapassar os 0.1, seguindo-se as probabilidades menores no intervalo]193,361] que representam probabilidades abaixo de 0.02 e os restante com uma probabilidade muito reduzida.

De forma a perceber qual a relação do atraso relativamente a cada servidor existente, está representado na Figura 4.7 um gráfico com o atraso médio por servidor. Onde é possível observar os valores do atraso de entrega de mensagens em cada servidor disposto na rede distribuída.

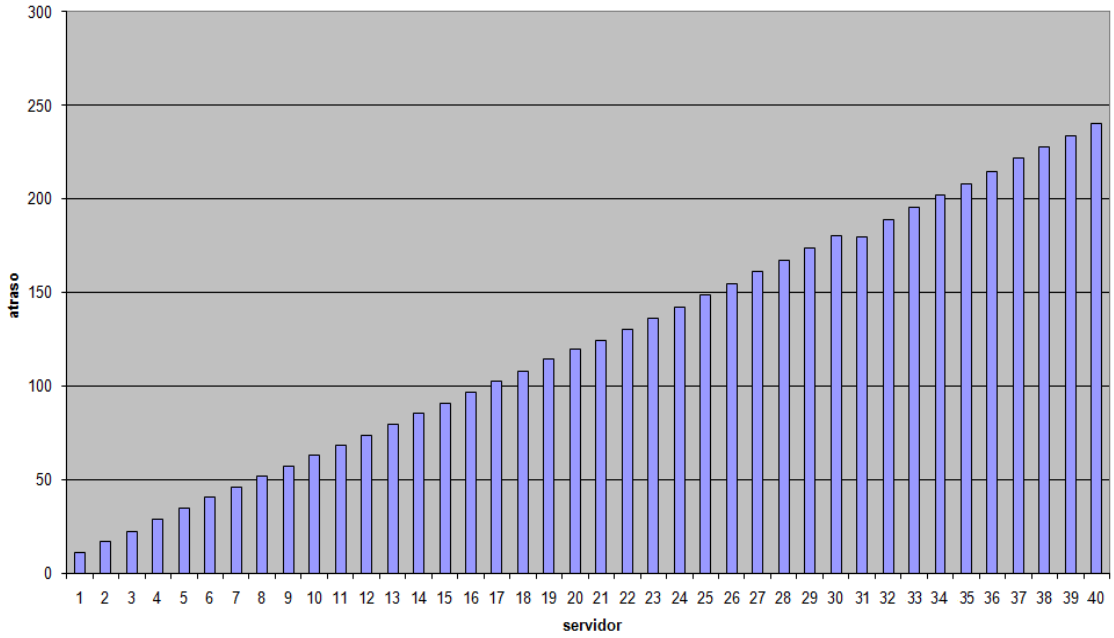

Figura 4.7: Modelo centralizado por notificação - atraso médio por servidor

Na Figura 4.7, onde o eixo vertical representa o atraso da propagação da mensagens e o horizontal o servidor onde ocorre a medição. Os servidores estão representados por uma numeração de um a quarenta. Verifica-se que o atraso médio é crescente segundo a numeração dos servidores. Este atraso varia desde 20 até um valor máximo de 240.

4.3.2 Modelo Centralizado (Polling)

No modelo centralizado por *polling* foi recolhida um amostra de resultados que se representa na seguinte na Figura 4.8, onde para um determinado período de tempo foi determinado o atraso de propagação de cada mensagem. O período da amostra varia entre zero e no valor final de cinco minutos e quarenta segundos.

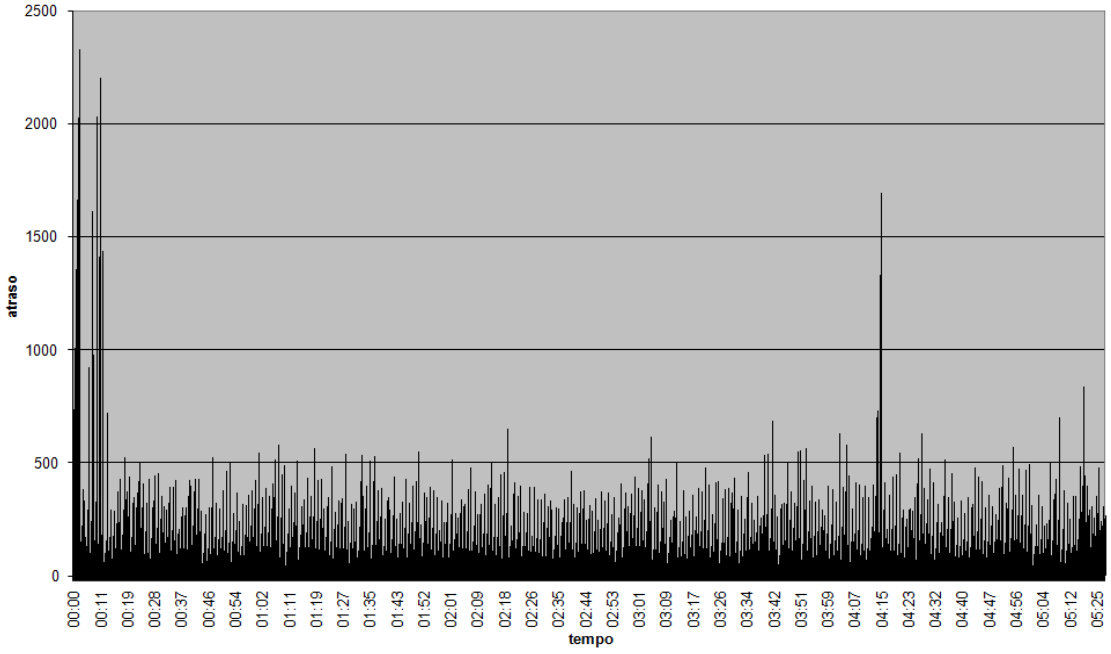

Figura 4.8: Modelo centralizado por polling total

A Figura 4.8 representa a amostra total de resultados para o Modelo Centralizado por Polling. Nela podem-se visualizar grandes picos no início, que chegam a ultrapassar o valor de 2000 milissegundos e um valor excepcional no período "04:12"que ultrapassa o valore de 1500 milissegundos. A maior parte dos valores observadores encontram-se abaixo do 500 milissegundos e com muito maior intensidade abaixo dos 200 milissegundos que representam maior parte dos valores observados..

De forma a visualizar melhor a amostra, foi retirado o período inicial que continha maior parte dos valores excepcionais, restando assim a maioria dos valores abaixo dos picos, como se pode verificar na Figura 4.9.

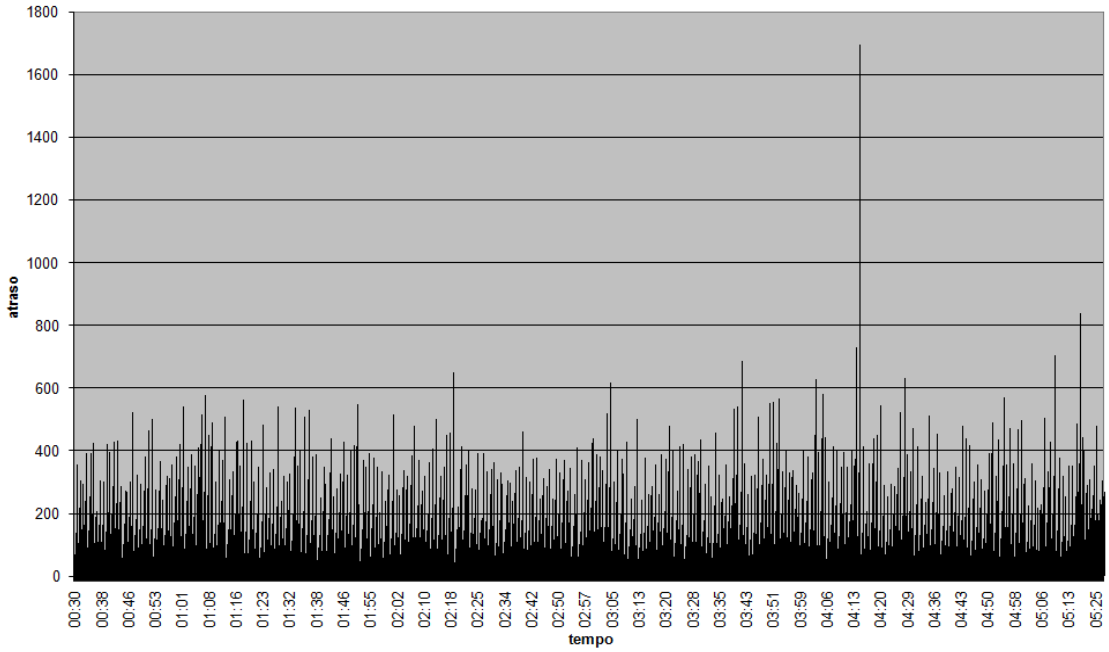

Figura 4.9: Modelo centralizado por polling filtrado

Na Figura 4.9 pode-se verificar um grande pico excepcional no período "04:12", que chega a ultrapassar os 1600 milissegundos. Embora a maioria dos resultados encontram-se muito abaixo deste valor, atingindo valores abaixo dos 200 milissegundos, existindo no entanto, bastantes acima deste atingindo valores de 400 milissegundos.

De forma a obter valores que possam traduzir da melhor forma o atraso da distribuição de mensagens, foi gerado um gráfico (ver Figura 4.10) que apresenta a probabilidade de atraso. Esta probabilidade de atraso representa o envio de mensagens para servidores, onde o atraso de entrega foi registado para o cálculo da probabilidade, à semelhança de resultados já apresentados no modelo anterior.

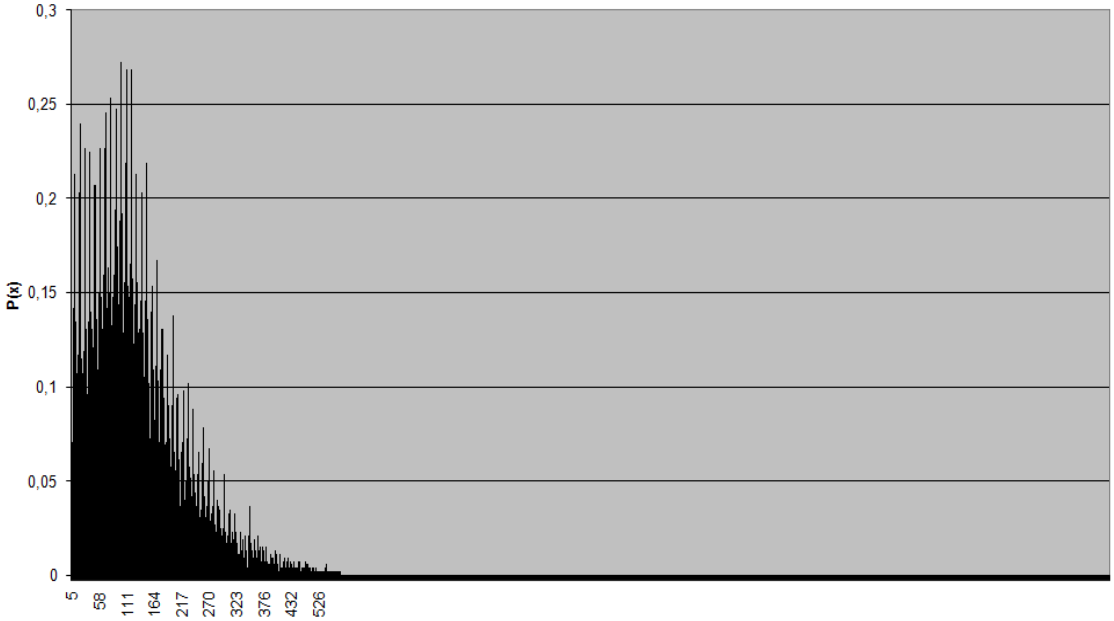

Figura 4.10: Modelo centralizado por polling - probabilidade de atraso

A Figura 4.10 apresenta os valores de probabilidade de atraso, onde o eixo vertical representa a probabilidade e o horizontal o respectivo atraso. O atraso varia entre os valores 5 e 1694 milissegundos, onde se pode verificar uma maior intensidade no intervalo [5; 213]. A maior probabilidade encontra-se abaixo de 0,15 embora existam vários picos de grandes valores ligeiramente acima de 0,25.

De forma a perceber qual a relação do atraso relativamente a cada servidor existente, está representado na Figura 4.11 um gráfico com o atraso médio por servidor.

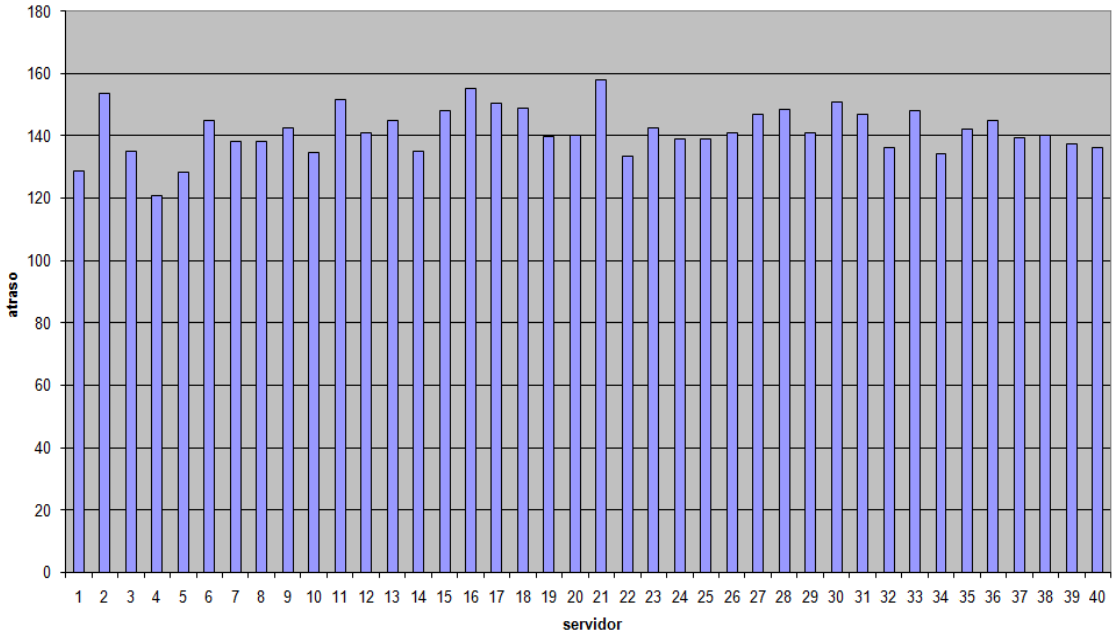

Figura 4.11: Modelo centralizado por polling - atraso médio por servidor

Na Figura 4.11 o eixo vertical representa o atraso e o horizontal o respectivo servidor onde ocorre a medição. Os servidores estão representados por uma numeração de um a quarenta, onde se pode observar que em todos os servidores o atraso médio anda próximo de 140 milissegundos, existindo algumas pequenas diferenças.

4.3.3 Modelo Gossip

No modelo Gossip foi recolhida um amostra de resultados apresentada na Figura 4.12, onde para um determinado período de tempo foi determinado o atraso de propagação de cada mensagem.

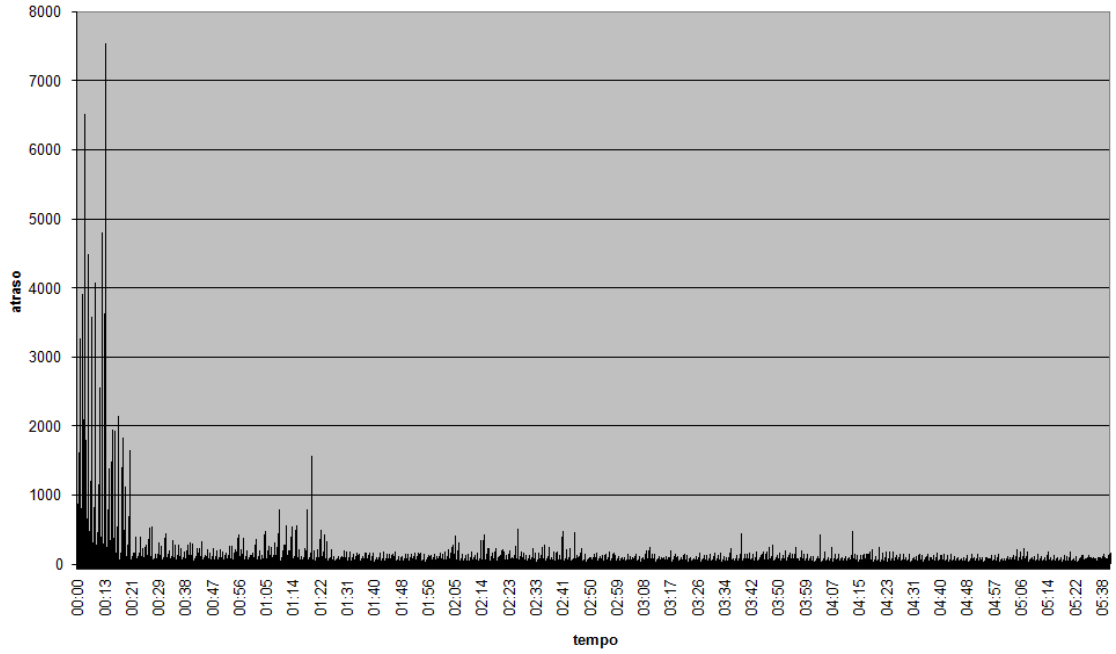

Figura 4.12: Modelo Gossip total

A Figura 4.12 representa a amostra total de resultados para o Gossip, onde se podem visualizar grandes picos no inicio, e alguns mais pequenos no período "01:21". Os restantes encontram-se muito abaixo dos 1000 milissegundos.

De forma a visualizar melhor a amostra, foi retirada a fase inicial onde existiam os grandes picos. Obtendo-se a Figura 4.13.

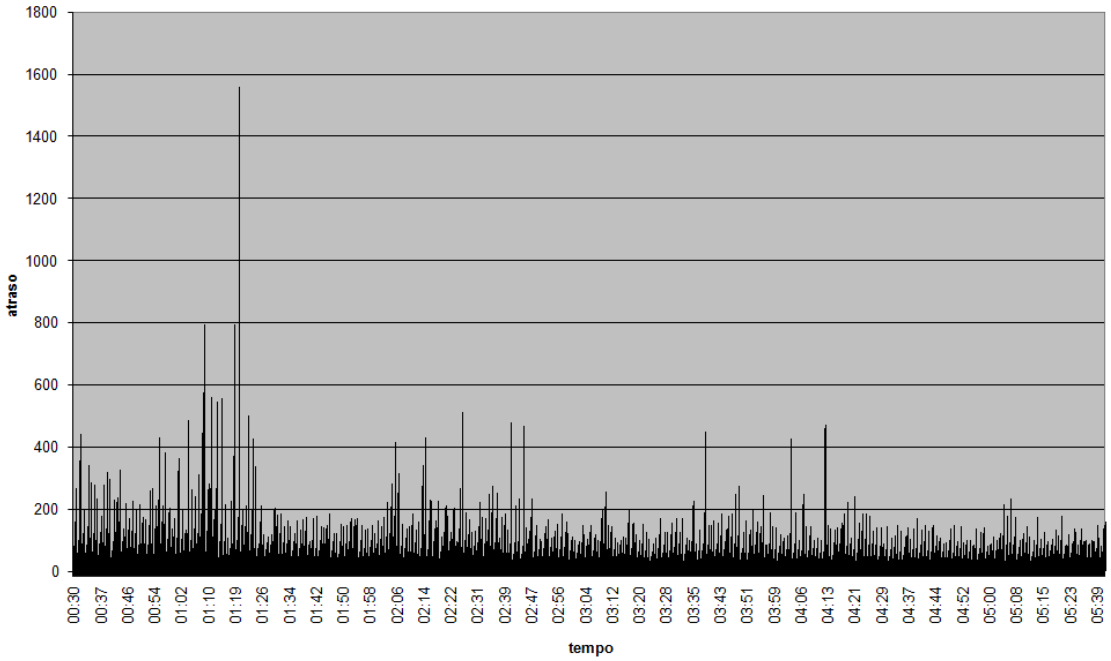

Figura 4.13: Modelo Gossip Filtrado

Na Figura 4.13 verificam-se alguns picos, dos quais os primeiros têm maior intensidade estando os restantes muito abaixo dos 200.

Para obter uma melhor análise dos valores foi construído um gráfico com a probabilidade de atraso das mensagens representado na Figura 4.14.

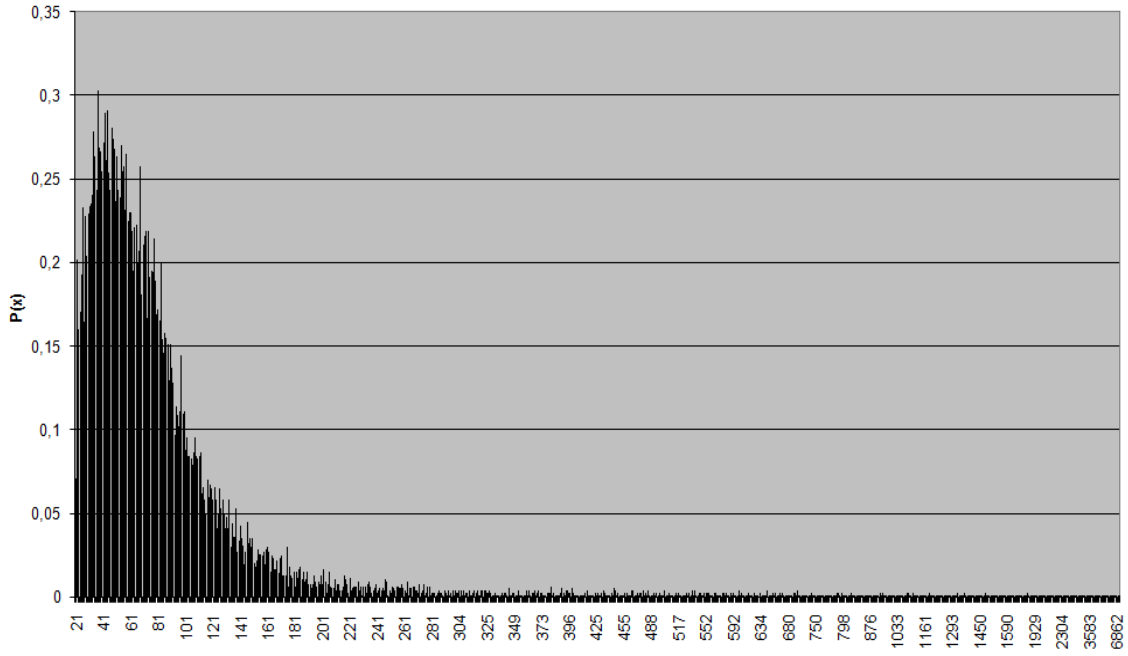

Figura 4.14: Modelo Gossip probabilidade de atraso

Na Figura 4.14 o eixo vertical representa a probabilidade e o horizontal o respectivo atraso. Os maiores valores de probabilidade de atraso encontram-se entre os valores 21 e 101, seguindo-se com valores muito decrescentes. Grande parte das probabilidades encontram-se abaixo 0,25 embora existam algum picos que chegam a ultrapassar os 0.3.

Para perceber qual a relação do atraso relativamente a cada servidor existente, está representado na Figura 4.15 um gráfico com o atraso médio por servidor.

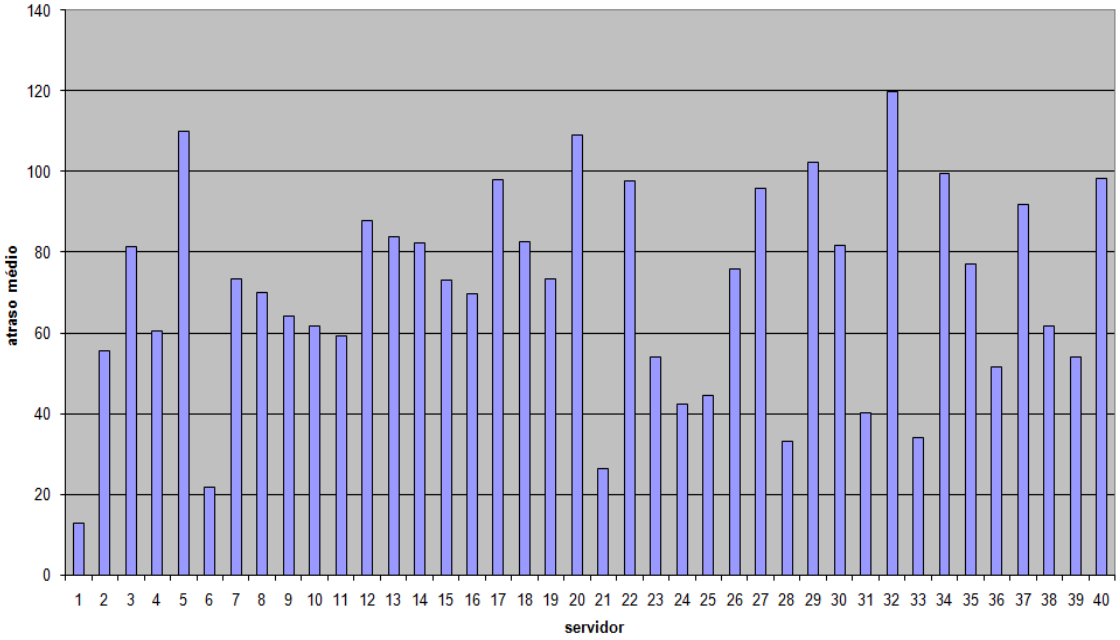

Figura 4.15: Modelo Gossip atraso médio por servidor

Na Figura 4.15 o eixo vertical representa o atraso e o horizontal o servidor onde ocorre a medição. Os servidores estão representados por uma numeração de um a quarenta, onde se pode observar que maior parte dos valores se encontra abaixo de 80 milissegundos, existindo alguns picos onde o máximo chega a atingir os 120.

4.4 Discussão

Os resultados apresentados nos três modelos pelas figuras 4.4, 4.8 e 4.12, apresentam valores iniciais muito elevados pelo que não são considerados para o cálculo de métricas. Estes representam o arranque do sistema e estabilização da rede, interferindo com o comportamento normal do sistema. Relativamente ao arranque, o Java utiliza um compilador JIT que introduz sempre algum atraso.

Nos resultados apresentados do modelo centralizado por notificação, verificam-se alguns picos de atraso elevado (ver Figura 4.5), onde numa amostra de grande dimensão se poderá justificar pela latência de rede ou carga de servidor. Estes valores recolhidos são mais perceptíveis de analisar nas figuras de probabilidade de atraso (Figura 4.6) e atraso médio por servidor (Figura 4.7), onde indicam que a probabilidade de atraso é muito elevada para valores inferiores a 217 milissegundos e o atraso médio por servidor varia entre 11 e 240.

Esta última apresenta valores crescentes pela forma como foi realizada a distribuição de mensagens, pois segundo o modelo, apenas um único servidor é que realiza a distribuição de mensagens sequencialmente para todos os outros servidores. Verifica-se então que o tempo máximo de propagação da mensagem pela rede é de 240, contudo os valores podem ser equilibrados entre os servidores se for utilizado um algoritmo de distribuição mais uniforme.

No modelo centralizado por *polling*, obteve-se valores próximos ao anterior pois o modelo continua a ser centralizado, mas existe uma diferença de carga nos servidores de rede. Este modelo obriga a uma carga muito superior comparativamente ao modelo por notificação, pois todos os servidores não sabem quando é que existem novas mensagens, obrigando a cada um deles questionar com frequência o servidor central. No caso do modelo centralizado por notificação, as mensagens são distribuídas apenas quando necessário obrigando o servidor central a ter uma maior carga, pois será o único a transmitir a informação para todos os restantes na rede.

Em ambos os modelos centralizados todo o tráfego está centralizado num único canal, o do servidor responsável pela distribuição das actualizações. Isto significa que toda a carga está a ser exercida num único ponto, podendo congestionar os canais de comunicação ou o poder de processamento do servidor limitando a escalabilidade do sistema.

No modelo distribuído de Gossip, verifica-se também como nos modelos anteriores alguns picos de atraso muito elevados que poderão ter origem na latência da rede ou carga do servidor (Figura 4.13). Contudo ao observar os valores de probabilidade de atraso (Figura 4.14), verifica-se que maior parte dos valores se encontram abaixo de 100 milissegundos e com maior intensidade por volta dos 40 milissegundos chegando a ultrapassar a probabilidade de 0.3. Neste caso estamos a falar de valores a mais de metade comparativamente com os modelos centralizados. Os valores observados podem ser confirmados com a apresentação da Figura 4.15 que representa o atraso médio por servidor.

Os valores apresentados indicam que no modelo centralizado por notificação, o atraso médio total para garantir que toda a rede está actualizada é aproximadamente 124 milissegundos. No modelo centralizado de Polling é se cerca de 142 milissegundos e no modelo Gossip atinge um valor máximo de 70.

Comparando estes três modelos verifica-se que o modelo de *polling* tem um atraso significativo em relação ao de notificação, e o modelo Gossip é o que obtém valores mais baixos.

O Modelo Gossip apresenta resultados muito superiores relativamente aos restantes modelos, pois o atraso médio de cada servidor é muito inferior aos valores apresentados no modelo centralizado de notificação de Polling.

O modelo de Gossip permite implementar um sistema distribuído, com uma maior eficiência garantindo as necessidades de ambientes distribuídos, onde garante a atomicidade e fiabilidade tal como descrito no capítulo 2.2 segundo os protocolos WS-Coordination,

WS-ReliableMessaging e WS-AtomicTransaction. Este modelo, também utiliza as mesmas propriedades do protocolo WS-Coordination, delegando um coordenador do serviço capaz de registar serviços Atom disponibilizados por outros servidores assim como definir a estrutura de coordenação entre estes. É garantido também, à semelhança do protocolo WS-ReliableMessaging, a entrega de mensagens através da disseminação por Gossip, tal como indicado no capítulo 2.3, mesmo na existência de falha de servidores ou de rede existe uma elevada probabilidade de entrega, assemelhando-se ao tipo de entrega AtLeastOnce do WS-ReliableMessaging, entregando as mensagens pelo menos uma vez podendo existir duplicação. Desta forma consegue-se garantir propriedades de coordenação dos Web Services SOAP mais tradicionais segundo as suas especificações, em Web Services REST através da utilização do Gossip como mecanismo de coordenação.

Resumindo o modelo Gossip consegue garantir atomicidade e fiabilidade na transmissão de mensagens, assim como supera todos os outros modelos na velocidade de transmissão de mensagens com melhorias aproximadas em 44% para o modelo centralizado por notificação e 51% no modelo centralizado por *polling*.

Capítulo 5

Conclusão

Apesar da grande adesão e evolução dos Web Services projectados de acordo com o paradigma REST verifica-se, ao contrário dos Web Services tradicionais, a inexistência de mecanismos genéricos de coordenação. O desafio desta dissertação, centrou-se em estudar a viabilidade de uma coordenação de Gossip em Web Services, avaliando-o através de um protótipo e *benchmarks*.

O mecanismo de coordenação baseou-se na utilização técnicas de Gossip de forma a disseminar a informação num ambiente distribuído, garantindo, à semelhança de protocolos já existentes na coordenação de Web Services tradicionais, a fiabilidade e atomicidade na coordenação dos serviços. Para avaliar este mecanismo, foi idealizada a criação de um serviço distribuído partindo de um modelo centralizado já existente. Isto foi conseguido através da utilização da tecnologia Atom, que segundo o paradigma REST utiliza uma semântica semelhante ao Gossip facilitando a utilização conjunta das três tecnologias.

Os modelos utilizados para a obtenção de métricas, foram divididos em três categorias: modelo centralizado por notificação, modelo centralizado por polling e um modelo distribuído com Gossip. Os modelos centralizados distinguem-se pela forma como é realizada a consulta de informação. No modelo por notificação existe um servidor que é responsável pelo envio de novas actualizações para todos os restantes. No modelo de polling são os restantes servidores que consultam com frequência o servidor para verificar a existência de nova informação. No modelo distribuído, são utilizadas as metodologias de Gossip que permitem que vários servidores se coordenem entre si, comunicando através de grupos de *membership* não havendo pontos centralizados de informação.

Com base nos tempos de atraso na disseminação de informação para todos os servidores existentes na rede, pode-se concluir que o modelo distribuído supera largamente os modelos centralizados, ao atingir performances de resposta muito elevadas com ganhos de cerca de 44% para o modelo centralizado por notificação e 51% no modelo centralizado por *polling*, para a propagação de informação em todos os servidores da rede.

Tendo em consideração estes resultados, verifica-se que a coordenação de RESTful Web Services com Gossip apresenta viabilidade, mantendo a simplicidade, eficiência e fiabilidade que caracteriza estas tecnologias.

5.1 Limitações deste trabalho e sugestões futuras

O trabalho apresentado embora concluído com sucesso, esteve limitado relativamente à quantidade de servidores disponíveis, pois seria interessante tentar reproduzir o mesmo sistema em grande escala com centenas ou milhares de instâncias. Deste modo, seria possível comprovar se se mantém o mesmo comportamento, com elevada performance, num universo amostral de maior dimensão, o que se traduz em resultados mais significativos.

Era também de todo interessante fazer uma comparação dos modelos centralizados com todos os métodos de Gossip indicados, permitindo indicar qual o melhor desempenho mediante as necessidades do modelo distribuído.

A aposta nesta área de investigação, poderá no futuro viabilizar a criação de protocolos e especificações de novas alternativas de coordenação de Web Services.

Bibliografia

[ACKM03] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, Berlin, Germany, October 2003.

[Cer02] E. Cerami. *Web Services Essentials*. O'Reilly & Associates, 2002.

[CMRW07] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0, 2007. `http://www.w3.org/TR/wsdl20`.

[EGKM04] P.T Eugster, R. Guerraoui, A.M Kermarrec, and L. Massoulie. Epidemic Information Dissemination in Distributed Systems. *IEEE Computer Society*, 2004.

[Fie00] R.T Fielding. *Architectural Styles andthe Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[FL05] C. Ferris and D. Langworthy. Web Services Reliable Messaging Protocol (WS-ReliableMessaging). IBM, 2005. http://www.ibm.com/developerworks/library/specification/ws-rm/.

[Fou10a] Apache Software Foundation. An Open Source Atom Implementation, 2010. `http://abdera.apache.org`.

[Fou10b] Eclipse Foundation. Jetty, 2010. `http://jetty.codehaus.org/jetty`.

[GKG02] I. Gupta, A.M. Kermarrec, and A.J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *Proceedings of the 21st Symposium on Reliable Distributed Systems*, 2002.

[Haa05] H. Haas. Foundations And Future Directions Of Web Services. 2005. http://www.w3.org/2005/Talks/0511-hh-www2005/slide9-0.html.

[HSAA03] J. Holliday, R. Steinke, D. Agrawal, and A. Abbadi. Epidemic algorithms for replicated databases. *Knowledge and Data Engineering, IEEE Transactions on*, 15:1218 – 1238, 2003.

[HSAA05] J. Holliday, R. Steinke, D. Agrawal, and A. Abbadi. Atom: The Standard in Syndication. *Internet Computing, IEEE*, 9:71 – 78, 2005.

[KMG03] A.M. Kermarrec, L. Massoulié, and A.J. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14:248–258, 2003.

[KS07] A.M. Kermarrec and M. Steen. Gossiping in Distributed Systems. *IEEE Computer Society*, 2007.

[KSSV00] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *In IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[LEH03] J. Luo, P.T. Eugster, and J.P. Hubaux. Route driven gossip: Probabilistic reliable multicast in ad hoc networks. In *IN PROC. OF INFOCOM*, pages 2229–2239, 2003.

[LM] M. Lin and K. Marzullo. Directional Gossip: Gossip in a Wide Area Network. In *In European Dependable Computing Conference*, pages 364–379.

[LW09] M. Little and A. Wilkinson. Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2, 2009. `http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os/wstx-wsat-1.2-spec-os.html`.

[MH08] P. Sandoz M. Hadley. JSR 311: JAX-RS: The JavaTM API for RESTful Web Services, 2008.

[Ogb00] U. Ogbuji. Using WSDL in SOAP applications, 2000. `http://www.ibm.com/developerworks/library/ws-soap/?dwzone=ws`.

[PGC06] J.A. Patel, I. Gupta, and N. Contractor. Jetstream: Achieving predictable gossip dissemination by leveraging social network principles. In *In NCA Š06: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications*, pages 32–39. IEEE Computer Society, 2006.

[RR07] L. Richardson and S. Ruby. *RESTful Web Services*. 2007.

[S.A06] Skype Technologies S.A. Skype Network Administrator's Guide, 2006. `http://www.pamconsult.com/faq/attachments/9/guide-for-network-admins-30b.pdf`.

[Til07] S. Tilkov. A Brief Introduction to REST, 2007. `http://www.infoq.com/articles/rest-introduction`.

[W3C01] W3C. Web Services Description Language, 2001. `http://www.w3.org/TR/wsdl`.

[W3C04] W3C. Web Services Architecture, 2004. http://www.w3.org/TR/ws-arch.

[W3C07] W3C. SOAP Version 1.2 Part 1: Messaging Framework, 2007. http://www.w3.org/TR/soap12-part1.

[W3C10] W3C. Extensible Markup Language (XML), 2010. http://www.w3.org/XML.

[WWC09] Web Services Coordination (WS-Coordination) Version 1.2, 2009. http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os/wstx-wscoor-1.2-spec-os.html.