Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Comanche: Aplicações Web Estendidas nos clientes

Por

Filipe Miguel Afonso

Nº 26103

Dissertação apresentada na Faculdade de Ciências e Tecnologia

da Universidade Nova de Lisboa para

obtenção do grau de Mestre em Engenharia Informática

Orientador

Prof. Doutor Nuno Preguiça

Monte da Caparica

2009

Agradecimentos

Quero agradecer a todos aqueles que possibilitaram, de uma forma ou outra, que este projecto fosse concluído.

Queria começar por agradecer ao meu orientador, Professor Doutor Nuno Preguiça. O seu apoio e dedicação durante todas as fases da elaboração desta dissertação foram fundamentais.

Deixo um agradecimento especial a todos os meus colegas do gabinete 254 do edifício E2 da FCT.

Não quero deixar de agradecer aos amigos que fiz durante o curso e que fizeram rir quando mais precisava. Nomeadamente ao André Rosa, Daniel Ramos, Pedro Boavida, Marco Teixeira, Gustavo Marques, André Simões, Pedro Lobo e Joana Dias.

Quero também agradecer a todos os amigos que me foram encorajando a acabar o curso. Especialmente à Ana Cristina Silva, Ricardo Noguês e Carlos Filipe.

Agradeço especialmente ao meu pai, Amado Pereira Afonso, à minha mãe, Maria dos Anjos Afonso, e à minha mana, Ana Rita Afonso, que tiveram a paciência de me aturar e sempre me deram coragem para continuar.

Resumo

A Internet é uma rede de comunicações que permite a troca de informação entre computadores localizados em qualquer ponto do mundo. Vários serviços foram desenvolvidos utilizando esta rede de comunicações. Entre estes, encontra-se a Web, que permite a qualquer utilizador aceder, utilizando o protocolo HTTP, a documentos hipermédia identificados através de um URL.

Com a popularidade da Web, surgiram várias propostas para melhorar o desempenho global da Web e permitir o desenvolvimento de aplicações complexas neste ambiente. Entre estas propostas encontram-se, por exemplo, novas versões do protocolo HTTP, sistemas de *caching* e replicação e sistemas de suporte à operação desconectada.

Neste trabalho apresenta-se o desenho e implementação do sistema Comanche, que permite tratar pedidos HTTP na máquina do cliente, possibilitando a adição de novas funcionalidades sem alteração dos servidores. Esta aproximação permite ainda minimizar o número de pedidos que têm de ser tratados pelos servidores, diminuindo assim a quantidade de pedidos processados e o tráfego na rede.

O Comanche é implementado como uma proxy de HTTP, que se encontra instalada na máquina do utilizador e consegue instalar e executar *Java Web Applications*. Estas aplicações permitem a criação local de respostas a pedidos HTTP. Estas aplicações podem também ser usadas para se adicionar funcionalidades extra, alterando as respostas ou recolhendo informação de valor para o utilizador, que pode ser posteriormente usada para fornecer outros serviços. O sistema permite combinar aplicações de forma a permitir a reutilização das mesmas, formando aplicações mais complexas, e disponibiliza uma funcionalidade de sincronização de recursos acessíveis por HTTP.

Abstract

The Internet is a communications network that allows the exchange of information between computers situated anywhere in the world. Several services have been developed using this communications network. One of these services is the Web that allows a computer to access, using the HTTP protocol, to hypermedia documents, identified by an URL.

With the popularity of the Web, there were several proposals to improve the overall performance of the Web and allow the development of complex applications in this environment. Among these proposals are, for example, new versions of the HTTP protocol, caching and replication systems and systems support disconnected operation.

This work presents the design and implementation of the Comanche, which can handle HTTP requests on the client machine, allowing the addition of new features without changing the servers. This approach also helps to minimize the number of requests to be handled by the servers, thereby reducing the number of applications processed and traffic on the network.

Comanche is implemented as a HTTP proxy, which is installed on the user's machine and can install and run Java Web Applications. These applications allow the creation of responses to requests HTTP locally. These applications can also be used to add extra functionalities, changing answers or collecting information of value to the user, which can later be used to provide other services. The system can combine applications, forming more complex applications, and offers a feature of synchronization of resources accessible by HTTP.

Índice

Índice de Figuras e Tabelas

1. Introdução

1.1 Contexto

A Internet é uma rede de comunicações que permite a troca de informação entre computadores localizados em qualquer ponto do mundo. Desde a sua criação, múltiplos serviços com diferentes funcionalidades foram desenvolvidos utilizando esta rede de comunicações.

Uma das principais funcionalidades tem sido a criação de sistemas de comunicação, entre os quais o sistema de correio electrónico merece especial realce. Mais recentemente, foram desenvolvidos e generalizaram-se os serviços de comunicação de voz (VoIP, ou Voice over Internet Protocol) e serviços de video-conferência, dos quais o Skype é apenas um exemplo.

Outro serviço que rapidamente ganhou uma grande popularidade, sendo hoje, provavelmente, o serviço mais utilizado, foi a Web. Este serviço permite a qualquer utilizador aceder, utilizando o protocolo HTTP, a documentos hipermédia identificados através de um URL.

Com a evolução da Web a informação disponibilizada por este sistema foi-se tornando mais complexa, consistindo não apenas em documentos estáticos mas também em documentos gerados dinamicamente. Isto permitiu a implementação de múltiplas aplicações, entre as quais lojas online, serviços de notícias, etc.

A vantagem da utilização, para os utilizadores finais, de serviços na Web é a facilidade de acesso aos mesmos. Assim, os utilizadores não têm de se deslocar fisicamente podendo aceder aos serviços em qualquer lugar e em qualquer momento.

O desempenho da Web depende da eficiência dos protocolos usados, do tráfego gerado pelos utilizadores e da carga dos servidores. Relativamente às comunicações, as versões mais recentes dos protocolos usados na Web são bastante eficientes, podendo-se ainda melhorar o seu desempenho aumentando a largura de banda disponível nos canais de comunicação. No entanto, a latência das comunicações e o elevado número de cliente pode levar a que o tempo de acesso a um serviço Web seja superior ao desejável.

Dado que existem limites físicos à latência das comunicações, uma possibilidade de melhorar o desempenho é servir os pedidos mais próximo dos clientes, reduzindo ainda o tráfego gerado na rede e o número de pedidos recebido pelos servidores.

1.2 Motivação

Os recursos computacionais dos clientes têm aumentado de forma significativa ao longo dos anos. Actualmente é comum os computadores para uso pessoal terem capacidades semelhantes a alguns servidores Web, tanto a nível de processamento como de capacidade de armazenamento. Por outro lado, a latência na Internet não tem sido reduzida de forma significativa, visto que existem limites físicos e a configuração da rede, com os seus múltiplos routers, impõe atrasos na propagação das comunicações. Assim, seria muitas vezes mais rápido executar um pedido localmente, mesmo que isso envolvesse processamento, que esperar pela resposta do servidor.

O processamento local poderia ainda permitir criar serviços complexos sem recurso aos servidores. Esta propriedade pode ser importante porque é possível que certos serviços não sejam comercialmente rentáveis, dificultando a sua implementação num servidor. Adicionalmente, quando estes serviços utilizam informação pessoal dos utilizadores, a sua execução nos clientes elimina os problemas de segurança e privacidade envolvidos.

O objectivo deste trabalho é melhorar a experiência dos clientes no acesso à Web. Para tal propõe-se um sistema, baseado em proxies instaladas nos clientes, que permite através da execução de aplicações nos clientes, aumentar o desempenho de acesso aos serviços Web e possibilitar a adição de novas funcionalidades.

1.3 Solução apresentada

O sistema desenvolvido é baseado numa *proxy* HTTP, instalada nas máquinas dos clientes, que irá interceptar e tratar os pedidos localmente, sempre que possível. Esta *proxy* tem a capacidade de encaminhar os pedidos para servidores remotos ou proceder ao seu processamento localmente. Para se tratar dos pedidos, sem recorrer ao seu encaminhamento para um servidor, é necessário que o sistema possua aplicações capazes de tratar o pedido. Para tal, o sistema permite a instalação dinâmica de novas aplicações a partir dos servidores. Estas aplicações podem ser de três tipos: aplicações finais, aplicações de observação e aplicações de transformação.

As aplicações finais destinam-se a processar um pedido localmente, gerando uma resposta ao mesmo. Esta resposta pode ser uma página estática ou uma página dinâmica gerada no momento a partir de informação local ou obtida remotamente. Por exemplo, uma aplicação final pode ser usada para implementar uma loja *online*, em que o cliente localmente selecciona os produtos desejados. Apenas no momento da conclusão da compra seria necessário contactar o servidor.

As aplicações de observação permitem recolher informação dos pedidos efectuados pelos utilizadores e das suas respostas. Por exemplo, uma aplicação de observação pode ser usada para analisar quais os sites e conteúdos que o utilizador mais se interessa. Esta informação pode ser usada posteriormente por uma aplicação final para sugerir novos sites com informação semelhante.

As aplicações de transformação têm como objectivo permitir a alteração dos pedidos e das respostas, sendo aplicadas também aos pedidos processados por aplicações finais. Por exemplo, uma aplicação de transformação pode ser usada para filtrar o conteúdo ou, com base na informação recolhida por uma aplicação de observação, adicionar *links* com informação relacionada com a que o utilizador está a consultar.

Para que as aplicações possam guardar informação, o sistema gere um repositório com informação das aplicações. Associado a cada repositório existe um serviço de sincronização que permite manter uma cópia local actualizada de um recurso Web. As aplicações podem solicitar a sincronização dos recursos que são relevantes para as mesmas. A combinação destas funcionalidades permite, ainda, que um utilizador possa continuar a aceder aos serviços instalados localmente mesmo em situações de desconexão.

1.4 Principais contribuições

A principal contribuição deste trabalho é o desenho e implementação dum sistema que permite estender as aplicações Web através da execução de código num *proxy* a executar na máquina do cliente. O objectivo desta solução é providenciar uma melhor experiência ao utilizador, aproveitando as capacidades das máquinas usadas no acesso à Internet, com uma eventual redução de pedidos que chegam aos servidores. Este facto pode permitir aumentar a escalabilidade dos mesmos sem que seja necessário melhorar o seu hardware ou a ligação à Internet disponível. Adicionalmente, a execução local dos pedidos pode melhorar a latência de resposta aos mesmos.

A proposta apresentada nesta dissertação apresenta uma solução que permite fornecer funcionalidades adicionais junto dos clientes, podendo tornar mais fácil a sua adopção generalizada (ao contrário de soluções baseadas em servidores poderosos ou em *Content Distribution Networks* que apenas estão ao alcance de alguns serviços). Por exemplo, a solução apresentada permite criar aplicações que forneçam numa só página Web a informação mais relevante obtida a partir de vários sites Web, implementando o conceito de *mashup* [16] próximo do cliente. Esta aproximação tem a vantagem de permitir esta funcionalidade sem a necessidade dum servidor Web. Quando comparadas com outras soluções, que permitem executar código de aplicações Web nos clientes, a solução apresentada exibe uma diferença importante: permite criar aplicações que interceptam todos os pedidos efectuados. Como se discutiu anteriormente, esta aproximação permite criar serviços adicionais que não são possíveis nas outras soluções.

A sincronização automática de forma transparente dos dados das aplicações é outra contribuição deste trabalho. As vantagens deste mecanismo são claras: se todos os dados necessários estiverem válidos localmente, o pedido será tratado quase instantaneamente e sem que seja necessário contactar eventuais servidores. Esta aproximação permite ainda, conjugada com a execução local das aplicações, responder a pedidos em situações de desconexão.

1.5 Organização

Após este capítulo em que se fez a apresentação do contexto deste trabalho e se descreveu de forma breve a solução implementada, este documento prossegue da seguinte forma. No capítulo 2 apresenta-se um resumo do estado da arte, organizado em duas secções que descrevem soluções para melhorar o desempenho de acesso à Web e a adição de novas funcionalidades, como o suporte para acesso durante períodos de desconexão. No capítulo 3 apresenta-se o desenho do sistema, descrevendo-se a arquitectura e as tecnologias usadas para o seu desenvolvimento. No capítulo 4 são discutidos os principais pormenores da implementação do sistema. O capítulo 5 apresenta a avaliação do sistema, descrevendo-se as aplicações desenvolvidas e os resultados de performance. Finalmente, no capítulo 6 apresentam-se as conclusões da dissertação.

2. Trabalho relacionado

A World Wide Web (ou simplesmente Web) é um sistema de hipermédia que permite o acesso a documentos e aplicações através da Internet. A utilização e importância deste sistema tem aumentado continuamente, desde a sua criação, podendo dizer-se que a sua utilização se tornou ubíqua, o que se reflecte não só no número de utilizadores, ver figura 2.1, mas também na sua importância para a economia. Desta forma é importante que o acesso a este sistema seja o mais eficiente possível. Adicionalmente, é importante que as funcionalidades que se podem implementar sejam o mais amplas possível. Neste capítulo apresentam-se várias aproximações propostas e em funcionamento que permitem o bom funcionamento da Web e a implementação de múltiplas funcionalidades.

2.1 Protocolo HTTP e extensões

2.1.1 Protocolo base

O protocolo HTTP, Hypertext Transfer Protocol, define a forma como os clientes e servidores interagem através da troca de pedidos e respectivas respostas. Este protocolo é a base do sistema Web. O protocolo permite transferir recursos entre servidores e clientes. Estes recursos podem ser, por exemplo, páginas estáticas, imagens e conteúdos dinamicamente gerados. Os recursos são identificados por um identificador único universal, mais especificamente um URL.

Na versão 1.0 do protocolo HTTP, RFC 1945 [9], por cada pedido de um recurso era necessário abrir uma ligação para o servidor. A abertura e encerramento dos canais TCP/IP consomem recursos a nível de processamento, largura de banda e memória. Adicionalmente a latência da abertura de uma conexão para uma transferência de pequena dimensão também não é desprezável. Estes factos começaram a ser significativos na utilização e desempenho do protocolo, o que levou ao desenvolvimento de uma nova versão que reutilizasse os canais abertos para vários pedidos sucessivos.

A versão 1.l do protocolo HTTP, RFC 2616 [10], introduz, entre outras funcionalidades, a possibilidade de reutilização das conexões já estabelecidas para serem feitos novos pedidos. Com as páginas Web a terem cada vez mais imagens e outros objectos *Web*, o desempenho e escalabilidade dos servidores *Web* aumentou. No entanto, o aumento massivo da utilização da *Web* durante o final do século XX e início do século XXI, figura 2.1, fez com que esta melhoria não fosse suficiente, sendo necessário melhorar o desempenho e escalabilidade da *Web* recorrendo a outros meios.

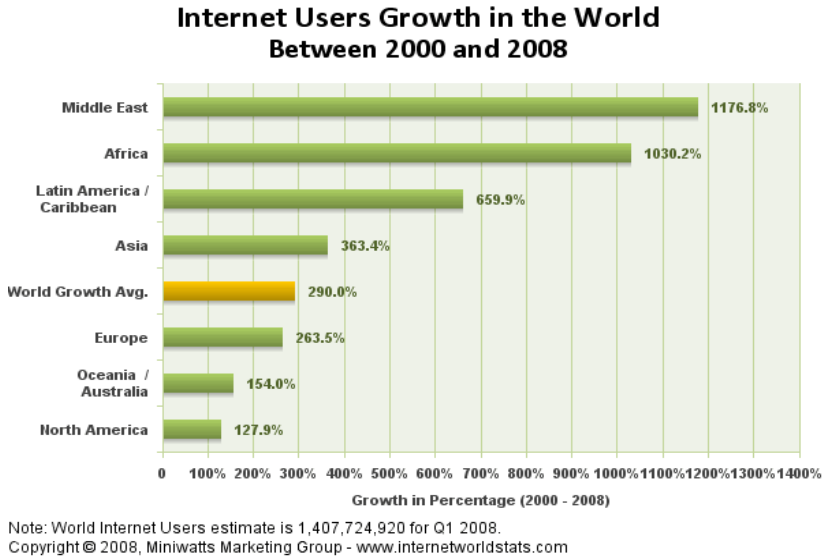

Figura 2.1 – Crescimento do número de utilizadores entre 2000 e 2008 [8]

2.1.2 Suporte base para caching e replicação

A existência de caches e *proxies* na Internet permite que se aumente a escalabilidade da mesma. Se um pedido puder ser respondido por uma entidade que se encontre mais próxima do cliente, o tempo de resposta irá ser menor e o servidor que deveria responder fica com menos pedidos para atender. No entanto, o *caching* e replicação pode levar a problemas de consistência.

Para permitir *caching* e replicação de conteúdos, o HTTP disponibiliza métodos que permitem validar as réplicas. Uma funcionalidade central para o *caching* e replicação é o suporte para pedidos condicionais [1]. Um servidor, ao processar um pedido condicional, responde normalmente se a condição for válida. Caso contrário, responde com um código especial indicando que a resposta enviada anteriormente ainda se encontra válida ou que a condição não pode ser resolvida. Os clientes especificam as condições do pedido usando meta-informação dos objectos, enviada previamente pelos servidores, nomeadamente a informação relativa aos

cabeçalhos *last-modified* e *Etag*. O cabeçalho *last-modified* contém a data, com precisão até aos segundos, da última modificação do objecto. O cabeçalho *Etag* é um identificador único para uma instância de um objecto. Por exemplo, este valor pode ser calculado aplicando uma função de síntese segura ao conteúdo do objecto. Ao guardar uma cópia dos dados os clientes guardam estas informações que são usadas nos pedidos condicionais como se explica de seguida.

Cabeçalhos HTTP usados para caching

Os cabeçalhos condicionais mais relevantes para *caching* de conteúdos são *if-modified-since* e *if-none-match*.

O cabeçalho *if-modified-since* do pedido contém o valor do cabeçalho *last-modified* da cópia do recurso mantida em cache. O servidor, ao receber um pedido com este cabeçalho irá retornar um novo objecto se o valor do cabeçalho *last-modified* do objecto no servidor for diferente do valor no cabeçalho *if-modified-since* do pedido. Caso sejam iguais responde com o código 304, indicando que não houve alterações. Neste caso, o pedido pode ser respondido com o valor que se encontra em cache.

O cabeçalho *if-none-match* contém o valor da *Etag*. Um servidor ao receber um pedido com o cabeçalho *if-none-match* irá comparar este valor ao valor da *Etag* do objecto no servidor. Se forem iguais responde com o código 304. Caso contrário responde com um novo objecto.

Embora o cabeçalho *Etag* ofereça um melhor controlo sobre os objectos, pois *o last-modified* não consegue distinguir alterações feitas durante um segundo, os pedidos condicionais são maioritariamente feitos usando *if-modified-since* pois foi introduzido em versões mais antigas do HTTP.

A disponibilidade dos recursos HTTP pode ser melhorada recorrendo à replicação dos mesmos entre servidores. O HTTP contém métodos, *POST*, *PUT* e *DELETE*, que permitem a manipulação de objectos entre entidades. O método *POST* é usado para enviar alterações de um objecto ao servidor. O método *PUT* para enviar um novo objecto e o método *DELETE* para remover um objecto. Por exemplo, um servidor A pode inserir, modificar e apagar objectos num servidor B. No entanto, operações que provoquem a alteração de uma réplica podem provocar inconsistências entre as várias réplicas do objecto. Por exemplo, consideremos dois clientes, CA e CB, e um objecto, contendo um número, que se encontra replicado nos servidores A e B. Inicialmente ambas as réplicas contêm o mesmo número. Se CA alterar o valor em A para 1 e CB

alterar o valor em B para 2, ambas as réplicas possuem agora valores distintos. Se A fizer *POST* em B, a alteração do cliente CB será perdida.

Cabeçalhos HTTP usados para replicação

De forma a garantir que as alterações são feitas apenas nas versões dos objectos aos quais se destinam, podem ser usados dois cabeçalhos condicionais: *if-unmofidied-since* e *if-match*. O cabeçalho *if-unmodified-since* é o inverso do *if-modified-since*. O pedido apenas é executado se ambos os valores forem iguais. O cabeçalho *if-match* contém o valor do *Etag*. Novamente, um pedido apenas é executado se ambos os valores forem iguais. Se, em ambos os casos, os valores forem diferentes é devolvido o código de erro 402, *Precondition Failed*.

Validação de conteúdos

O HTTP também fornece suporte para os sistemas de cache decidirem quando validar os objectos que se encontram guardados na cache, o qual é baseado em um período de validade, *time to live* (TTL). O TTL de um objecto é o tempo durante o qual ele pode ser considerado válido numa cache. Um objecto que se encontre em cache há mais tempo que o permitido pelo seu TTL é considerado expirado e não pode ser fornecido como resposta a um pedido, sendo necessário que a cache o valide usando um pedido condicional ao servidor de onde o objecto originou.

Um servidor HTTP pode indicar explicitamente o tempo de vida de um objecto (TTL) usando os cabeçalhos *expires* e *cache-control*. O cabeçalho *expires* indica a data até à qual o objecto é considerado válido. O cabeçalho *cache-control* controla, por meio de directivas, o comportamento das caches que se encontram entre o servidor e o cliente. Este cabeçalho pode ser utilizado tanto em pedidos como em respostas.

Directivas usadas no pedido HTTP		
Directiva	**Valor**	**Significado**
no-cache	nenhum	Objectos em cache não podem ser usados para satisfazer este pedido
no-store	nenhum	Qualquer informação, seja do pedido ou da resposta ao mesmo, não pode ser guardada em cache
max-age	segundos	Apenas objectos mais recentes que o valor podem ser usados para satisfazer o pedido
min-fresh	segundos	Apenas objectos que não expirem antes do tempo especificado podes ser usados
max-stale	segundos	Objectos em cache já expirados até ao tempo especificado podem ser usados
no-transform	nenhum	Apenas as respostas sem alterações fornecidas pelo servidor de origem podem ser usadas.
only-if-cached	nenhum	Uma *proxy* não deverá encaminhar o pedido se o objecto não se encontrar em cache

Directivas usadas nas respostas HTTP		
Directiva	**Valor**	**Significado**
no-cache	nenhum	A resposta não pode ser guardada em cache
no-store	nenhum	A resposta não pode ser guardada em nenhum cliente (*proxies ou browsers*)
private	nenhum	A resposta pode ser reutilizada mas apenas para o cliente que fez o pedido
public	nenhum	A resposta pode ser guardada em cache e partilhada por diferentes clientes
must-revalidate	nenhum	A cache tem de revalidar o objecto antes de o usar
proxy-revalidation	nenhum	A cache tem de revalidar o objecto antes de o usar mas só se for uma *proxy*
max-age	segundos	A cache terá de revalidar o objecto antes de o usar para responder a um pedido após o objecto atingir a idade especificada
s-maxage	segundos	A regra é igual à *max-age* mas só se aplica a proxies
no-transform	nenhum	Apenas a resposta fornecida pelo servidor, sem alterações, pode ser usada

Tabela 2.1 – directivas do *cache-control*

A utilização deste cabeçalho permite múltiplas variantes. Na tabela 2.1 apresenta-se a lista completa de directivas que podem ser usadas. Por exemplo, o cabeçalho

Cache-control:max-age=120, no-transform

que se encontra num pedido, especifica que a resposta pode ser dada por uma cache que se encontre entre o cliente e o servidor mas que o objecto não pode ter sido modificado e que a sua idade máxima é de 120 segundos.

2.1.3 Proxy Caching

A utilização de proxies tende a ser generalizada, porque a mesma pode levar a reduções tanto ao nível da largura de banda como da latência. Por exemplo, um ISP pode beneficiar de uma *proxy* de ambas as maneiras. Um pedido que seja respondido por uma *proxy* antes do acesso ao *backbone* faz com que esse pedido não seja tratado pelo *backbone*, poupando largura de banda ao ISP. Por outro lado, como a resposta é dada por um sistema que se encontra mais perto do cliente, a latência associada ao pedido será menor. Como a largura de banda necessária para um ISP depende do tráfego gerado pelos clientes, e a *Web* representa uma fracção cada vez mais significativa deste tráfego, a redução do tráfego Web pode ter um impacto importante no funcionamento dos ISP's.

A eficiência de uma *proxy* cache pode ser medida em número de pedidos que são satisfeitos pela *proxy* a dividir pelo número total de pedidos, chamado *hit rate.* Ao ser feito um pedido pelo browser, se existir uma cache pelo meio e outro utilizador já tiver feito o mesmo pedido, assumindo que o objecto contínua válido, esta irá responder ao pedido. Análises a várias proxies mostraram que, com um número aceitável de utilizadores (alguns milhares), estas podem satisfazer, em média, 40% dos pedidos [11].

2.1.4 Caching de Conteúdos dinâmicos

Estudos efectuados [11] mostraram que a utilidade de *proxy caching* está limitada pelo valor de *hit rates*, que é cerca de 40%. A principal razão para que o *hit ratio* não seja superior é que perto de 43% dos pedidos *Web* são para conteúdos que não podem ser guardados. Logo, apenas 57% das respostas podem ser reutilizadas para outros pedidos. Para estes valores existem várias causas [1].

O mau uso de cookies é uma das razões. O HTTP permite a definição de novos cabeçalhos desde que sigam a estrutura "nome:valor". Os *cookies*, que não fazem parte do standard HTTP,

são uma das inovações que permite a criação de sessões entre o cliente e o servidor, permitindo ultrapassar o problema do protocolo HTTP ser *stateless*. Logo, a maioria das caches não guarda os objectos associados a cookies pois assume que estes conteúdos são específicos para cada utilizador e que não os pode usar para responder a outros pedidos. Estudos mostram que, na sua maioria, as respostas são idênticas. Isto é especialmente verdade para imagens. Existem proxies que têm como opção fazer *caching* de imagens que estejam associadas a *cookies*. No entanto, estudos [18] mostraram que, neste caso, em 13% das respostas, as imagens são diferentes levando a respostas erradas por parte da *proxy*. Uma solução possível é encaminhar para o servidor de origem o pedido mas indicando qual o conteúdo que se encontra em cache, através de um pedido condicional usando *Etag*. Desta forma, se algum dos conteúdos puder ser reutilizado, não será necessário o servidor enviar o conteúdo da resposta.

Outra causa que leva a uma baixa taxa de reutilização dos dados em cache é a cada vez maior utilização de páginas dinâmicas. Neste caso existem várias soluções que podem ser utilizadas para minimizar a transferência de dados. Uma possibilidade é ter, para páginas dinâmicas, uma base estática. Cada objecto dinâmico é transferido a pedido e a página completada no browser. No entanto, a existência de muitos objectos de pequena dimensão podem levar à sobrecarga do servidor, o que é inaceitável. Por outro lado, o encapsulamento de vários conteúdos dinâmicos num só objecto pode levar a que seja necessário transferir uma grande fracção da página, o que derrota o objectivo da reutilização.

Outra possibilidade é o cálculo e envio das diferenças entre uma versão anterior do objecto e a versão actual. Se o cliente e a *proxy* partilharem uma versão antiga, a *proxy* pode calcular as diferenças e enviá-las ao cliente. De notar que o único ganho é na quantidade de informação que se tem de enviar ao cliente a partir da *proxy*.

De seguida apresentam-se outras soluções usadas para lidar com conteúdos dinâmicos.

2.1.4.1 Base-instance Caching

Para diminuir o tráfego gerado pelas páginas dinâmicas podem-se enviar as diferenças entre versões caso o cliente e o servidor partilhem uma mesma versão base da página. Esta técnica chamada *Base-instance Caching* [1] pode ser facilmente implementada usando *proxies* modificadas em ambos os extremos, uma próxima do cliente e outra do servidor. As diferenças podem ser enviadas numa só ligação e o cliente reconstrói a página após a transferência. Adicionalmente, podem ser aplicadas técnicas de compressão para tornar o conteúdo mais

pequeno. Esta solução tem o problema que o servidor e ou a *proxy* próxima do servidor tem de guardar a versão base para cada cliente, o que pode ser inaceitável se existirem muitos clientes. No entanto é transparente para os criadores de páginas *Web*.

Esta solução é igualmente usada para minimizar o tráfego transferido em sistemas que permitem a visualização de conteúdos em ambientes de computação móvel, como por exemplo, o sistema WebExpress [20].

2.1.4.2 Template Caching

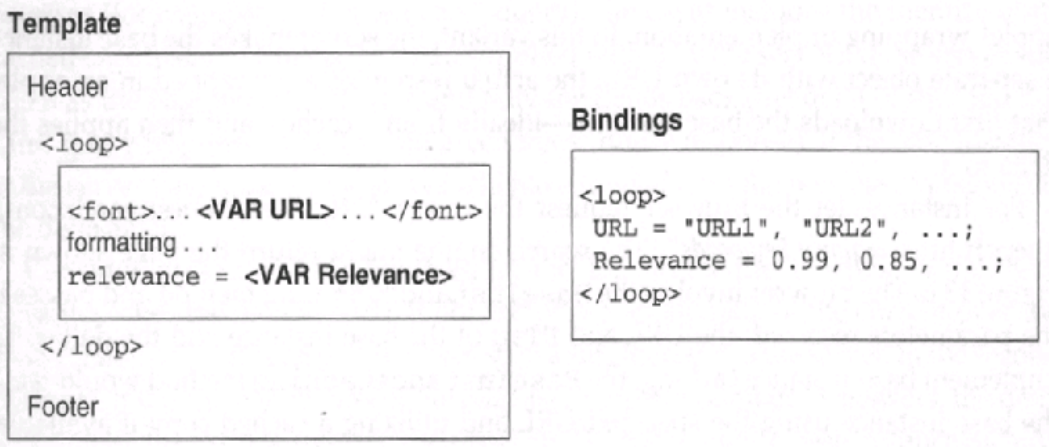

Figura 2.2 – Template Caching [1]

Outra técnica possível para lidar com conteúdos dinâmicos é o *Template Caching* [1,21]. No primeiro acesso, o servidor envia ao cliente a resposta e um *template* da página. Este *template* é constituído pela parte estática da página, *Template*, e instruções que se referem ao conteúdo dinâmico da mesma, os *Bindings*, como se pode ver na figura 2.2. No segundo acesso o servidor apenas tem de enviar o conteúdo dinâmico, tendo o cliente que reconstruir a página seguindo as instruções do template. A vantagem desta aproximação é que, mesmo que o conteúdo dinâmico mude radicalmente, é simples reconstruir a página, bastando aplicar os *bindings* ao *template*. E estas estruturas podem ser computadas antes de haver qualquer acesso, não sendo gasto tempo em processamento durante o tratamento do pedido do cliente.

2.1.5 Soluções de replicação

Já foram mencionados os métodos que o protocolo HTTP disponibiliza para replicação de recursos. Nesta secção são apresentadas algumas soluções de replicação na Internet. As *Content Distribution Networks*, CDN, são sistemas de computadores distribuídos na Internet que cooperam para fornecer um dado conteúdo, em geral conteúdos relativos a páginas Web. O objectivo destes sistemas é aumentar a disponibilidade dos recursos e o desempenho dos servidores mantendo uma total transparência para os clientes.

Uma CDN é composta por um conjunto de repositórios de conteúdos para fornecer um melhor serviço aos utilizadores finais. Um servidor Web, *CDN customer*, regista os conteúdos numa CDN. Estes conteúdos serão replicados pelos repositórios dessa CDN. Estes repositórios encontram-se geralmente localizados dentro de ISP's e o objectivo é responder aos pedidos dos clientes o mais cedo possível, evitando congestionamento nos *backbones* e no servidor Web original. Ao contrário das cache proxies, que tentam aumentar a disponibilidade dos recursos ignorando a sua origem, um CDN apenas fornece os recursos de servidores que aderem ao seu serviço, estabelecendo uma relação de negócio entre ambos. As CDN possuem também algoritmos de encaminhamento complexos, tentando distribuir a carga entre os vários servidores e oferecer ao cliente a melhor qualidade de serviço possível.

Existem vários tipos de CDN's, dependendo das suas características. Um primeiro aspecto que se pode considerar é a sua relação com os *ISP's*. Uma CDN pode não estar afiliada a nenhum ISP. Este tipo de CDN, *multi-ISP CDN*, tenta estabelecer parcerias, e instalar os seus servidores, no maior número de ISP's. A vantagem desta abordagem é poder servir pedidos de clientes que se encontrem nesses ISP's sem que o pedido seja encaminhado para o *backbone*. A Akamai [12] é um exemplo de uma CDN deste tipo.

Um ISP pode igualmente fornecer um serviço de CDN. Neste caso, o ISP distribui os servidores CDN pela periferia da rede. Um ISP com uma rede global pode conseguir uma boa cobertura sem recorrer a parcerias. Esta aproximação permite o acesso a mecanismos de rede de baixo nível, como *multicast*. Exemplo de uma CDN deste tipo é o serviço do ISP AT&T [13].

As CDN também podem ser classificadas pela sua relação com os *CDN customers*. Numa *relaying CDN*, apenas os repositórios de replicação pertencem à CDN, ficando o servidor de origem sob o controlo do *CDN customer*. A CDN encaminha para o servidor de origem todos os pedidos que os servidores da CDN não puderem satisfazer.

Numa *hosting CDN*, o servidor de origem pertence à CDN. Uma das vantagens desta solução é a implementação de algoritmos proprietários para melhorar a eficiência da rede. Uma das utilizações poderá ser o uso de algoritmos mais eficientes na manutenção da consistência entre as várias réplicas.

Existem várias soluções para fazer com que o pedido do cliente seja encaminhado para a rede CDN em vez de para o servidor de origem. Se o servidor de origem pertencer à CDN, como numa *hosting CDN*, este problema não se coloca. No entanto, numa *relaying CDN* os pedidos têm de ser encaminhados para os servidores da CDN. Uma das aproximações consiste em encaminhar o pedido para o servidor de origem. A resposta ao pedido conterá instruções para o cliente enviar os próximos pedidos para a CDN. Por exemplo, o servidor de origem responde a um pedido de um *website*, e certos conteúdos dentro desse website apontam para servidores da CDN. Outra solução é, quando se controla o DNS, devolver o endereço do servidor CDN mais próximo do cliente, em vez do servidor original.

De forma semelhante aos sistemas de cache, um dos problemas que se colocam numa CDN é a manutenção da consistência das réplicas. Neste caso, a consistência dos conteúdos registados pode ser mantida por invalidação. Assim, quando um objecto é modificado no servidor de origem, este envia uma mensagem de invalidação a todos os servidores com réplicas. O que torna esta solução possível em CDN é o número limitado de servidores CDN, sendo possível guardar informação no servidor de origem de quais os repositórios que contêm objectos replicados. Esta solução, que não pode ser implementada nos sistemas de cache standard, devido ao seu número, faz com que as réplicas possam responder imediatamente aos clientes sem contactar os servidores. Apesar desta solução ser bastante eficiente, é obvio que é sempre possível implementar outro tipo de algoritmos de consistência de réplicas.

2.1.6 Adição de valor à cache

A implementação de *proxies* permite que sejam adicionadas funcionalidades que não se encontram de outra forma disponíveis, como por exemplo a filtragem de conteúdos. Neste caso, existem várias abordagens para a implementação de filtragem. Uma delas é efectuar a filtragem no próprio *browser*. A vantagem desta aproximação é que se encontra totalmente sob o controlo do utilizador final. Essa propriedade é também a sua grande desvantagem. Por exemplo, um filtro para controlo de acesso para crianças é facilmente desinstalado por estas, pois muitas crianças

possuem um maior conhecimento sobre computadores que os pais. No entanto, um filtro que se encontre numa *proxy* pela qual todos os pedidos têm de passar é incontornável.

Outra funcionalidade, útil para dispositivos móveis, é a transformação dos conteúdos. Um exemplo de um sistema que implementa este tipo de funcionalidade é o Odyssey [14]. Uma *proxy* pode alterar uma página facilitando a sua visualização num dispositivo móvel. Como estes dispositivos possuem *displays* com uma resolução mais reduzida, a diminuição de qualidade de uma imagem até um certo limite não é perceptível. Adicionalmente, o envio de menos informação resulta em um gasto menor de largura de banda, geralmente limitada neste tipo de equipamento.

Nos últimos anos têm sido efectuadas várias propostas de sistemas/standards que fornecem este tipo de funcionalidade. Entre as várias propostas, encontra-se o ICAP (RFC 3507) [15], que define um standard para adição destas funcionalidades.

2.1.6.1 Protocolo ICAP

O protocolo de adaptação de conteúdos da Internet, ICAP [15], é uma proposta que permite uma solução distribuída e standard para o problema de estender as funcionalidades de um servidor. O objectivo é transferir o processamento de conteúdos personalizados dos servidores de periferia, *edge servers*, para um servidor ICAP. Uma das vantagens é que permite manter os *edge servers* simples, sendo a funcionalidade adicional fornecida pelos servidores ICAP.

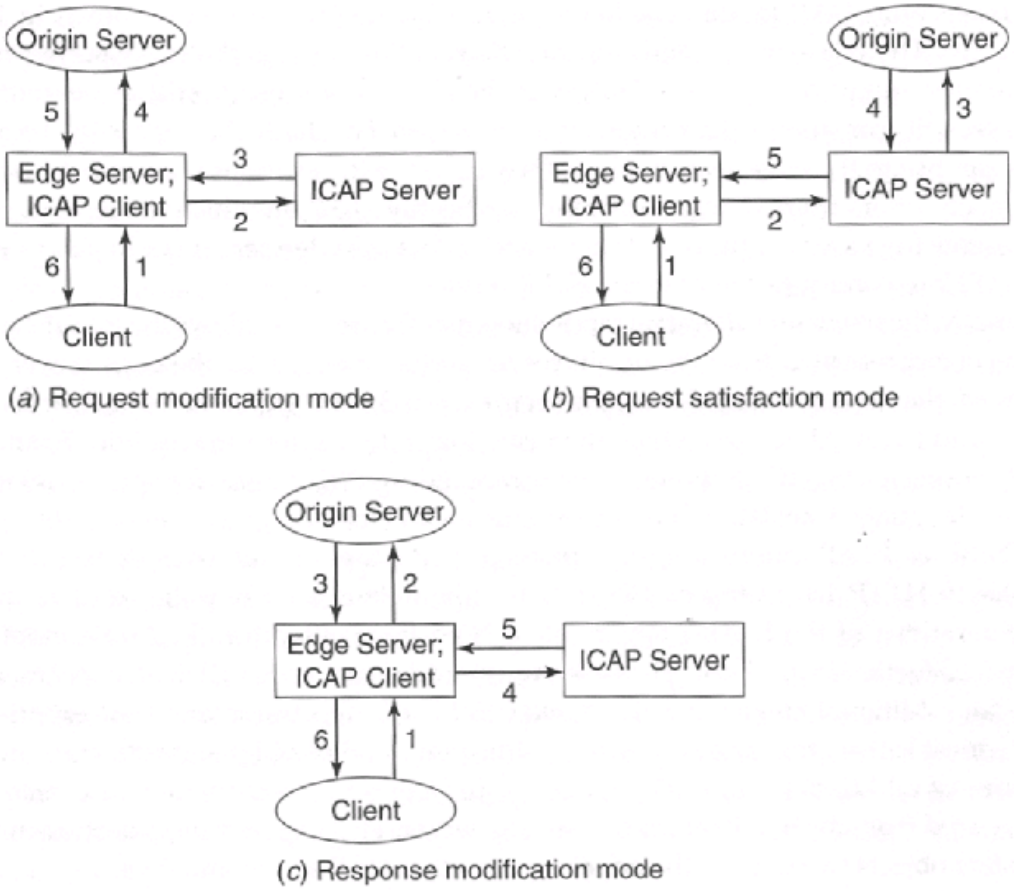

Figura 2.3 - Modos de funcionamento do protocolo ICAP

O ICAP inclui três modos de funcionamento, como se apresenta na figura 2.3. O modo *request modification* permite a modificação do pedido do cliente antes de ser encaminhado para o servidor de origem. Uma das aplicações é a filtragem de conteúdos. No modo request *satisfaction* o servidor ICAP recebe o pedido que vem do cliente ICAP. O servidor resolve o pedido como entender, consultando o servidor de origem se necessário. O modo *response modification* permite a modificação do objecto retornado pelo servidor. Aplicações para este modo são, por exemplo, verificação de vírus, tradução da página para uma língua ou modificação de elementos da página para dispositivos móveis.

2.2 Sistemas baseados na extensão dos clientes/proxies

Nesta secção serão apresentados sistemas baseados na extensão dos clientes e proxies Web através da execução de código genérico. Os sistemas apresentados podem-se dividir em dois tipos. Primeiro, sistemas experimentais, propostos no âmbito de projectos de investigação. Neste contexto, apresentam-se apenas duas soluções exemplificativas das diferentes aproximações propostas (entre outras propostas, encontram-se, por exemplo o sistema Proxy+ [17] e Web& [22]). Segundo, sistemas "comerciais", actualmente disponíveis para utilização generalizada. Alguns destes sistemas começaram como projectos experimentais, estando alguns ainda em fase experimental (e.g. o Google Gears encontra-se em versão Beta).

2.2.1 Soluções experimentais

2.2.1.1 Active Cache

O sistema Active Cache [5] permite executar código nas caches. Para tal o sistema transfere partes de aplicações do servidor para proxies de maneira flexível e a pedido, sob forma de cache applets. Uma cache applet é composta por código fornecido pelo servidor e executada localmente na *proxy*. Uma cache applet está associada a um URL ou a uma colecção de URLs. De forma a resolver o problema da heterogeneidade, o código é escrito numa linguagem independente da plataforma, neste caso o Java.

O funcionamento do Active Cache é o seguinte. Ao receber um pedido, este invoca a cache applet guardada, caso se encontre em cache, e executa localmente, como se descreve na figura 2.4. A cache applet decide então o que fazer, tendo três opções:

- Produzir um novo documento e enviá-lo ao utilizador, podendo guardá-lo localmente na *proxy*.
- Enviar um documento previamente guardado na *proxy*.
- Dar instruções à *proxy* para encaminhar o pedido para o servidor.

Se a cache applet correspondente não se encontrar guardada localmente, o pedido é enviado para o servidor de origem, o qual fornece a cache applet correspondente e a resposta ao pedido.

Em qualquer caso, a *proxy* tem sempre a liberdade de decidir se faz cache ou não de uma certa “cache applet”. Independentemente de ter uma cache applet guardada localmente, o sistema pode encaminhar o pedido para o servidor sempre que achar necessário. A única garantia que o sistema oferece é que não responderá a pedidos com informação guardada em cache se esta não

se encontrar válida. Desta forma a *proxy* tem o controlo sobre que recursos guarda, fazendo a sua própria gestão.

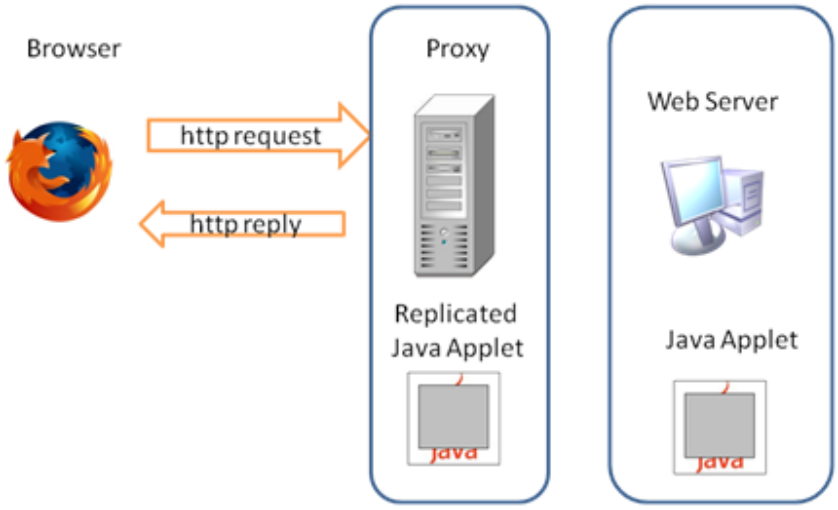

Figura 2.4 – Modelo Active Cache.

Uma cache applet consiste numa classe Java que deve implementar uma interface chamada *ActiveCacheInterface*. Esta interface inclui o seguinte método, fundamental para o funcionamento do sistema:

FromCache (User_HTTP_Request,Client_IP_Address,Client_Name,Cache_File,New_File).

O resultado deste método indica se o ficheiro em cache pode ser devolvido, se um novo ficheiro foi criado e deve ser devolvido como resposta ou se a invocação deve ser encaminhada para o servidor.

Além dos ficheiros de resposta, uma cache applet pode criar objectos e guardá-los em memória persistente. Uma cache applet só pode ler objectos por ele criados. Estes objectos podem também ser lidos pelo servidor de origem da cache applet.

A linguagem Java foi escolhida devido aos seus mecanismos de segurança. No Active Cache os problemas de segurança que se levantam por executar código obtido de um servidor são atenuados pelas operações feitas. Uma cache applet só pode processar o pedido de um cliente na *proxy* e enviar a resposta. Desta forma, classes que possuem problemas de segurança, como a biblioteca *java.io.libraries*, não são usadas. Uma cache applet tem um modelo de segurança bastante restrito: uma cache applet só pode perguntar se alguns objectos existem, ler e escrever objectos e comunicar com o servidor. As restrições de acesso a objectos garantem que cache applets que tenham origem num mesmo servidor confiem umas nas outras e que não cooperem com applets de servidores diferentes.

A *proxy* também mantém controlo do consumo de recursos das várias applets:

- Tamanho do armazenamento em disco.
- Consumo de disco, ou seja, número de *bytes* escritos e lidos.
- Consumo de largura de banda, incluindo o número de *bytes* lidos e escritos.
- Uso do CPU, incluindo em modo de utilizador e modo de sistema.
- Tamanho em memória virtual.

De forma a evitar ataques *DoS*, são impostos limites superiores a cada um dos recursos. Se algum dos limites for ultrapassado então a applet aborta e o pedido é enviado para o servidor.

Um servidor com websites possuindo *banners* rotativos pode usar o Active Cache para reduzir o tamanho da resposta a cada pedido. O servidor exporta para uma *proxy* intermédia, que se encontra entre o servidor e os clientes, uma cache applet que tem como função completar as respostas aos pedidos com os *banners*. Inicialmente é transferida a cache applet, juntamente com todos os *banners* necessários. Deste modo as respostas dadas pelo servidor aos pedidos dos clientes não conterão os *banners*. Estes serão adicionados numa *proxy* que se encontra mais perto dos clientes, idealmente no mesmo ISP, reduzindo o tempo necessário para responder a cada pedido e aumentando a escalabilidade do servidor.

2.2.1.2 Globe, a framework for Consistent, replicated Web objects

O Globe [6] é um sistema distribuído de replicação de objectos *Web* que tem como objectivo melhorar a escalabilidade em acessos a servidores de documentos *Web*. O sistema é construído como um sistema de *middleware* no topo de redes e sistemas operativos existentes. A arquitectura consiste numa colecção de serviços básicos de suporte para replicar objectos distribuídos. A ideia geral por detrás do Globe é que os processos comunicam através de objectos partilhados distribuídos. Os objectos são passivos e múltiplos processos podem aceder simultaneamente ao mesmo objecto. A alteração de um objecto por um processo é visível para todos os outros. Estes objectos partilhados distribuídos são fisicamente distribuídos, ou seja, um objecto pode estar particionado e replicado por múltiplas máquinas ao mesmo tempo. No entanto este particionamento é transparente para os processos dos clientes.

Para um processo aceder a um objecto terá de se registar num ponto de contacto desse objecto. O registo resulta numa interface pertencente ao objecto que está a ser colocado no

espaço de endereçamento do cliente, com uma implementação dessa interface. Esta implementação chama-se Objecto Local.

Um objecto local reside num espaço de endereçamento e comunica com outros objectos noutros espaços de endereçamento. A implementação da interface pode ser diferente dependendo do ponto de contacto. Por exemplo, um mesmo objecto pode fornecer a um cliente uma interface que trata todos os pedidos num servidor, usando por exemplo RPC's, e fornecer a outro cliente uma interface que actua em réplicas locais dos mesmos ficheiros. Claro que estas implementações são transparentes para os processos de cada cliente: eles apenas vêem a interface que lhes é fornecida pelo objecto local. Cada objecto local é composto por vários sub-objectos, como se apresenta na figura 2.5: *Semantic object*, *Comunication object*, *Replication object* e *Control object.*

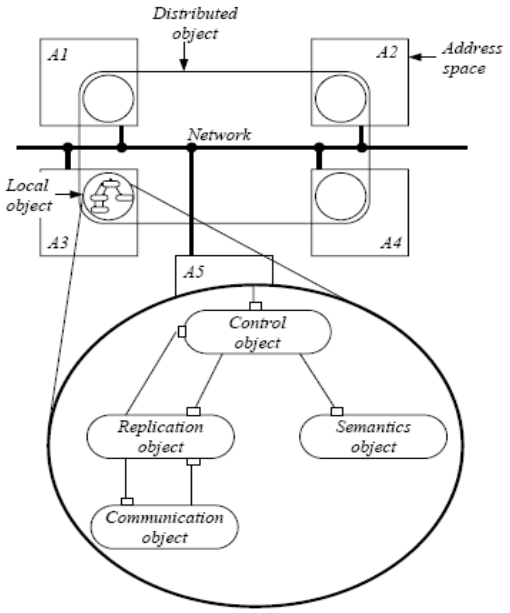

Figura 2.5 – Objecto distribuído por 4 endereços [6]

O objecto *Semantic Object* implementa as semânticas do objecto distribuído. No caso de objectos Web, os *semantic objects* encapsulam os ficheiros que compõem o documento *Web*. O programador apenas é responsável pela construção desses ficheiros e do seu encapsulamento em objectos semânticos. Todos os outros objectos podem ser obtidos a partir de bibliotecas ou criados implementando a interface respectiva.

O objecto *Communication Object* é responsável pela comunicação entre as várias partes do objecto distribuído que reside em diferentes endereços. Dependendo da necessidade de

comunicar com os outros componentes, o *communication object* pode oferecer primitivas de comunicação ponto a ponto, *multicast* ou ambas.

O estado global de um objecto distribuído é composto pelos vários estados dos *semantic objects*. Por razões de desempenho ou tolerância a falhas, *semantic objects* podem encontrar-se replicados. O *Replication Object* é responsável por manter a consistência desses objectos de acordo com uma estratégia de sincronização. Diferentes objectos distribuídos podem ter diferentes *replication objects*, usando algoritmos de replicação diferentes. O *replication object* é controlado pelo *control object*.

O *Control Object* trata das invocações dos processos dos clientes e controla a interacção entre *semantic objects* e *replication objects*. As invocações de pedidos também são tratadas por este objecto.

Quando aplicado a aplicações *Web*, cada documento *Web* é representado por um objecto distribuído. As interfaces destes objectos deverão fornecer métodos de leitura do objecto em formato HTML, para que os *browsers* não necessitem de ser alterados.

2.2.2 Aplicações comerciais

Nesta secção serão apresentados alguns dos sistemas comerciais disponíveis para utilização generalizada. Embora cada um deles tenha diferentes características, incorporam características propostas inicialmente em sistemas experimentais.

2.2.2.1 Flash Lite

O Flash Lite [4] é uma versão do Adobe Flash para dispositivos móveis. Actualmente, a utilização de conteúdos *Flash* em páginas *Web* é muito comum. Uma das razões para tal é o facto do *Flash* permitir animações e transições sem que seja necessário que o cliente de Internet carregue uma nova página. Toda a interacção entre a aplicação e o utilizador pode ser feita em *Flash*. Possui também métodos de acesso à Internet, permitindo a sua utilização em aplicações que necessitem de aceder a recursos em servidores *Web*. O *Flash* começou como *software* para criar animações de forma simples mas foi-se tornando um ambiente de desenvolvimento de aplicações multimédia. O facto de uma aplicação *Flash* poder ser auto-contida torna-a interessante como forma de executar operações sem que seja necessário haver conectividade.

O desenvolvimento do Flash Lite permitiu que estas aplicações sejam usadas em PDA’s e telemóveis. O Flash Lite permite a cache destas aplicações por parte dos dispositivos móveis.

Assim, inicialmente existirá a transferência da aplicação, que pode ser de um tamanho considerável. Após essa transferência, e visto que a aplicação *Web* consiste na aplicação *Flash*, a comunicação entre cliente e servidor pode ser mais reduzida ou mesmo inexistente. Usando como exemplo uma loja *online*, a comunicação pode apenas consistir na actualização dos produtos após a transferência da aplicação, que ocorre apenas no primeiro acesso ao serviço.

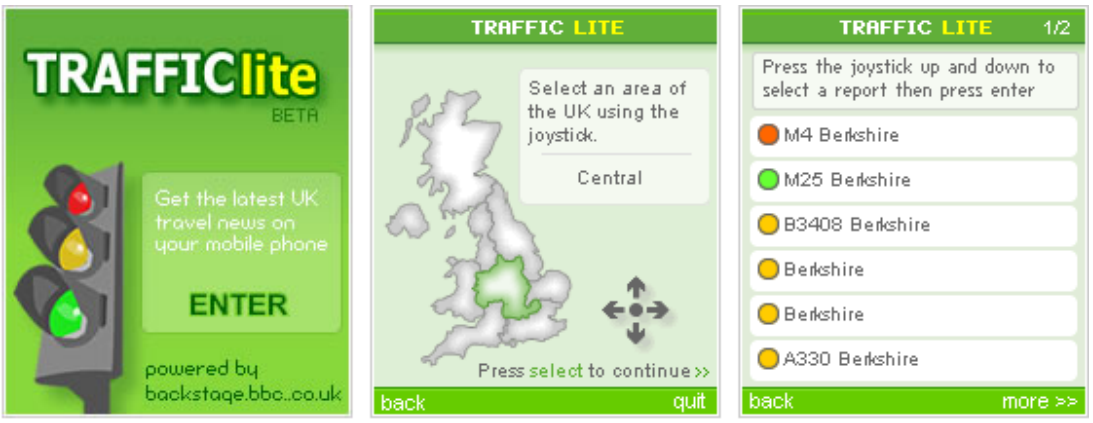

Figura 2.6 – Traffic Lite

Na figura 2.6 apresenta-se um exemplo de uma aplicação *Flash*, o *Traffic Lite*, que permite ver informações sobre estradas do Reino Unido. A informação é actualizada por *RSS feeds*. Os dados são transferidos só quando existem actualizações e permanecem disponíveis mesmo que não exista conectividade. A aplicação base é transferida e instalada uma única vez, não sendo preciso voltar a transferi-la.

2.2.2.2 Google Gears

O *Google Gears* [3] é uma extensão aos browsers que permite a execução de aplicações *Web* em modo parcialmente ou totalmente desconectado. Para tal, os servidores transferem para o cliente todos os ficheiros necessários de modo a que as operações sejam executadas localmente.

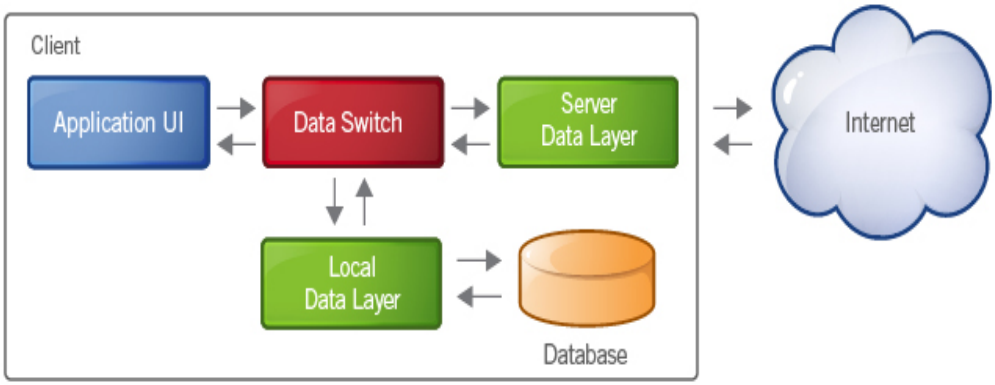

Figura 2.7 – Arquitectura de uma aplicação *Google Gears* [3]

As aplicações desenvolvidas para o Google Gear têm de ser implementadas em JavaScript. O sistema disponibiliza seis bibliotecas para serem usadas pelas aplicações, cada uma com um papel específico: *LocalServer*, *DataBase*, *WorkerPool*, *HttpRequest*, *Timer* e *Factory*.

O *LocalServer* guarda e disponibiliza os recursos necessários à execução da aplicação quando não existe acesso à *web*. A *DataBase* disponibiliza um servidor SQL para o armazenamento de dados localmente. O *WorkerPool* permite a execução de código javascript assincronamente. O módulo *HttpRequest* disponibiliza a funcionalidade de enviar pedidos HTTP, tanto na página principal como nos *workers*. O *Timer* disponibiliza um temporizador que pode ser usado tanto pela aplicação principal como pelos *workers*. A *Factory* instancia todos os outros objectos do *Google Gears*.

As aplicações *Web* desenvolvidas para o *Google Gears* podem dividir-se em dois grupos: *Modal* e *Modeless*.

Numa aplicação *Modal* são os utilizadores que especificam de que forma querem trabalhar, *online* ou *offline*. A vantagem é que estas aplicações são relativamente simples de implementar. No entanto, sempre que a ligação falhar a aplicação irá entrar em modo *offline*, sendo precisa a intervenção do utilizador para voltar a ficar online quando a ligação à rede for recuperada. Por exemplo, o *Google Reader*, apresentado na figura 2.8, é uma aplicação que permite a criação e partilha de documentos entre utilizadores. Um utilizador pode explicitamente indicar à aplicação para correr em modo desconectado, não recebendo nenhumas actualizações aos documentos que foram seleccionados nem podendo fazer actualizações aos documentos que se encontram partilhados.

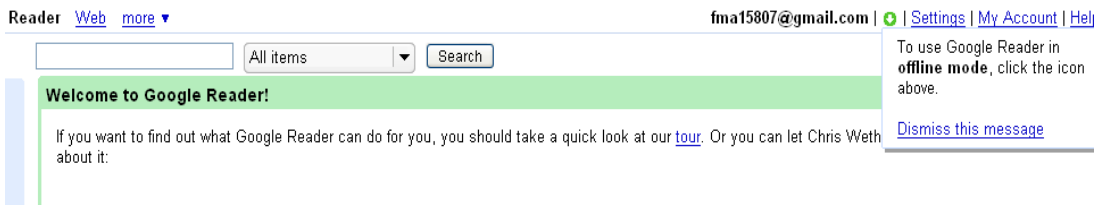

Figura 2.8 – Google Reader

Uma aplicação *Modeless* faz a própria gestão dos modos em que se encontra. Estas aplicações proporcionam uma experiência mais satisfatória aos utilizadores finais pois estes não precisam de se preocupare com o estado da rede. Assim a aplicação trabalhará sem que se note que a rede se encontra intermitente. Um exemplo de uma aplicação que funciona neste modo é o *Gearpad*, apresentado na figura 2.9, que sincroniza os dados implicitamente. Esta aplicação é um processador de texto simples que sincroniza o conteúdo da cópia local com a cópia original sempre que existe conectividade entre o cliente e o servidor.

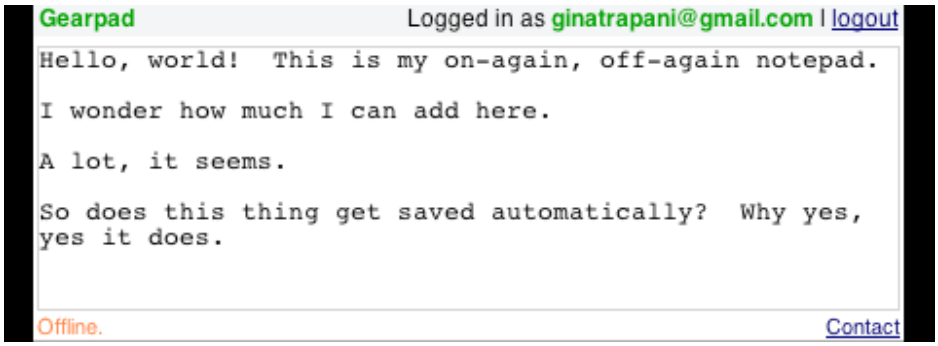

Figura 2.9 - Gearpad

2.2.2.3 WidSets

WidSets [2] é um sistema para telemóvel, inicialmente desenvolvido pela Nokia Research, que permite a visualização de *widgets*. Os *widgets* são mini-aplicações que executam nos telemóveis e permitem o acesso rápido aos conteúdos Web. No exemplo da figura 2.10, apresentam-se dois *widgets*, que permitem aceder a notícias e meteorologia, na mesma interface.

O sistema tem dois modos de execução: *Online* e *Offline*. Quando em modo *online*, os *widgets* são actualizados a pedido ou automaticamente através de *RSS Feeds*. O comportamento dos *widgets* em modo desconectado fica inteiramente à responsabilidade do programador. Por exemplo, o *widget* da *Wikipédia* não permite visualizar recursos previamente descarregados. Outros *widgets*, como alguns jogos, permitem que vários cenários sejam transferidos em modo online, ficando disponíveis para serem jogados mesmo em modo *offline*.

O sistema permite o desenvolvimento de novos *widgets* por programadores que não estejam associados à WidSet, sendo fornecidos uma API e exemplos. Os *widgets* têm de ser desenvolvidos em Helium, uma linguagem de *scripting*. Esta linguagem é muito similar à linguagem JavaME, fornecendo quase todos os métodos que a última oferece, mas permite o carregamento de classes, necessário para a implementação de novas funcionalidades.

Os novos *widgets* ficam à disposição do resto da comunidade. Actualmente estão disponíveis inúmeros *widgets*, entre eles *Gmail* e *Wikipédia*, fazendo com que as necessidades da maioria dos utilizadores sejam satisfeitas.

Figura 2.10 – Interface da aplicação WidSets com dois widgets.

As características anteriores tornam o WidSets um sistema extremamente simples de ser usado pelos utilizadores, simplificando o acesso a informação relevante em dispositivos móveis através da utilização de vários *widgets*. O mecanismo de sincronização implícita garante que os conteúdos estarão sempre actualizados.

2.3 Resumo

As soluções de *caching* estudadas apresentam limitações que não são de fácil resolução. Os métodos baseados na informação dos cabeçalhos HTTP estão dependentes dessa mesma informação, que não é obrigatória. Por outro lado, não é possível fazer *caching* de conteúdos dinâmicos. Técnicas como o *Template Caching* complicam a implementação de aplicações Web. E técnicas baseadas em proxies modificadas, como *Base Instance Caching*, implicam a inserção de máquinas na rede, que depende da boa vontade dos ISP's.

A replicação também nem sempre está acessível a todos os serviços. As *Content Distribution Networks* permitem a replicação de conteúdos mas implicam um investimento financeiro por parte das empresas.

A implementação de novas funcionalidades nas proxies a executar nos servidores padece do mesmo problema que o *Template Caching*: alteração das proxies existentes. Protocolos como o ICAP necessitam da alteração das proxies, que dependem de outras entidades.

O sistema Globe apresenta uma proposta interessante, mas que se baseia numa rede distribuída, obrigando a que as várias entidades colaborem entre si.

O *Active Cache*, *Google Gears*, *Flash Lite* e *WidSets* permitem que as aplicações executem nos clientes. Esta abordagem não implica a alteração da infra-estrutura da rede. O sistema desenvolvido nesta dissertação usa uma aproximação semelhante. Além das várias diferenças nas aproximações tomadas, o sistema Comanche é o único que permite interceptar todos os pedidos efectuados por um cliente. Esta aproximação permite fornecer serviços adicionais utilizando a informação extraída, como se discutiu no capítulo 1.

3. Desenho

3.1 Objectivos e Requisitos do sistema

O sistema a desenvolver, denominado Comanche, pretende melhorar a experiência de cada utilizador na utilização da Web sem recorrer à alteração dos servidores. Por um lado, pretende-se que o sistema permita o aumento de desempenho, explorando os recursos computacionais do cliente. Por outro lado, pretende-se permitir a criação de novos serviços complexos, com geração de conteúdos dinâmicos, sem necessidade de suporte especial nos servidores. Finalmente, pretende-se permitir a criação de um novo tipo de serviço, que actue sobre qualquer acesso do utilizador a páginas Web.

Para que não seja necessário modificar os servidores, o sistema será baseado numa *proxy*, instalada no cliente, que irá interceptar e tratar os pedidos localmente, sempre que possível. Os benefícios do sistema irão variar consoante o tipo de equipamento usado e meio de acesso à Internet. De forma a tratar os pedidos localmente, o sistema será baseado na importação de aplicações, a partir de servidores, para que os pedidos possam ser processados localmente. Para manter a transparência, estas aplicações devem devolver um resultado igual ao que se obteria se o pedido fosse encaminhado para o servidor de destino. Neste caso, o sistema funciona como uma cache da aplicação em vez de ser uma cache de páginas.

A execução de aplicações, cujo código foi obtido remotamente e das quais não se possui qualquer tipo de informação, deve ser controlado por motivos de segurança. Dessa forma o sistema deverá ser capaz de manter as aplicações a correr de forma isolada, não permitindo que estas acedam a recursos, excepto aqueles que elas próprias necessitem e dentro das regras de segurança do sistema.

Assim deverá ser possível definir regras de acesso aos diferentes recursos (e.g. sistema de ficheiros, rede, etc.) para cada aplicação. Desta forma, pode-se evitar que uma aplicação envie informação para fora do sistema ou que receba informação da Internet, aceda a ficheiros locais não permitidos, etc.

Para que o sistema possa ser usado, terá de permitir a criação das aplicações de forma simples, sem que seja necessário aos programadores aprender um novo paradigma. Este requisito é muito importante pois, como se verificou no trabalho relacionado, um sistema que necessite de mudanças significativas no modelo de programação das aplicações ou da infra-estrutura já existentes dificilmente será usado.

O acesso a informação actualizada sempre foi importante. Com o crescente dinamismo da Internet, este requisito impõe uma maior dificuldade às aplicações. Para simplificar este processo, permitindo fazer a sincronização da informação entre o que se encontra em cache nos clientes e o que existe nos servidores, o sistema deverá integrar esta funcionalidade. Embora esta característica tenha pouco impacto em dispositivos bem conectados à Internet, dispositivos móveis poderiam validar os dados em cache sem intervenção do utilizador, garantindo assim que, em períodos de desconexão curtos, poder-se-á aceder localmente a informação relativamente fresca. A qualidade desta informação irá sendo menor quanto maior for o período de desconexão.

3.2 Arquitectura

A arquitectura do sistema, apresentada na figura 3.1, será baseada numa *proxy* HTTP que intercepta todos os pedidos. De forma a correr os pedidos sem ter de os encaminhar para o servidor, o sistema controla a importação do código dinamicamente, sem que seja necessário reiniciar ou recompilar.

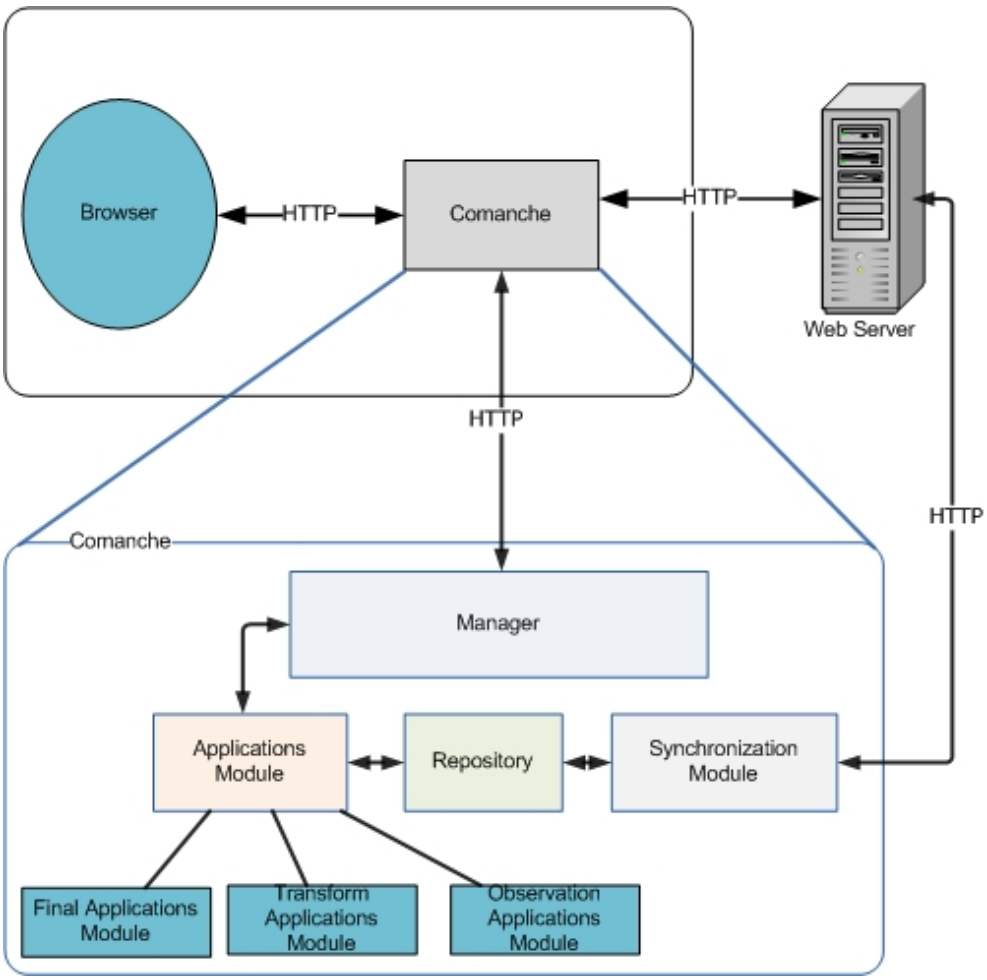

Figura 3.1 – Arquitectura do Comanche

O módulo principal do sistema é o Manager, responsável por processar os pedidos HTTP. Estes pedidos podem ser processados completamente pelo sistema ou reencaminhados para o servidor final.

De forma a guardar eventuais dados, o Comanche possui um sistema de repositório que permite às aplicações abstraírem-se do sistema de ficheiros da máquina em que o Comanche se encontra instalado. Adicionalmente permite controlar a segurança dos acessos efectuados pelas aplicações, nomeadamente aos recursos correspondentes a dados que devam ficar guardados persistentemente.

A gestão das aplicações importadas é efectuada pelo "Applications Module". Este módulo faz as associações entre pedidos e aplicações, para que o Manager possa executar o pedido de forma correcta, executando todas as aplicações necessárias. Tal como o repositório, este

subsistema controla a segurança dos acessos à rede, usando as regras definidas para cada aplicação.

Para além destes módulos, foi indicado como requisito a possibilidade de sincronização de conteúdos HTTP. O módulo de sincronização possibilita que as aplicações indiquem quais os conteúdos que devem ser sincronizados, sendo o conteúdo guardado num ficheiro previamente especificado. De seguida detalham-se os vários módulos do sistema.

3.2.1 Manager

O Manager é o módulo responsável pela execução de pedidos HTTP, aceitando os mesmos numa porta pré-definida. Estes pedidos podem ser reencaminhados para um servidor, executados localmente ou serem pedidos de instalação de uma aplicação. Para tal, será preciso fazer o reconhecimento do pedido HTTP e proceder-se ao seu tratamento. Um pedido poderá ser apenas encaminhado para o servidor de destino, caso este não esteja associado a nenhuma aplicação instalada localmente. Desta forma garante-se a compatibilidade do sistema com as aplicações Web já existentes, continuando estas a funcionar de forma transparente.

Um pedido poderá estar associado a uma aplicação que pode ser executada localmente. Neste caso o pedido é encaminhado para o módulo das aplicações que instala as novas aplicações e faz o processamento dos pedidos relativos às aplicações instaladas, agora localmente. Na secção 3.3 detalha-se este processamento, em face dos diferentes tipos de aplicações.

3.2.2 Applications Module

O módulo das aplicações é responsável pela instalação e gestão das aplicações web estendidas. Está dividido em três módulos, cada um administrando um tipo de aplicação. Estes módulos guardam as aplicações e todos os dados necessários para a execução das mesmas. Os vários tipos de aplicações são descritos em pormenor na secção 3.3.

3.2.3 Repositório

O módulo de repositório fornece todas as primitivas necessárias para as aplicações poderem guardar permanentemente os dados que necessitem. Este módulo permite que as aplicações importadas se abstraiam do sistema de ficheiros e da localização dos recursos, usando este módulo para ter acesso aos mesmos. Adicionalmente, permite às aplicações o acesso a um espaço

do sistema de ficheiros isolado. Isto permite às aplicações guardar dados em ficheiros ou em bases de dados SQL, usando o suporte base do Java.

De forma a permitir uma maior flexibilidade do sistema, é possível associar aplicações Web estendidas. A partilha de informação entre estas aplicações pode ser feita através dos repositórios. Por questões de segurança, os dados de um repositório de uma aplicação apenas podem ser lidos pelas outras aplicações. No futuro, pretende-se permitir que uma aplicação possa conceder permissões de escrita em parte do seu repositório a um conjunto de outras aplicações.

```
public interface Repository
{
        /**
         * Returns the File by it's fileName.
         *
         * @param fileName Name of the file, given by the application
         * @return File or null if doesn't exist
         */
        public File getFile(String fileName);

        /**
         * Creates a new File, or gets the File with the fileName.
         *
         * @param fileName
         * @return the File with the specified fileName. if the File doesn't exist, it is created.
         */
        public File newFile(String fileName);

        /**
         * Deletes the file.
         *
         * @param file
         */
        public void deleteFile(File file);

        /**
         * Deletes the file using the file's name.
         *
         * @param fileName
         */
        public void deleteFile(String fileName);
}
```

Figura 3.2 – Interface do módulo Repositório

A interface do módulo de repositório, apresentada na figura 3.2, permite aceder ao repositório, escondendo os detalhes da implementação. Assim, dado um nome do ficheiro/directoria, a função getFile devove o objecto File que representa o ficheiro. A função de criação de ficheiro, newFile, cria um ficheiro na área da aplicação, sendo o nome passado como argumento à função. Caso o ficheiro já exista, este é devolvido não sendo necessário chamar a

função getFile. A remoção de ficheiros pode ser feita passando como argumento o nome do ficheiro, ou o objecto que o representa, à função deleteFile.

3.2.4 Módulo de Sincronização

O sistema disponibiliza um módulo de sincronização de recursos. Visto que as aplicações importadas irão correr localmente, pode surgir a necessidade de usarem dados actualizados, que devem ser obtidos a partir dos servidores. As aplicações podem indicar quais os recursos para os quais pretendem manter uma cópia local sincronizada. Para tal, devem indicar o seu URL e um ficheiro de destino para guardar o conteúdo obtido, para que em momentos de desconexão se possa ter acesso a informação relativamente fresca.

```
public interface Synchronize
{
	/**
	 * Creates a new Synchronize Resource.
	 *
	 * @param url The URLConnection object that will be used to establish the HTTP connection.
	 * @param fileName The file's name where the content will be saved.
	* @param syncObj this object contains the function executeAfterSync, that will be invoked if the
content accessed changes.
	 * @return true if the resource can be synchronized.
	 * False if the file is already being used to synchronize a resource.
	 *
	 */
	public boolean synchronizeResource(URLConnection url, String fileName, long period,
SynchronizeObject syncObj);

	/**
	 * Starts the synchronization of the resource. Should be called if the resource has been stopped.
	 * @param filename
	 */
	public void startSync(String filename);

	/**
	 * Stops the synchronization of the resource.
	 *
	 * @param filename
	 */
	public void stopSync(String filename);

	/**
	 * Removes the resource from the database. The file is not deleted and its content
	 * is preserved.
	 *
	 * @param filename
	 */
	public void removeSync(String filename);

}
```

Figura 3.3 – Interface do módulo de sincronização

A interface deste módulo, apresentada na figura 3.3, permite que uma aplicação defina um recurso, através do objecto URLConnection do Java, e o nome do ficheiro onde irá guardar os dados. Os dados obtidos são guardados no repositório associado à aplicação que pede a sincronização. Também permite, através do nome do ficheiro, iniciar, parar ou remover a sincronização. O período de sincronização é dado no campo “period”.

Se o conteúdo dos dados sincronizados se alterar, existe a possibilidade de informar a aplicação que os requisitou dessa alteração. Isto é feito chamando o método executeAfterSync, definido na interface SynchronizeObject, apresentada na figura 3.4.

```
public interface SynchronizeObject
{
        public void executeAfterSync();
}
```

Figura 3.4 – Interface SynchronizeObject

Sempre que o conteúdo da resposta se alterarm este método será invocado, caso o objecto que implementa a interface se encontre instanciado.

3.3 Extended Web Applications

O Comanche recorre à importação de aplicações baseadas em Java Web Application. Estas aplicações, a que designámos genericamente de extended web applications, podem ser de três tipos (ver figura 3.5): Final Web Applications, Transform Web Applications, e Observation Web Applications.

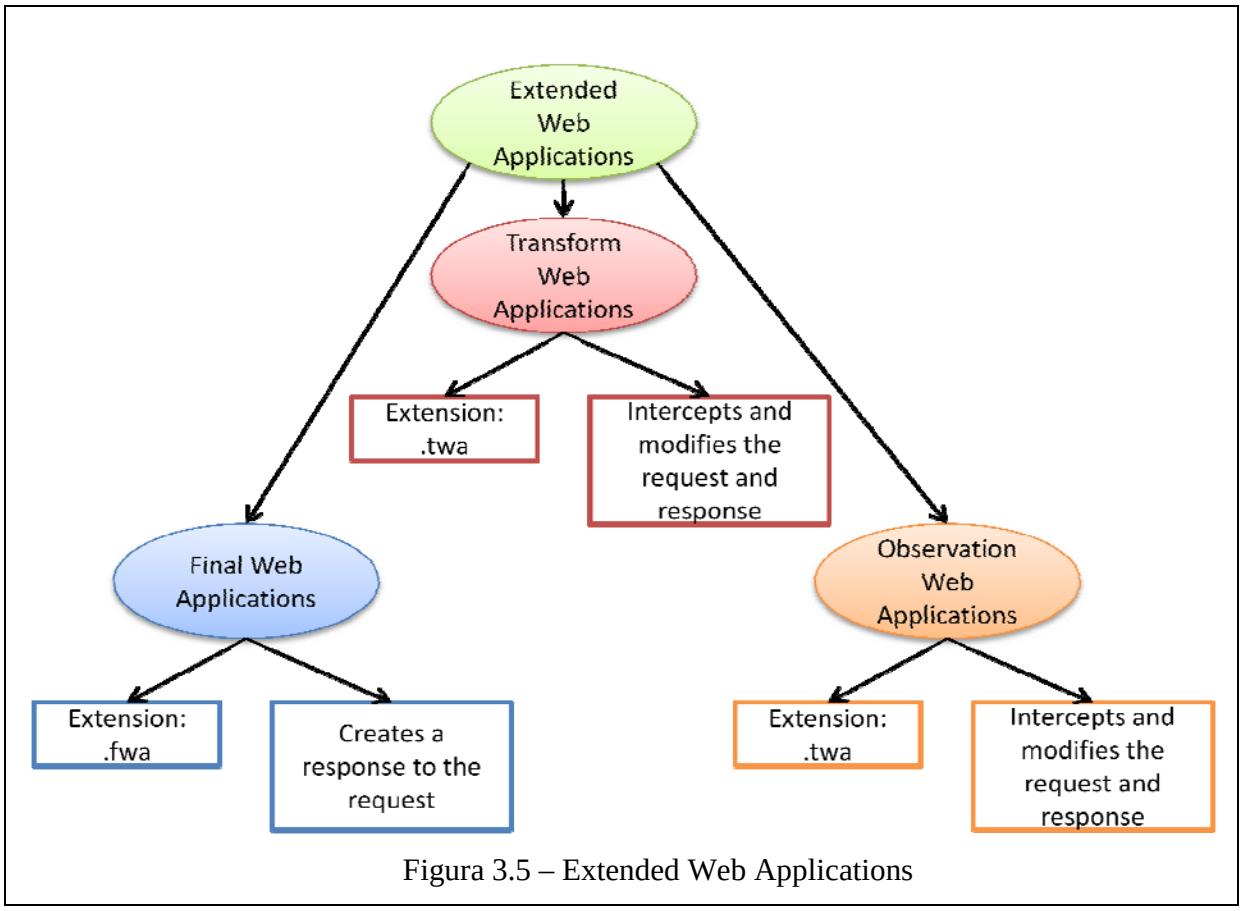

Figura 3.5 – Extended Web Applications

As Final Web Applications, ou aplicações finais, geram localmente páginas que são usadas para responder aos pedidos do cliente.

As Transform Web Applications, ou aplicações de transformação, permitem a alteração dos pedidos dos clientes e das respectivas respostas.

As Observation Web Applications, ou aplicações de observação, podem aceder ao pedido e à resposta, antes desta ser alterada pelas aplicações de transformação, mas não os podem modificar.

Qualquer destas aplicações pode conter ficheiros com conteúdos estáticos (tipicamente páginas HTML, dados da aplicação, etc.) e dinâmicos (Java Servlets e Java Servlet Filters, páginas JSP, código JavaScript, etc.) encapsulados num pacote com a extensão ".war". Estas aplicações, quando instaladas num servidor que as suporte, como o Apache Tomcat [26], executam o pedido e devolvem uma resposta. No entanto, as aplicações de transformação e de observação apenas é normal que contenham Java Servlet Filters.

Uma aplicação Web estendida é composta por dois ficheiros. O ficheiro war, que contém a aplicação, e um ficheiro de configuração, em XML.

O formato do ficheiro de configuração obedece a um XML Schema, apresentado na secção 4.2, indicando para cada tipo de aplicação quais os campos obrigatórios e opcionais.

Um ficheiro "war" contém um ficheiro de configuração, web.xml, e possivelmente, ficheiros das classes que foram acima mencionadas. O ficheiro web.xml contém o mapeamento entre os endereços e os Servlets contidos pela aplicação. Como exemplo, apresenta-se parte do ficheiro web.xml que faz a correspondência entre o caminho do pedido e o Servlet invocado:

```
<servlet-mapping>
        <servlet-name>RssFeedDisplayer</servlet-name>
        <url-pattern>/RssFeedDisplayer</url-pattern>
</servlet-mapping>
```

3.3.1 Final Web Applications

As aplicações finais são aplicações que processam um pedido e produzem uma resposta, evitando assim que o sistema encaminhe o mesmo para um servidor remoto. As aplicações finais representam aplicações Web em Java que estariam a correr em servidores e as quais se pretende que executem, total ou parcialmente, no cliente, poupando os recursos computacionais dos servidores de destino e reduzindo a latência da operação.

Não se pretende que as aplicações finais sejam cópias exactas das aplicações existentes, mas apenas uma parte das mesmas. No entanto aplicações stand-alone poderão executar totalmente do lado do cliente, sem qualquer parte a executar no servidor.

Estas aplicações podem ser complexas, como uma loja de comércio online, ou mais simples, fornecendo apenas uma interface gráfica para o utilizador comunicar com a aplicação a correr num servidor remoto.

Uma aplicação final é, geralmente, composta por um conjunto de páginas estáticas (ficheiros HTML e CSS) e dinâmicas (Java Servlets e páginas JSP).

Para se instalar uma aplicação final basta aceder a um endereço que inclua na sua *path* a extensão ".fwa/". Por exemplo, ao aceder ao URL http://www.acme.com/myapp.fwa/x.html, o sistema vai instalar a aplicação que se encontra definida em http://www.acme.com/myapp.fwa.

3.3.2 Transform Web Applications

As aplicações de transformação interceptam tanto o pedido como a resposta, o que permite proceder à alteração dos mesmos. Estas aplicações podem implementar qualquer um dos modos do protocolo ICAP [15], permitindo desta forma estender as funcionalidades de um servidor sem a necessidade de instalação de hardware especializado.

Uma aplicação de transformação pode ser um filtro de conteúdos que, por exemplo, não permite que pedidos de imagens continuem a ser processados, ou que altere parte da resposta a um dado pedido.

Uma aplicação de transformação é definida como um Java Servlet Filter, como se detalha na secção 4.2.2.

Para se instalar uma aplicação de transformação basta fazer um pedido HTTP cuja terminação seja ".twa". Por exemplo, o acesso ao URL http://www.acme.com/transformApp.twa irá resultar na instalação da aplicação respectiva.

3.3.3 Observation Web Applications

As aplicações de observação processam os pedidos e respostas mas sem alterarem os mesmos.

Estas aplicações são semelhantes às aplicações de transformação. No entanto, o sistema garante que qualquer processamento por parte destas aplicações não poderá alterar o pedido nem a resposta. As aplicações de observação têm como objectivo a recolha de dados que podem ser usados por outras aplicações finais. Outra diferença, em relação às aplicações de transformação, é o seu processamento. O sistema executa as aplicações de observação de forma assíncrona em relação ao processamento do pedido. Isso implica que o custo adicionado pela execução de uma

aplicação de observação é muito menor, quando comparado com o custo da execução de uma aplicação de transformação.

Como as aplicações de transformação, as aplicações de observação podem estar associadas a aplicações finais, tendo as aplicações finais acesso aos repositórios das aplicações de observação associadas. Assim pode-se ter várias aplicações finais que façam uso de uma mesma aplicação de observação.

Os endereços onde estas aplicações residem têm a terminação ".owa". Por exemplo, o acesso ao endereço http://www.acme.com/observationApp.owa levará o sistema a instalar a aplicação correspondente.

3.4 Instalação e processamento das Extended Web Applications

O Comanche é instalado localmente, sendo necessário que o cliente encaminhe os pedidos para a porta em que este executa. Os pedidos executados podem levar a dois tipos de operação: acesso a página/recurso e instalação de aplicação.

Os pedidos de acesso a uma página/recurso são encaminhados para um servidor remoto ou para uma aplicação final, sendo aplicadas as aplicações de transformação e de observação associadas ao pedido. Um pedido pode estar associado a várias aplicações de transformação e de observação, que devem ser invocadas. Para definir a ordem de execução para as aplicações de transformação, usou-se o conceito de altitude. Quanto maior o valor de altitude, mais tarde é chamada a aplicação de transformação. Ou seja, aplicações com valores baixos de altitude têm acesso ao pedido mais cedo. Aplicações com valores mais altos têm acesso à resposta mais cedo, tal como se pode ver na figura 3.6.

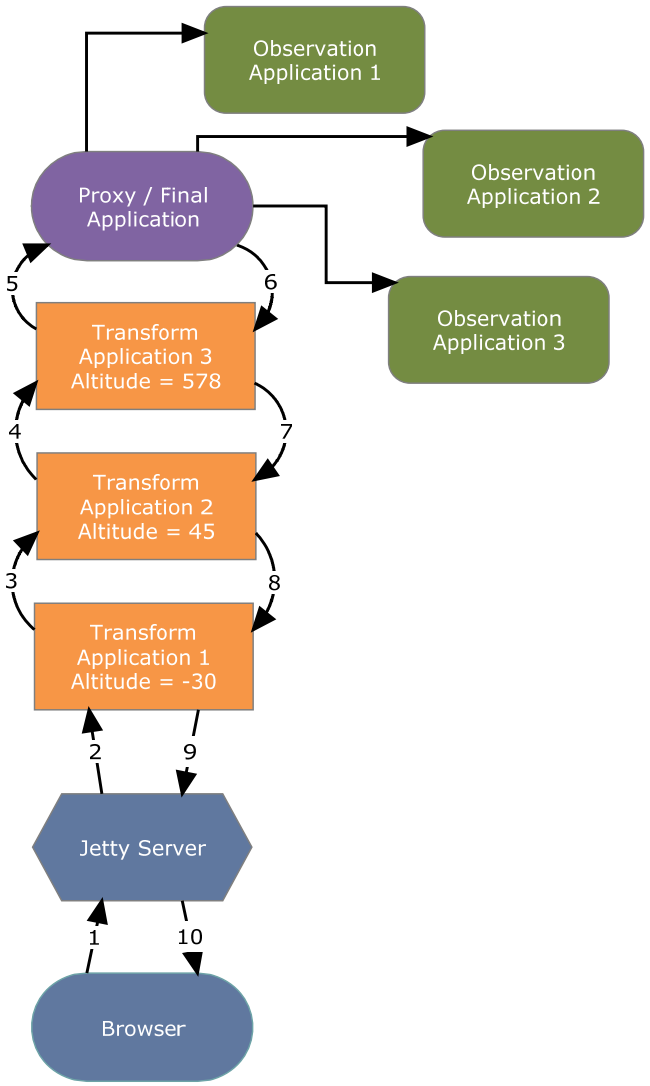

Figura 3.6 – Execução de um pedido com aplicações de transformação e observação associadas

As aplicações de observação não necessitam de altitude, pois o sistema garante que as respostas a que elas têm acesso não sofreram alterações.

Quando o cliente acede a um URL que desencadeia uma operação de instalação, o sistema encaminha este pedido para o módulo das aplicações, que acede ao ficheiro de configuração e cria uma página de instalação com a informação relevante para o utilizador. Com base na informação apresentada, o cliente decide se aceita instalar a aplicação. Se decidir instalar, o módulo das aplicações descarrega o ficheiro da aplicação, com a extensão .war, e procede à instalação da aplicação. Após instalada, a aplicação fica a executar localmente.

As aplicações finais ficam associadas ao endereço de instalação, sendo todos os pedidos, com esse prefixo, redireccionados para a aplicação que lhe está associada para que seja executado localmente. Por exemplo, a instalação da aplicação residente em http://www.acme.com/myapp.fwa irá fazer com que os pedidos http://www.acme.com/myapp.fwa/x.html e http://www.acme.com/myapp.fwa/x.gif sejam encaminhados para a aplicação, de forma a serem processados por ela localmente.

As aplicações de transformação ficam associadas a pedidos que satisfaçam uma dada expressão regular. Assim pode-se fazer uma aplicação de transformação que não deixe prosseguir os pedidos de imagens com a extensão GIF ou bloquear o acesso a um servidor. A instalação de aplicações de transformação que alterem as respostas obtidas é da responsabilidade do utilizador final, pois as mesmas não podem ser instaladas sem o consentimento do mesmo.

As aplicações de observação seguem as mesmas regras que as aplicações de transformação, ficando associadas a expressões regulares, mas não interferem no pedido nem na resposta dada ao cliente.

Quando se usa o sistema, os pedidos destinados a servidores externos têm um custo acrescido, visto que se procede à análise do mesmo e a eventuais execuções de aplicações de transformação, antes de o encaminhar para o servidor de destino. No entanto, como se mostra na secção 5.2, este atraso não é significativo.

3.5 Segurança

A importação de código, do qual não se tem qualquer conhecimento nem base de confiança, pode ser um factor de desencorajamento para o uso do sistema. Assim um dos requisitos do sistema é garantir a segurança da máquina do utilizador. Para tal as aplicações correm em isolamento.

Para além de garantir o isolamento, o sistema controla o acesso aos recursos locais. No protótipo actual do sistema, apenas se controla o acesso a ficheiros e à rede, mas seria possível controlar igualmente a possibilidade de criar *Threads*, o tempo de execução, etc. Para todos os recursos podem ser definidas regras que especifiquem as permissões e limites dos recursos a que uma aplicação tem direito. Na secção 4.6 detalha-se a implementação do mecanismo de segurança.

4. Implementação

Neste capítulo descreve-se a implementação do sistema. Esta descrição não pretende ser exaustiva, mas antes focar os pontos mais importantes e não triviais do mesmo.

4.1 Tecnologias de suporte

O Comanche foi implementado com base na linguagem Java. Esta decisão deveu-se a várias razões que se identificam de seguida.

O Java apresenta como característica base ser independente da plataforma onde é compilado. Como se pretende fazer um sistema que funcione na maior variedade de dispositivos possível, incluindo dispositivos móveis, o Java permite que o mesmo código possa executar nas várias plataformas sem sofrer alterações. Por outro lado, o Java oferece uma panóplia de bibliotecas que facilitam a interacção com a Web.

Visto que se pretende executar aplicações Web no cliente, decidiu-se usar como suporte um servidor aplicacional existente que permitisse executar as mesmas sem grandes modificações. A escolha recaiu sobre o Jetty [23], um servidor aplicacional implementado em Java, que permite a execução de Java Web Applications. Este servidor também permite a inclusão do mesmo em aplicações Java, sendo esta uma característica importante para o sistema que se pretende desenvolver. Testes iniciais mostraram que era possível implementar o sistema usando-o como base. Outra vantagem que o servidor Jetty apresentou foi a existência de uma proxy que trata os pedidos HTTP.

Inicialmente os dados e estruturas internas do sistema que são necessários guardar de forma persistente, ou seja, em disco, eram serializados. No entanto isso revelou-se pouco eficiente, pois a mínima alteração implicava uma escrita de grandes dimensões em disco. Assim, decidiu-se usar uma base de dados para guardar esta informação. Foi tomada a opção de usar uma base de dados transaccional leve, a JavaDB. Esta base de dados é a versão suportada pela Sun da base de dados Derby [24], desenvolvida pela Apache, e totalmente implementada em Java. A facilidade de ser embebida nas aplicações Java e a sua reduzida dimensão tornam-na atractiva como solução para guardar dados persistentemente.

O uso do Jetty como servidor simplifica a implementação do sistema. Primeiro, o servidor recorre a eventos para processar o pedido, sendo assim um servidor assíncrono capaz de processar os pedidos concorrentemente. Desta maneira, não é necessária a criação de Threads que processem os pedidos, sempre que um novo pedido chega ao sistema. Segundo, o servidor, sempre que recebe um pedido HTTP, cria os objectos HttpServletRequest e HttpServletResponse, que representam o pedido e a resposta respectivamente. O sistema implementado usa a informação do pedido para o encaminhar para o módulo correspondente.

4.2 Comanche

O Comanche é composto por vários módulos, como já foi descrito na secção 3.2. Este módulos necessitam de áreas em disco onde possam guardar persistentemente os dados. Em seguida apresenta-se a organização em disco do sistema.

4.2.1 Armazenamento

A directoria base organiza-se em oito directorias (figura 4.1). Cada uma destas directorias contem um tipo de ficheiros.

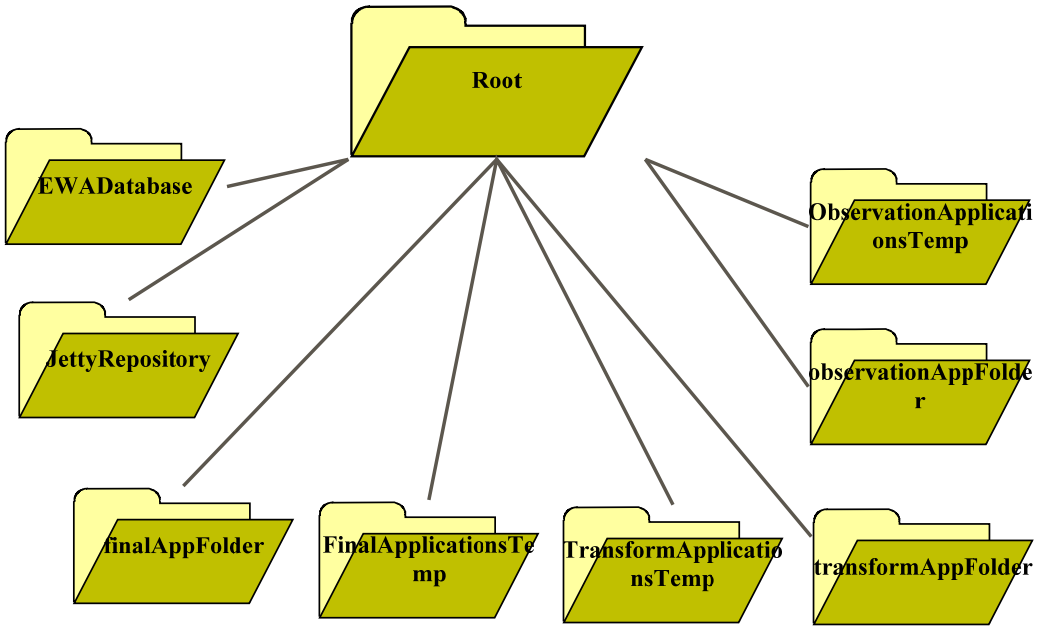

Figura 4.1 – Estrutura da directoria base do Comanche

A directoria EWADatabase contém a base de dados usada para guardar a informação do sistema.

A directoria finalAppFolder contém os ficheiros XML e war das aplicações finais instaladas. Os ficheiros das aplicações de transformação e observação também têm directorias equivalentes, transformAppFolder e observationAppFolder.

A directoria JettyRepository contém as áreas em disco das aplicações. Nesta directoria são criadas pastas, uma para cada aplicação, onde a aplicação poderá criar e apagar ficheiros.

As directorias FinalApplicationsTemp, TransformApplicationsTemp e ObservationApplicationsTemp contêm as pastas temporárias que o Jetty cria, quando uma aplicação é montada.

4.2.2 Interacção com o Comanche

O utilizador pode interagir com o sistema, mais especificamente pode instalar aplicações, a partir das páginas de instalação, parar uma aplicação e removê-la do sistema. Para as duas últimas opções foi criado um serviço Web que permite gerir as aplicações instaladas. Este serviço pode ser acedido no endereço http://www.extendedwebapp.org/. Ao ser acedido, o sistema encaminhará o pedido para a aplicação de gestão do sistema. Por agora apenas se permite que as aplicações sejam iniciadas, paradas ou removidas, sendo também possível alterar a altitude das aplicações de transformação.

4.3 Extended Web Applications

As Extended Web Applications são, como já foi descrito na secção de 3.3, Java Web Applications. Estas aplicações são desenvolvidas para serem executadas em servidores. Para correrem localmente, o Jetty fornece objectos e métodos que permitem a sua execução. Assim, para instalar uma nova aplicação, o Comanche regista no Jetty uma nova aplicação, associando-a a um dado URL.

Para que as aplicações possam usar o módulo repositório, é necessário que este seja passado para as aplicações. Para tal, adiciona-se um atributo ao contexto em que a aplicação executa, o qual pode ser acedido pela aplicação de forma standard.

Para as aplicações Web finais é adicionado mais um atributo, que contém todos os repositórios das aplicações de transformação e observação que ela indicou necessitar.

A gestão das aplicações, e a instalação das mesmas é da responsabilidade do módulo das aplicações. Este encontra-se divido em três sub-módulos, como indicado na secção 3.2. Cada um dos módulos é responsável por um dos tipos de aplicação implementados.

Tal como já foi mencionado, os ficheiros contendo a aplicação e a configuração são guardados em disco. Para evitar que ocorresse a colisão de nomes de ficheiros, é aplicada uma função de dispersão segura, neste caso SHA-256, ao URL acedido para instalar a aplicação. É assim gerado um prefixo único que é depois usado para criar todas as pastas e ficheiros necessários. Nomeadamente, é usado como prefixo para gerar o nome dos ficheiros transferidos do servidor, aquando da instalação. Também é usado para nomear a pasta temporária que o Jetty necessita para executar as aplicações e a pasta que indica a raiz da área em que cada aplicação tem permissões de leitura e escrita.

4.3.1 Aplicações Web Finais

Uma aplicação final é composta por um ficheiro de configuração e um ficheiro war, que irá ser instalado de forma a poder processar os pedidos. O ficheiro de configuração obedece ao XML Schema apresentado na figura 4.2.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="EWAfinalApp">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="TransformApplication" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="TransformApplicationUrl" type="xs:string"/>
        <xs:element name="TransformApplicationOpenConnPermission" type="xs:int"/>
        <xs:element name="TransformApplicationReceiveConnPermission" type="xs:int"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="ObservationApplication" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="ObservationApplicationUrl" type="xs:string"/>
        <xs:element name="ObservationApplicationOpenConnPermission" type="xs:int"/>
        <xs:element name="ObservationApplicationReceiveConnPermission" type="xs:int"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="MessageToUser" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figura 4.2 – XML Schema do ficheiro de configuração das aplicações web finais

Existe apenas um campo obrigatório: a mensagem para o utilizador. Os elementos opcionais indicam quais as aplicações de transformação e de observação associadas, e as permissões para as mesmas.

A informação sobre as aplicações instaladas é guardada na base de dados de forma a garantir a sua persistência.

Sempre que se detecta um endereço com a cadeia de caracteres ".fwa/", extrai-se o endereço até essa cadeia e verifica-se se essa aplicação está instalada. Se estiver instalada, a aplicação é invocada. Caso não se encontre instalada, o pedido é remetido para a Proxy, que o encaminha para o módulo das aplicações. Este módulo encarrega-se da instalação, já descrita na secção 3.4.

4.3.2 Aplicações Web de Transformação

Uma aplicação de transformação tem como objectivo interceptar o pedido e a resposta. Devido à possível complexidade das aplicações de transformação, foi tomada a decisão de usar novamente Java Web Applications para se proceder à importação destas aplicações. Estas, tal como as aplicações finais, são constituídas por um ficheiro de configuração e um ficheiro war. O ficheiro de configuração tem de obedecer a um XML Schema, diferente do usado para as aplicações finais e que se apresenta na figura 4.3.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="EWATransformApp">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="fileLocation" type="xs:string"/>
    <xs:element name="altitude" type="xs:long"/>
    <xs:element name="AssociatedExpressions">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="AssociatedExpression" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="MessageToUser" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figura 4.3 – XML Schema do ficheiro de configuração das aplicações web de transformação.

Os elementos obrigatórios deste ficheiro são o endereço do ficheiro war, a altitude e, no mínimo, uma expressão à qual a aplicação irá ficar associada.

Ao contrário das aplicações finais, estas não estão associadas a um endereço Web. As aplicações de transformação são associadas a expressões regulares. Para simplificar, estas expressões podem ser constituídas apenas por strings. Assim, uma aplicação Web de transformação pode estar associada a todos os pedidos de imagens do tipo GIF ou associada apenas a um servidor. Por exemplo, a expressão "fct" fará com que uma aplicação intercepte todos os pedidos que contenham fct, ou seja, a qualquer página residente nos servidores da Faculdade de Ciências e Tecnologia. No entanto se a expressão for "di.fct.unl.pt", só os pedidos que forem encaminhados para os servidores do Departamento de Informática serão interceptados.

As aplicações de transformação são definidas como *Java Servlet Filters*. Estes objectos têm acesso ao pedido e à resposta antes e depois destes serem processados. Ao ser instalada uma aplicação de transformação, esta terá de ter definido no seu ficheiro "web.xml" um Filter com o nome "EntryFilter". Este objecto será invocado quando um pedido deva ser processado pela aplicação de transformação.

Todas as aplicações de transformação podem parar a execução do pedido, devolvendo imediatamente o resultado, pois para este prosseguir terão de chamar o método doChain. Deste modo pode-se implementar, por exemplo, aplicações de controlo de conteúdos, executando uma aplicação de transformação que não deixe prosseguir certos pedidos.

As aplicações de transformação têm acesso à resposta ao pedido em dois formatos: vector de bytes e DOM. As alterações feitas a qualquer um dos formatos irão repercutir-se na resposta. Para produzir a resposta em formato DOM, usa-se a biblioteca org.cyberneko.html [27].

4.3.3 Aplicações Web de Observação

As aplicações de observação têm como objectivo retirar informações do pedido e da resposta, sem os alterar. Estas aplicações são, como as aplicações de transformação, baseadas em *Java Servlet Filters*. O seu ficheiro de configuração é praticamente igual ao das aplicações de transformação, tendo de ser validado pelo Schema apresentado na figura 4.4.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="EWAObservationApp">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="fileLocation" type="xs:string"/>
    <xs:element name="AssociatedExpressions">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="AssociatedExpression" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="MessageToUser" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figura 4.4 – XML Schema do ficheiro de configuração das aplicações web de observação.

Como se pode ver, apenas o nome do elemento raiz é alterado. No entanto, a execução das aplicações de observação é muito diferente da das aplicações de transformação.

Visto que não contribuem directamente para o resultado da resposta enviada ao cliente, o processamento pode ser feito assincronamente. Isto permite que o custo temporal, acrescido a um pedido, de correr uma aplicação de observação seja mínimo.

Dada a impossibilidade de se guardarem cópias dos objectos HttpServletRequest e HttpServletResponse, por os mesmos estarem ligados à execução corrente de um pedido, a solução não pode ser baseada directamente em cópias destes objectos. Assim, decidiu-se, para ter uma solução rapidamente, definir a interface ObservationApplication, apresentada na figura 4.5, que as aplicações de observação devem implementar.

```
package ObservationWebApplications;

import org.w3c.dom.Document;

public interface ObservationApplication
{
public void execute(String requestUrl, String request, String response, byte[] responseContent, Document doc);
}
```

Figura 4.5 – Interface das aplicações de observação

Esta interface apenas contém um método que recebe como parâmetros o endereço do pedido, a representação do pedido e da resposta, em cadeias de caracteres, o conteúdo da resposta, como um vector de bytes, e a representação do mesmo conteúdo em formato DOM. Por

agora, estes parâmetros são suficientes para o processamento das aplicações de observação existentes.

Analogamente às aplicações de transformação, o *Java Servlet Filter* invocado deverá ter como nome, no ficheiro web.xml, EntryFilter.

4.4 Proxy

A Proxy é o sub-módulo do Comanche que está encarregue de processar os pedidos que não são executados localmente. Para tal, alterou-se o módulo de Proxy HTTP do Jetty para lidar com os pedidos de instalação de aplicações. Assim a *proxy* interpreta os pedidos e executa-os, podendo encaminhá-los para o servidor de destino ou, caso sejam pedidos de instalação de uma aplicação Web estendida, encaminhá-los para o respectivo módulo.

4.5 Repositório

O módulo Repositório permite que as aplicações Web estendidas possam escrever e ler ficheiros, abstraindo-se da sua localização física em disco.

A sincronização de conteúdos foi incluída neste módulo, pois os conteúdos a sincronizar são guardados em ficheiros na área da respectiva aplicação. Um ficheiro apenas pode ser usado para sincronizar um dado recurso.

Para implementar a sincronização de pedidos, o sistema mantém threads que efectuam esse processo, de acordo com as regras indicadas pelas aplicações.

4.6 Base de dados

Como se referiu anteriormente, para armazenar a informação interna de forma segura e eficiente, foi usada uma base de dados relacional, para guardar persistentemente os dados, neste caso o JavaDB. Para permitir tempos de resposta mais rápidos no processamento dos pedidos, são mantidos em memória central, recorrendo a estruturas de dados como tabelas de dispersão e listas, os mesmos dados. Assim, sempre que se alteram os dados, estas alterações são efectuadas em memória central e na base de dados.

Os dados a serem guardados dizem respeito às aplicações Web estendidas. Para além de todos os dados necessários para a aplicação ser instalada sempre que o Comanche inicia, como a directoria onde está o ficheiro que representa a aplicação e as permissões de acesso, guarda-se

também informação sobre os ficheiros criados nos repositórios. O esquema da base de dados é apresentado na figura 4.6.

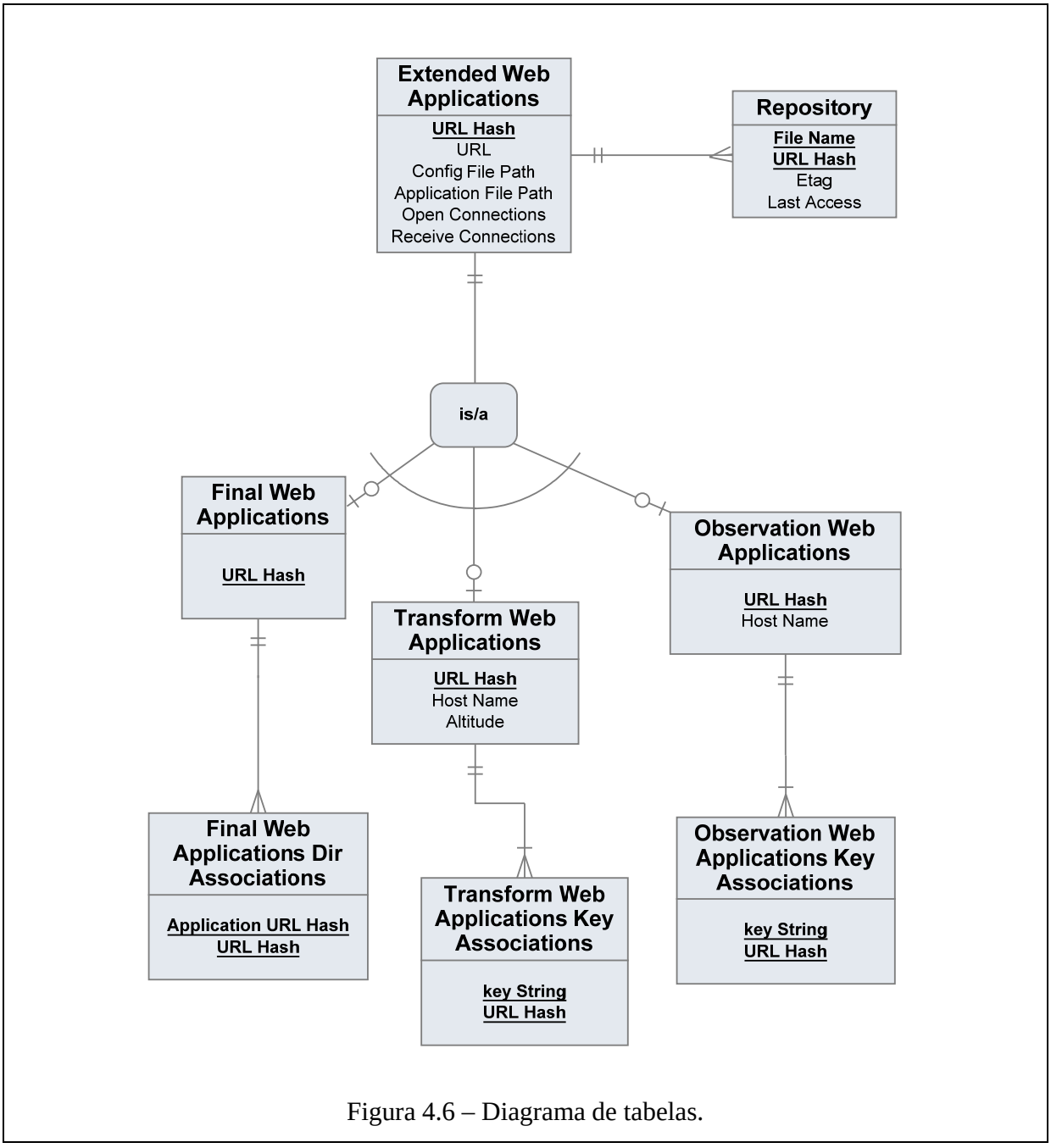

Figura 4.6 – Diagrama de tabelas.

A tabela Extended Web Applications contém a informação comum aos vários tipos de aplicações. Os tuplos desta tabela são constituídos pelos campos URL, que contém o endereço da aplicação na rede, Config File Path e Application File Path, que contêm os caminhos para os ficheiros de configuração e da aplicação, Open Connections e Receive Connections, que indicam

as permissões que a aplicação contém, e URL Hash, que é o valor obtido quando se faz a dispersão do endereço e que é usado como chave primária.

A tabela Final Web Applications não contém qualquer campo, pois apenas necessita da chave URL Hash. Esta tabela é necessária para se poder identificar rapidamente quais são as aplicações web finais.

A tabela Final Web Applications Dir Associations guarda a informação sobre as associações entre aplicações. Os campos de cada tuplo são o valor da dispersão do endereço da aplicação final e o valor da dispersão do endereço da aplicação de transformação ou da aplicação de observação. Estes dois campos constituem a chave primária desta tabela.

Os dados necessários para montar uma aplicação de transformação são guardados na tabela Transform Web Applications. Os tuplos são constituídos pelo nome do servidor, em Host Name, e o valor da altitude, no campo Altitude.

Como foi já mencionado nas secções 3.3 e 4.2, as aplicações de transformação encontram-se associadas as expressões. Estes dados são guardados na tabela Transform Web Applications Key Associations. Os campos são o valor da dispersão do endereço e a expressão. Ambos os valores constituem a chave primária, sendo o campo URL Hash uma chave externa para a tabela Transform Web Applications.

As tabelas Observation Web Applications e Observation Web Applications Key Associations guardam os dados referentes às aplicações de observação. A diferença é que as aplicações de observação não possuem altitudes.

A tabela Repository contém a informação sobre os ficheiros criados, a partir do módulo repositório, pelas aplicações. Cada elemento da tabela é constituído pelo nome do ficheiro, File Name, o valor URL Hash, que identifica a aplicação que o criou, o valor Last Access e Etag. Os últimos dois campos são usados para guardar informação relativamente a conteúdos sincronizados, indicando qual o valor do último acesso e o valor Etag, caso este seja enviado na resposta do pedido.

4.7 Segurança

Uma das questões que surgiu com a execução de código importado foi garantir a segurança dos dados presentes no sistema de ficheiros. Visto que as aplicações Web podem conter qualquer código, seria simples implementar uma aplicação que lesse ficheiros do utilizador, e que os

transmitisse para uma outra máquina, ou que simplesmente apagasse todos os ficheiros a que tivesse acesso.

Assim, foi necessário implementar políticas de segurança, recorrendo ao mecanismo de segurança do Java. Nesta versão do protótipo, a segurança apenas está implementada relativamente a dois recursos: acesso a ficheiros e acesso à Web. No entanto, a solução desenvolvida permite estender o controlo a todos os recursos controlados pelo mecanismo de segurança do Java.

O acesso a ficheiros foi restringido a uma área, por aplicação, onde podem ler e escrever. Assim garante-se que todos os ficheiros alterados por uma aplicação foram criados por ela ou colocados pelo utilizador nessa zona, ficando este responsável por eventuais danos. Adicionalmente, uma aplicação final tem acesso de leitura às áreas das aplicações de transformação e de observação associadas.

Para o controlo de acesso à Internet, foram criados três níveis de segurança, para a criação e recepção de ligações: sem restrições, apenas o servidor de origem e sem acesso. Regras mais elaboradas poderão ser definidas no futuro.

Para implementar esta política de segurança foi necessário definir um novo gestor de segurança e um novo *classLoader* e integrá-los com o sistema Jetty, de acordo com o modelo de segurança do Java que permite associar a cada classe um conjunto de permissões.

No Java, ao verificar as permissões de execução de uma operação, verificam-se as permissões de todas as classes que se encontram na pilha de execução. Assim surgiu um problema importante. Como o encadeamento das aplicações de transformação é feito por chamadas recursivas, a pilha de execução iria levar à reunião de todas as restrições associadas às aplicações de transformação do pedido a ser processado. Por exemplo, uma aplicação de transformação deixaria de ter permissões de leitura e escrita no seu repositório se uma outra aplicação de transformação fosse invocada antes. Para solucionar o problema, quando se verificam as permissões numa pilha de execução, apenas se verificam as restrições até surgir o primeiro encadeamento. Assim, as permissões verificadas dizem respeito apenas à aplicação que está a ser executada, permitindo o acesso ao repositório da mesma.

5. Avaliação

A avaliação do sistema Comanche foi feita em duas vertentes. Primeiro realizou-se uma avaliação qualitativa do sistema, em que através do desenvolvimento de aplicações se verificou se o Comanche conseguia alcançar os objectivos propostos, executando as aplicações web estendidas de forma correcta e permitindo o seu desenvolvimento de forma simples.

De seguida realizou-se uma avaliação quantitativa, onde se pretendeu avaliar o custo acrescido na execução de pedidos HTTP não executados localmente e o impacto do sistema no tempo de resposta para os pedidos a aplicações finais.

5.1 Avaliação qualitativa

Após o desenvolvimento do sistema Comanche, foram desenvolvidas algumas aplicações para avaliar o funcionamento do sistema.

Primeiro, pretendeu-se avaliar a facilidade de adaptar aplicações Web existentes para utilização do sistema Comanche. Para tal, usou-se uma aplicação já existente antes do início do desenvolvimento. Esta aplicação é um jogo no qual os utilizadores devem tentar decifrar uma mensagem cifrada usando criptografia clássica – cifras de César e cifras de substituição. A aplicação FCT-CIA, que está permanentemente a executar em http://di196.di.fct.unl.pt:9657/expofct-cia/cia.html, é composta por páginas HTML, Java Servlets, páginas JSP, ficheiros de texto, imagens e ficheiros CSS.

A migração sem qualquer alteração funciona correctamente com a excepção de um caso. A aplicação FCT-CIA escreve os dados referentes aos resultados num ficheiro localizado numa directoria específica. Devido ao isolamento imposto, esta operação não é permitida e os dados não eram gravados. Para solucionar esta situação, foi necessário alterar a localização de gravação do ficheiro, o que foi bastante simples recorrendo ao repositório do sistema Comanche. Alternativamente, poder-se-ia ter decidido manter os resultados no servidor, fazendo a invocação dessa operação no servidor.

A segunda aplicação desenvolvida, ModifyImages, é uma aplicação de transformação que altera imagens JPEG, GIF e PNG por imagens retiradas de um servidor. Com esta aplicação

pretendia-se demonstrar a possibilidade do sistema alterar as respostas devolvidas pelas aplicações finais e pela Proxy, obtidas de servidores finais. A aplicação faz uso do módulo de sincronização do repositório para obter e manter as imagens de substituição actualizadas. Para testar esta aplicação, associou-se a mesma a várias expressões, nomeadamente às expressões "fct" e "google.pt". Verificou-se que a aplicação conseguia efectivamente alterar as imagens de acordo com as regras definidas no ficheiro de configuração. Na figura 5.1, apresenta-se a página inicial da Google com a imagem modificada.

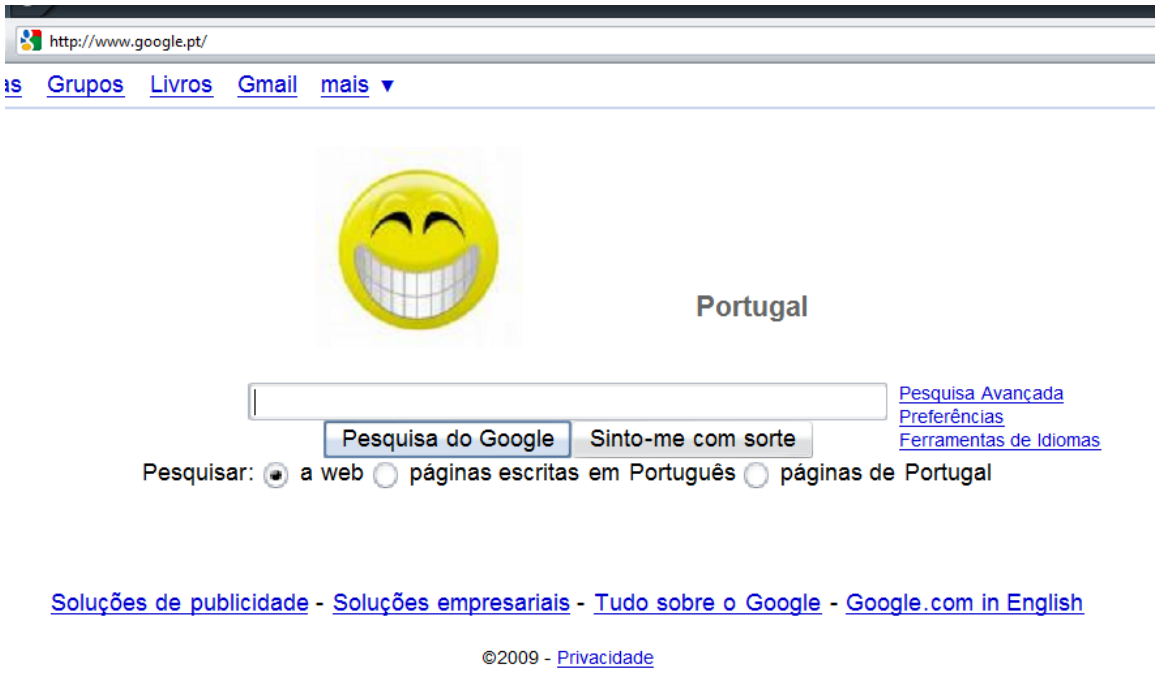

Figura 5.1 – Página www.google.pt, após processada pela aplicação de transformação ModifyImages.

A terceira aplicação pretendia mostrar a possibilidade de combinar aplicações finais e de transformação/observação para fornecer um serviço adicional face à navegação normal dum utilizador na Web.

Assim, decidiu-se implementar uma aplicação que gerisse os artigos visitados no Portal ACM. Esta aplicação, composta por uma aplicação final e uma aplicação de observação, permite manter e consultar informação sobre os artigos visitados, quais destes foram realmente acedidos (abrindo o PDF) e a informação BibTex associada.

A aplicação de observação é associada às páginas com artigos da ACM e aos pedidos de acesso aos ficheiros PDF, através das expressões “portal.acm.org/citation.cfm” e “portal.acm.org/ft_gateway.cfm”. A partir do conteúdo da página, extrai-se o endereço do ficheiro PDF e o endereço da página de bibtex. O conteúdo da página bibtex é imediatamente acedido e guardado localmente, para poder ser mostrado sem haver um novo acesso ao servidor. Estes dados são guardados num ficheiro, que é depois acedido pela aplicação final.

A aplicação final, constituída por Java Servlets, permite duas funcionalidades. A primeira, aceder aos artigos consultados, incluindo informação sobre se estes foram acedidos, a informação Bibtex e a data do último acesso. A segunda funcionalidade permite procurar entradas, por palavras no título, e/ou informação sobre acesso aos ficheiros. O acesso aos dados é feito via o repositório da aplicação de observação, que é passado como parâmetro para a aplicação final quando esta é instalada. Se a aplicação de observação não se encontrar no sistema, esta é instalada. Na figura 5.2 apresenta-se a página inicial desta aplicação final, após alguns acessos a artigos.

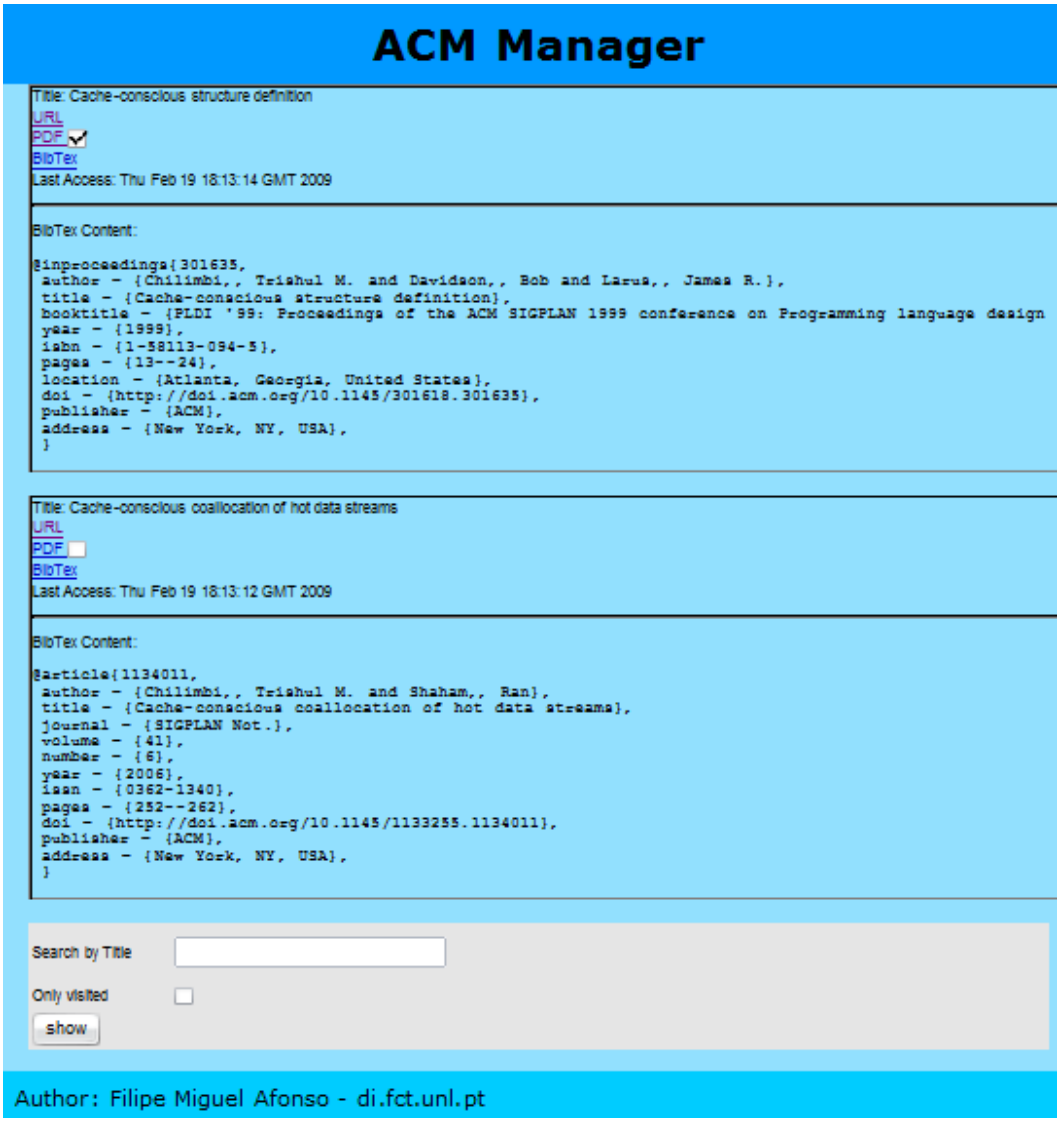

Figura 5.2 – Página inicial da Aplicação Final ACM.

Na Introdução foi mencionada a possibilidade da implementação de *mashup* sobre conteúdos HTML. Para verificar a possibilidade de usar o sistema Comanche para implementar esta funcionalidade, desenvolveu-se uma aplicação final de agregação de RSS Feeds. RSS Feeds são fontes estruturadas de informação que permitem o acesso às últimas actualizações de um conteúdo. O RSS é um documento XML com um formato bem definido, sendo os RSS Feeds utilizados generalizadamente na publicação das últimas novidades de blogs, sites de notícias, etc. A sua utilização tem-se generalizado nos últimos anos.

A aplicação final desenvolvida, composta por dois Java Servlets, permite que endereços RSS sejam adicionados e o seu conteúdo visualizado. É disponibilizada a opção de manter os endereços introduzidos sincronizados, sendo necessário indicar a periodicidade

O conteúdo dos endereços é sempre validado por um XML Schema que irá indicar se o formato do documento XML está correcto. Após a validação, os conteúdos dos vários endereços são combinados para formar um único documento XML. A transformação do documento XML para um formato HTML é feita usando um ficheiro XSLT, apresentado em detalhe no anexo 1. O resultado produzido será parte da página de visualização. A opção de sincronização permite que se aceda aos conteúdos em períodos de desconexão, minimizando a desactualização dos mesmos.

Na figura 5.3 apresenta-se a página de visualização da aplicação final, com dois RSS Feeds adicionados, neste caso relativos aos sites http://rss.cnn.com/rss/cnn_topstories e http://rss.news.yahoo.com/rss/topstories.

Figura 5.3 – Página de visualização de conteúdos da aplicação final RSSFeed.

Como parte de uma bolsa de introdução à investigação, foi proposto ao aluno João Moreno, da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, que desenvolvesse algumas aplicações para o sistema. Foi implementada uma aplicação Web de transformação que insere nos ficheiros HTML código JavaScript que permite fornecer funcionalidades adicionais (semelhante ao Ubiquity dos Mozilla Labs [25]). Neste momento, apenas existe uma nova funcionalidade, a tradução de texto. Na Figura 5.4 apresenta-se uma página alterada por esta aplicação de transformação, com a tradução do texto seleccionado.

As aplicações implementadas permitem demonstrar a grande versatilidade do sistema, permitindo implementar novos serviços nos clientes.

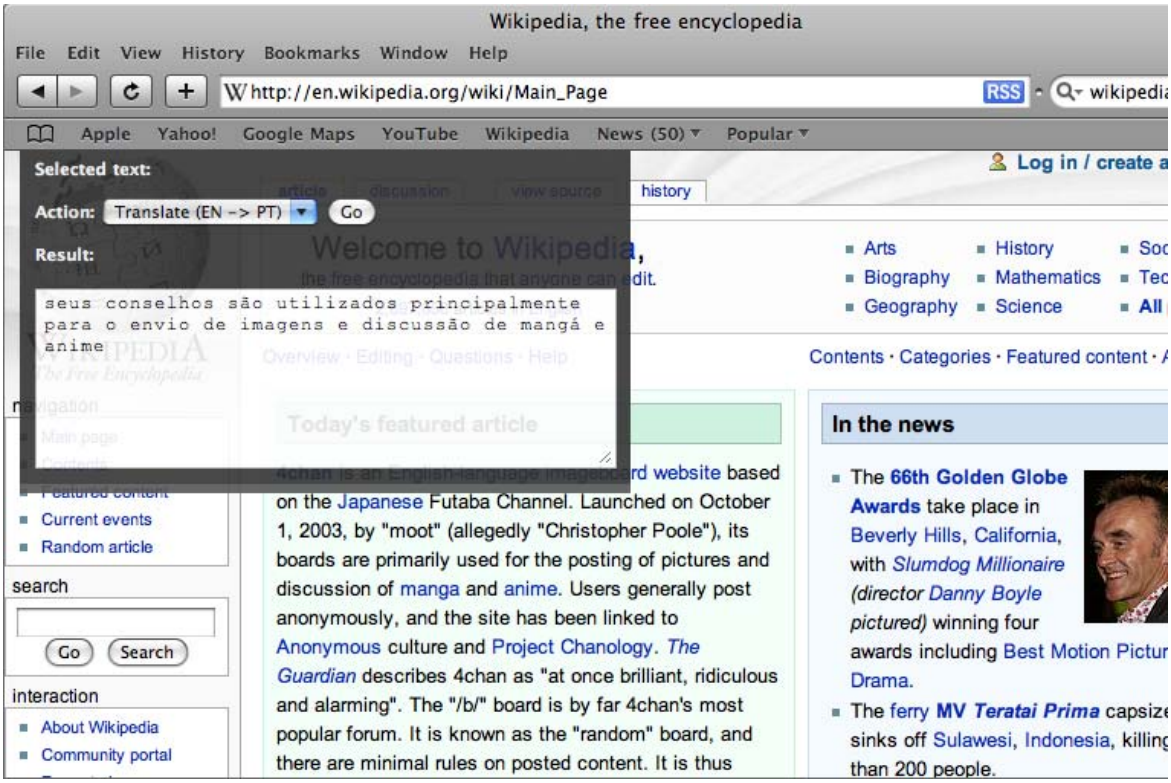

Figura 5.4 – Página alterada com a execução da aplicação de transformação Ubiquity.

5.2 Avaliação quantitativa

A avaliação quantitativa focou-se em dois aspectos: desempenho ao encaminhar pedidos e eficiência ao processar pedidos localmente. Como tal, os resultados destes testes estão intrinsecamente ligados com as características da máquina usada e da ligação no acesso à Web. A ligação usada durante os testes, por cabo, possuía uma velocidade máxima de download de 18

megabits por segundo e uma velocidade máxima de upload de 1 megabit por segundo. O computador usado possuía um processador Intel Dual Core T2130, a 1,86 GHz, e 2 Gigabytes de memória RAM.

A performance no encaminhamento diz respeito ao custo adicional imposto pelo uso do sistema Comanche ao enviar pedidos para servidores externos, isto é, o custo adicional na utilização normal da Web. Para avaliar este custo, foram utilizados 4 ficheiros, com os tamanhos de 10kB, 100kB, 437kB e 3MB. Estes ficheiros foram alojados no servidor pessoa.fct.unl.pt, com um RTT de 30 milissegundos, obtido usando o Ping. Para a recolha dos resultados foi criado um programa em Java que fazia os pedidos HTTP recorrendo a Sockets Java.

Os valores apresentados são os tempos médios de resposta ao pedido, calculados com base em 50 pedidos para cada ficheiro, excluindo os 5 melhores e os 5 piores tempos. O sistema foi testado sem aplicações e com aplicações instaladas no sistema, quando estas não estejam associadas com o pedido. Os resultados obtidos estão na figura 5.5.

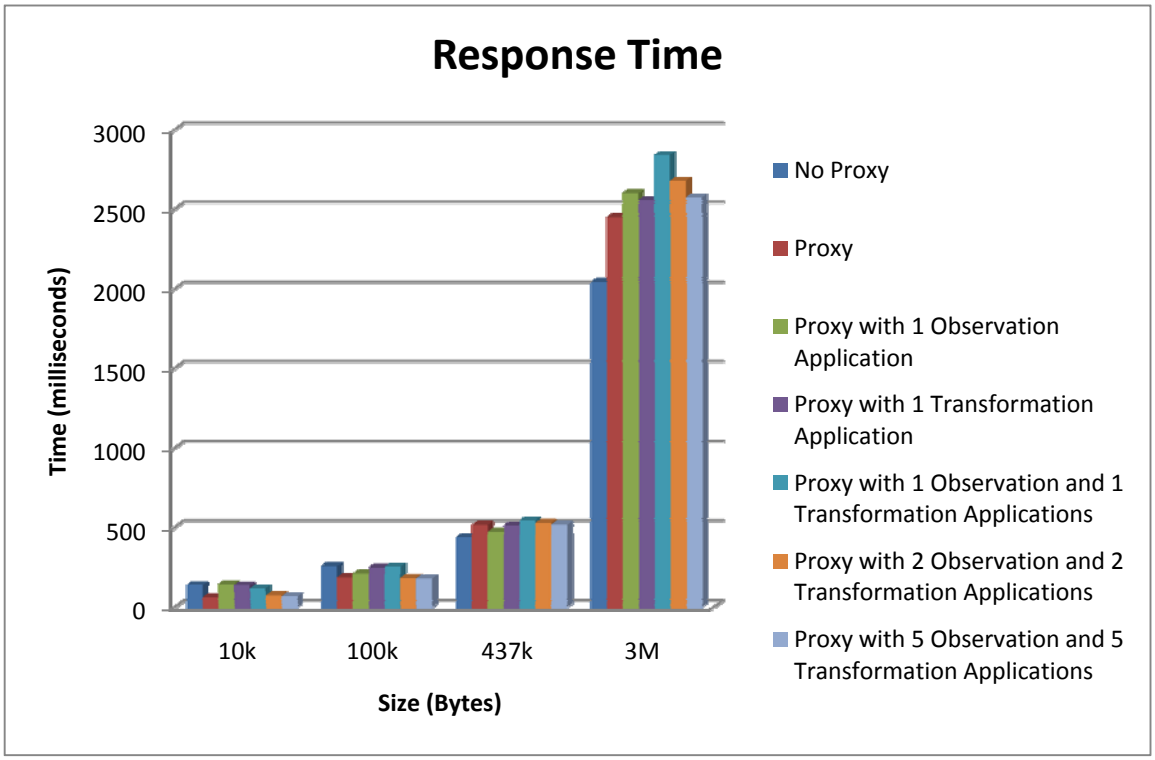

Figura 5.5 – Gráfico com os resultados dos pedidos a ficheiros.

Os resultados obtidos para os testes de performance mostram que o Comanche não introduz um custo adicional excessivo. A maior diferença obteve-se no acesso ao ficheiro de 3

MegaBytes, onde o tempo de resposta ao pedido com *proxy* foi, em média, mais 500 milissegundos (20%) que o acesso directo ao mesmo ficheiro. A diferença não se altera significativamente quando se encontram aplicações instaladas. Estes resultados são justificados pela actual implementação do sistema, em que a *proxy* apenas encaminha a resposta para o cliente após a obter completamente. Assim, é normal que o aumento da dimensão dos ficheiros cause um aumento do atraso na obtenção da resposta. No futuro pretende-se avaliar esta opção de desenho, embora a mesma tenha sido tomada de forma a simplificar o processamento das respostas pelas aplicações de transformação e observação.

Com ficheiros mais pequenos, a diferença é desprezável. Estes resultados mostram também que o custo adicional é uma pequena fracção da latência total para obtenção de uma resposta na maior parte das situações.

Os resultados mostram ainda que o custo adicional não se altera com o número de aplicações não relacionadas com o pedido instalado.

Para testar a eficiência ao executar aplicações finais e a possibilidade de diminuir os tempos de resposta a um pedido, utilizou-se a aplicação FCT-CIA. A comparação foi efectuada através do acesso à aplicação, a executar no servidor di196.fct.unl.pt e localmente no sistema Comanche. Os resultados, apresentados na figura 5.6, são os tempos médios de resposta ao pedido de validação de uma chave, fazendo o pedido directamente, encaminhando-o a partir do sistema e executando-o localmente. Cada pedido foi repetido 30 vezes, tendo sido eliminados os 5 melhores e os 5 piores de forma a eliminarem-se eventuais atrasos excessivos introduzidos pela rede ou quando esta apresentava um desempenho acima do normal. O RTT obtido para este servidor foi de 11 milissegundos, usando novamente o Ping. Os valores foram obtidos recorrendo ao *browser Firefox*, versão 3.0.6, com o *plugin Extended Statusbar*, versão 1.5.3, que permite medir o tempo de obtenção da página associada a um URL.

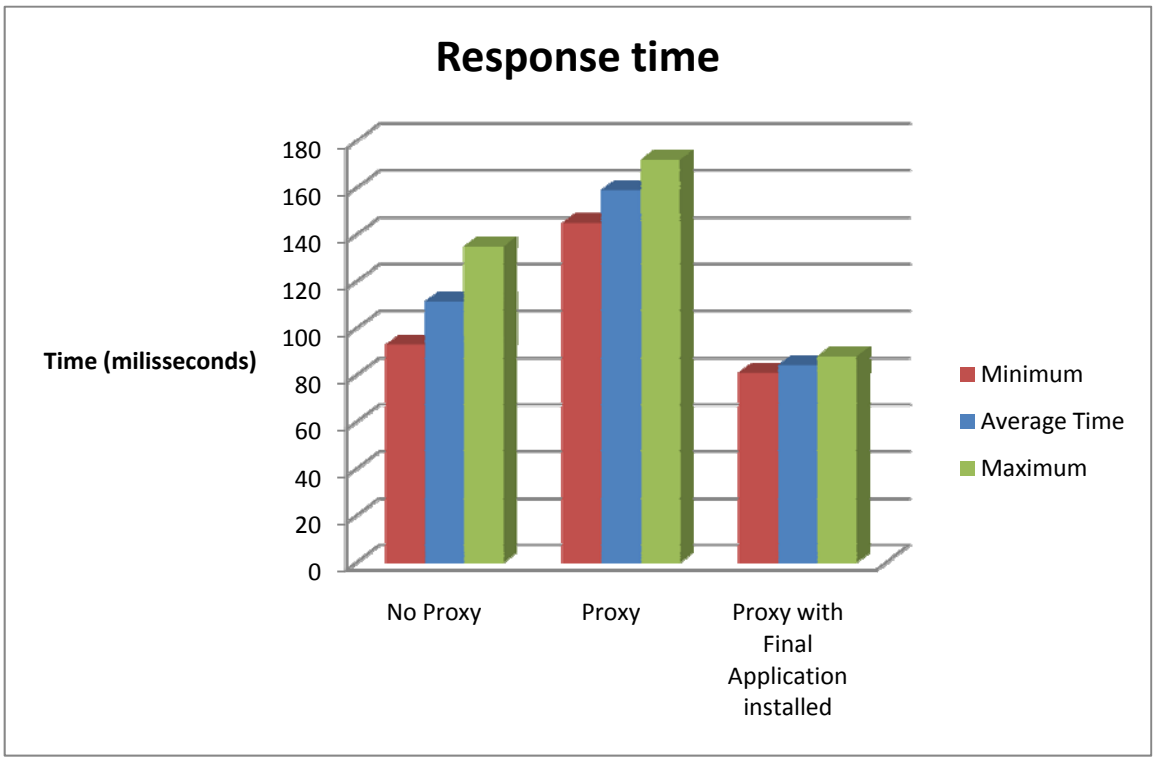

Figura 5.6 – Gráfico com o tempo médio de resposta a um pedido.

Os resultados mostram que a execução da aplicação localmente produz tempos de resposta menores. Como o pedido é processado localmente, o tempo de encaminhamento na rede é nulo. Adicionalmente, consegue-se explorar a maior capacidade/menor carga da máquina local face ao servidor final. Outro resultado interessante é que os tempos obtidos quando se executava o pedido localmente foram muito uniformes, ao contrário dos pedidos feitos directamente ao servidor ou fazendo o encaminhamento do mesmo através da *proxy* do Comanche.

Também se pode verificar, tal como anteriormente, que o encaminhamento através da *proxy* introduz algum custo adicional. No entanto, mesmo tendo em conta este custo adicional, foi possível obter um melhor desempenho face a aceder directamente ao servidor.

6. Conclusões

6.1 Avaliação Critica

A Web é, provavelmente, o serviço mais utilizado na Internet, estando disponíveis as mais variadas aplicações e serviços (serviços bancários, lojas *online*). Para que estas aplicações funcionem de forma satisfatória, é importante que o desempenho do sistema seja elevado. Adicionalmente, devido à sua utilização generalizada, é importante simplificar a criação de novos serviços e aplicações.

Neste trabalho, implementou-se um sistema baseado numa *proxy* HTTP instalada no cliente que permite executar *Java Web Applications* localmente. Assim é possível aproveitar os recursos computacionais dos clientes para processar os pedidos que teriam de ser tratados num servidor remoto. Desta forma, o servidor apenas atende os pedidos que não podem ser satisfeitos localmente, aumentando a escalabilidade do servidor e da aplicação. A execução local do pedido é, para o utilizador, completamente transparente, com excepção da possível diminuição do tempo de resposta. Os testes realizados confirmam esta possibilidade num cenário concreto. Adicionalmente, torna-se mais simples criar novas aplicações e serviços, porque o suporte necessário no servidor é mais reduzido.

O sistema suporta três tipos de aplicações que podem executar localmente: aplicações finais, aplicações de transformação e aplicações de observação.

As aplicações Web Finais têm como objectivo o processamento, total ou parcial, do pedido, sendo gerada uma resposta.

As aplicações Web de Transformação permitem interceptar o pedido e a resposta, podendo alterar os mesmos. Este tipo de aplicações pode ser usado para fornecer serviços adicionais, como filtragem de conteúdos. No entanto, a sua execução tem impacto no tempo de resposta, pois têm de ser executadas ordenadamente. A ordem de execução destas aplicações é determinada pelo valor de altitude de cada aplicação.

As aplicações Web de observação permitem a recolha de informação a partir dos pedidos efectuados pelos utilizadores e respectivas respostas. A execução assíncrona das aplicações de observação permite a análise da resposta sem que exista impacto no tempo de resposta.

O sistema inclui um repositório que permite às aplicações armazenar informação de forma segura, abstraindo-se completamente do sistema de ficheiros local. O sistema permite que diferentes aplicações partilhem o acesso a uma mesma área, sendo uma maneira simples de partilhar informação.

O módulo de sincronização permite que recursos acessíveis pelo protocolo HTTP possam ser guardados de forma sincronizada. Quando combinada com a execução local das aplicações, esta funcionalidade permite diminuir a dependência dos servidores e mesmo suportar a operação em situações de desconexão.

Como foi indicado, o Comanche introduz restrições de segurança às aplicações instaladas, nomeadamente no acesso ao sistema de ficheiros e à rede. Esta propriedade é fundamental para garantir a segurança dos clientes no carregamento dinâmico de código a partir de servidores.

As aplicações desenvolvidas permitiram demonstrar que o sistema permite alcançar os objectivos definidos. Assim, é possível criar aplicações que executam nos clientes de forma simples. Estas aplicações podem fornecer serviços aos utilizadores que seriam difíceis de fornecer a partir dum servidor. Adicionalmente, os resultados de desempenho mostram que o *overhead* para acessos não tratados localmente é aceitável.

6.2 Trabalho Futuro

O sistema desenvolvido apresenta algumas limitações que podem ser tratadas no futuro. Nesta secção apresentam-se as que parecem ser mais importantes.

Na instalação das aplicações, o utilizador é sempre chamado a confirmar esta instalação. No futuro seria possível definir condições nas quais as aplicações fossem automaticamente instaladas, como por exemplo, as aplicações assinadas que não peçam acesso a recursos adicionais. Desta forma o utilizador não teria de ser incomodado sempre que houvesse necessidade de instalar uma aplicação.

A segurança é muito importante para os utilizadores, principalmente quando se executa código do qual não se possui qualquer tipo de conhecimento. A implementação de restrições a outros recursos irá aumentar o nível de confiança que um utilizador terá quando instalar uma aplicação, tornando o sistema mais apelativo.

Até à data, o Comanche apenas suporta o protocolo HTTP. Este suporte deverá ser estendido a outros protocolos, como HTTPS, permitindo assim aumentar o número de recursos a que se pode aceder e de pedidos que poderão ser processados localmente.

Finalmente, deve ser feito um melhor estudo sobre possíveis aplicações. As aplicações desenvolvidas são relativamente simples. No entanto as soluções desenvolvidas para o sistema podem ser mais complexas. Um exemplo seria uma aplicação que permita a colaboração entre utilizadores sem recorrer a um servidor. Por exemplo, uma aplicação Web final que permita que os utilizadores partilhem *links* interessantes, estabelecendo uma rede *peer-to-peer* entre eles.

7. Anexos

Anexo 1 – XSLT usado na aplicação RSSFeed

```
<?xml version="1.0"?>
<!-- Edited by Lee Sykes DNN Creative Magazine http://www.dnncreative.com -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"/>
<xsl:param name="TITLE"/>
<xsl:template match="rss">

  <!-- Do not show channel image -->
  <xsl:for-each select="channel/item">
    <br>
      <!-- to open links in a new window, change target="_main" to
target="_new" -->
    <strong><a href="{link}" target="_main"><xsl:value-of
select="title"/></a></strong><br></br>
    <!-- only display markup for description if it's present -->
    <xsl:value-of disable-output-escaping="yes" select="description"/>
    </br>
    <br></br>
  </xsl:for-each>
</xsl:template>

<xsl:template match="description">
  <br>
    <xsl:value-of select="."/>
  </br>
</xsl:template>
</xsl:stylesheet>
```

8. Bibliografia

[1] Rabinovich, Michael, e Oliver Spatscheck. *Web Caching and Replication*. Addison Wesley, 2002.

[2] WidSets - Mobilize Your Web . http://www.widsets.com (acedido em Abril 05, 2008).

[3] Google Gears. http://gears.google.com (acedido em Abril 24, 2008).

[4] Adobe - Flash Lite. http://www.adobe.com/products/flashlite (acedido em Abril 30, 2008).

[5] Zhang, Jin, Pei Cao, e Kevin Beach. "Active Cache: Caching dynamic contents on the web." *Proc. of Middleware'98*, 1998, 373-388.

[6] Kermarrec, Anne, Ihor Kuz, Maarten van Steen, and Andrew Tanenbaum. "A Framework for Consistent, Replicated Web Objects." *Proceedings of the The 18th International Conference on Distributed Computing Systems*, 1998, 276-284.

[7] Armstrong, Trevor, Olivier Trescases, Cristiana Amza, and Eyal de Lara. "Efficient and transparent dynamic content updates for mobile clients." *Proceedings of the 4th international conference on Mobile systems, applications and services*, 2006, 56-68.

[8] Internet Usage World Stats - Internet and Population Statistics. http://www.internetworldstats.com/stats.htm (acedido em Maio 05, 2008).

[9] Berners-Lee, T., R. Fielding, e H. Frystyk. *Hypertext Transfer Protocol - Http/1.0.*, RFC 1945, 1996.

[10] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., e T. Berners-Lee. *Hypertext Transfer Protocol -- Http/1.1*. RFC 2616, 1999.

[11] Duska, Bradley, David Marwood, and Michael Joseph Feeley. "The Measured Access Characteristics of World-Wide-Web Client Proxy Caches." *Proceedings of the USENIX Symposium on internet Technologies and Systems on USENIX Symposium on internet Technologies and Systems*, 1997, 3.

[12] Akamai: The Leader in Web Application Acceleration and Performance Management, Streaming Media Services and Content Delivery. http://www.akamai.com/html/technology/edgeplatform.html (acedido em Maio 07, 2008).

[13] Enterprise Business Networks and Internet, Phone, and Data Solutions | AT&T. http://www.business.att.com/enterprise/Family/eb_hosting_storage_and_it/eb_intelligen_content_distribution/ (acedido em Maio 07, 2008).

[14] Noble, Brian. "System Support for Mobile, Adaptive Applications." *IEEE Personal Communications*, 2000.

[15] Elson, J. e Cerpa, A. *Internet Content Adaptation Protocol (Icap)*. RFC 3507, 2003.

[16] Mashup - Wikipédia, a enciclopédia livre. http://pt.wikipedia.org/wiki/Mashup (acedido em Maio 14, 2008).

[17] Yuan, Chun, Zhigang Hua, e Zheng Zhang. "Proxy+: Simple Proxy Augmentation for Dynamic Content Processing." *Web Content Caching and Distribution: Proceedings of the 8th international Workshop*, 2004, 91-108.

[18] Wills, Craig E., e Mikhail Mikhailov. "Examining the cacheability of user-requested Web resources." *Proceedings of the 4rd Web Caching and Content Delivery Workshop*, 2009.

[19] Challenger, Jim, Paul Dantzig, Arun Iyengar, e Karen Witting. "A fragment-based approach for efficiently creating dynamic web content." *ACM Transactions on Internet Technology (TOIT)*, 2005, 5, 2, 359-389.

[20] Housel, Barron, George Samaras, e David Lindquist. "WebExpress: A client/intercept based system for optimizing Web browsing in a wireless environment ." *Proceedings of the 2nd Annual international Conference on Mobile Computing and Networking (MobiCom '96)*, 1996, 108-116.

[21] Rabinovich, Michael, Zhen Xiao, Fred Douglis, and Chuck Kalmanek. "Moving Edge-Side Includes to the Real Edge: the Clients." *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, 2003, 12.

[22] Phatak, Shirish Hemant, V. Esakki, Badrinath R. Badrinath, e L. Iftode. "Web&: An Architecture for Non-interactive Web." *Proceedings of the Second IEEE Workshop on Internet Applications (wiapp '01)*, 2001, 104.

[23] Jetty WebServer. http://www.mortbay.org/jetty/ (acedido em Outubro 2, 2008).

[24] DB Apache Project - Welcome to DB. http://db.apache.org/derby/ (acedido em Novembro 11, 2008).

[25] Mozilla Labs Ubiquity. http://labs.mozilla.com/projects/ubiquity/ (acedido em Janeiro 8, 2009).

[26] Apache Tomcat. http://tomcat.apache.org/ (acedido em Janeiro 10, 2009).

[27] NekoHTML. http://nekohtml.sourceforge.net/ (acedido em Janeiro 13, 2009).