Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia da Universidade de Coimbra

9ticks

Event Processing Engine and Crawler

Relatório Final de Estágio

Mestrado em Engenharia Informática

4 de Setembro de 2009

Aluno
Rafael Marmelo
rjmm@student.dei.uc.pt
nr. 501070595

Orientadores
Prof. Doutor Paulo Marques
Prof. Doutor Pedro Bizarro

Resumo

Na Internet existem milhares de fontes de informação. Esta informação é actualizada frequentemente, estruturada e tem, em muitos casos, a data e hora de criação ou actualização. Actualmente, esta informação não está a ser recolhida, perdendo-se definitivamente com o passar do tempo. Este documento descreve um sistema, *9ticks*, que recolhe, interpreta, armazena, consulta, agrega e visualiza a informação outrora efémera e dispersa pela Internet.

Palavras-chave:

1. Internet
2. Recolha de informação da Internet
3. Eventos
4. Processamento de eventos
5. *Data mining*
6. Base de dados

Conteúdo

Lista de Símbolos e Abreviaturas

Símbolo	**Significado**
AJAX	Asynchronous Javascript And XML
CORBA	Common Object Request Broker Architecture
CSV	Comma-separated Values
EJB	Enterprise JavaBeans
ESB	Enterprise Service Bus
HTML	HyperText Markup Language
IPC	Inter-process Communication
ISO	International Organization for Standardization
JEE	Java Enterprise Edition
JME	Java Micro Edition
JMS	Java Message Service
JSON	JavaScript Object Notation
JSON-RPC	JSON Remote Procedure Call
JVM	Java Virtual Machine
LRU	Least Recently Used
MIME	Multipurpose Internet Mail Extensions
MOM	Message Oriented Middleware
MSHTML	Microsoft Hypertext Mark Up Language
MSMQ	Microsoft Message Queuing
MVC	Model-View-Controller
RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RRDtool	Round Robin Database tool
RSS	Really Simple Syndication
SMS	Short Messaging Service
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
tmpfs	Temporary File System
URL	Uniform Resource Locator
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language
XML-RPC	XML Remote Procedure Calling
XPath	XML Path Language
XQuery	XML Query Language
XSD	XML Schema Definition

Símbolo	**Significado**
XSLT	eXtensible Stylesheet Language Transformation
XSRF	Cross-Site Request Forgery
XSS	Cross-Site Scripting
XUL	XML User Interface Language
YAML	Yet Another Multicolumn Layout

"Basically, our goal is to organize the world's information and to make it universally accessible and useful. That's our mission."

Larry Page

1 Introdução

O presente trabalho foi desenvolvido no âmbito da cadeira de Dissertação / Estágio do Mestrado em Engenharia Informática do Departamento de Engenharia Informática (DEI) da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (FCTUC). O mesmo teve uma duração de aproximadamente 1 ano, entre 1 de Setembro de 2008 e 4 de Setembro de 2009.

O estágio teve como entidade acolhedora o DEI-FCTUC, em particular o grupo *Software and Systems Engineering* (SSE), tendo sido desenvolvido no laboratório E.5.4 do DEI e orientado pelos Professores Paulo Marques e Pedro Bizarro.

A equipa de trabalho do *9ticks* foi constituída apenas pelo estagiário, não menosprezando o importante *feedback* que foi recebido quer dos orientadores quer dos restantes elementos do grupo SSE, em particular os que partilharam esse mesmo laboratório.

O estágio foi suportado pelo projecto BiCEP, financiado por uma bolsa Marie Curie para Reinserção Internacional (FP6 MIR6-CT-2007-46478).

1.1 Motivação

Existem na Internet milhares de fontes de informação com actualizações muito frequentes. Na sua maioria, a informação é apresentada em páginas *web*, com estruturas estáveis e conteúdos actualizados frequentemente. Fala-se de informação das mais variadíssimas áreas como, por exemplo, valores de acções da bolsa, resultados desportivos, valores meteorológicos, estado de voos e estado de trânsito. Em todas estas situações estamos perante informação com estrutura bem definidas, em que os diversos valores obedecem à priori a determinados tipos de dados.

No dia-a-dia, o utilizador que queira estar actualizado acerca desta informação, necessita monitorizar as várias páginas onde ela reside. Actualmente, existem soluções que permitem visualizar vários pedaços de informação, distintos, numa só página: os *mashups*. Falamos de soluções como *iGoogle* [39], *NetVibes* [56], *PageFlakes* [59] e muitas outras (Capítulo 2). No entanto, estas aplicações têm por base um leque bem definido de fontes de informação adicionáveis ao *dashboard* (*widgets*). Estas apresentam-se pouco personalizáveis e, em vários casos, repletas de informação que em nada interessa ao utilizador.

O utilizador que queira seguir a informação contida numa página para a qual não exista nenhum *widget*, depara-se com duas hipóteses: *a)* tem conhecimentos de programação suficientes para o criar ou, na maioria dos casos, *b)* não pode visualizar o que pretende.

Se o utilizador sabe onde a informação se encontra, porque não existe uma forma simples e intuitiva de ele identificar as porções da página de seu interesse e, posteriormente, permitir a sua adição a um *dashboard* pessoal?

Por outro lado, a Internet é efémera. No presente, os motores de pesquisa percorrem a *web*, indexam e armazenam a última versão de cada página em bruto, sem qualquer análise estruturada, e permitem ao público efectuar pesquisas sobre esses dados. A informação que ontem, no mês passado ou há dez anos atrás se encontrava nas páginas *web* está, muito provavelmente, completamente perdida ou inserida em bases de dados às quais o utilizador comum não tem acesso. A informação existiu e perdeu-se com o passar do tempo.

Existem actualmente algumas formas de olhar para o passado, como por exemplo *a) Internet Archive* [4], *b) cache* do *Google* ou, de uma forma mais interessante, *c) google trends* [75]. Todos eles armazenam páginas em bruto e indexam-nas sem qualquer preocupação com a sua estrutura ou tipos de dados. Realizam esta recolha todos os meses, dias ou, caso se trate de páginas com actualizações mais frequentes, várias vezes ao dia. Mas, o que acontece com as páginas que apresentam informação com mudança muito frequente, por exemplo, várias vezes no mesmo minuto? Não existe nenhum sistema que recolha, armazene e interprete essa informação efémera tendo em conta os seus tipos de dados, possibilitando a sua consulta pelo utilizador.

1.2 Contextualização

Para a correcta compreensão do documento é importante explicitar alguns termos fundamentais:

Fonte de Eventos – Qualquer local na *web* que disponibilize informação actualizada frequentemente. Por exemplo, páginas *web* e *feeds* podem ser considerados fontes de eventos. Pode também ser chamada de fonte de informação, fonte de dados ou, simplesmente, fonte.

Evento – Qualquer actualização ao conteúdo de uma fonte. Por exemplo, a adição de uma notícia a um *feed* ou a actualização de um resultado desportivo continuem eventos.

Fonte Predefinida – Fonte definida à partida pelo sistema. O utilizador pode subscrevê-la directamente a partir do sistema.

Fonte Personalizada – Fonte definida pelo utilizador. O utilizador, através do uso de uma ferramenta adicional, identifica a origem da informação e define quais os elementos nela são alvo de actualização. Após a sua criação, qualquer utilizador pode subscrevê-la a partir do sistema.

1.3 Visão

O *9ticks* consiste, visualmente, numa página *web* que pode ser personalizada pelo utilizador. Este espaço, usualmente chamado *dashboard* ou painel, pode ser composto através da adição de fontes de informação.

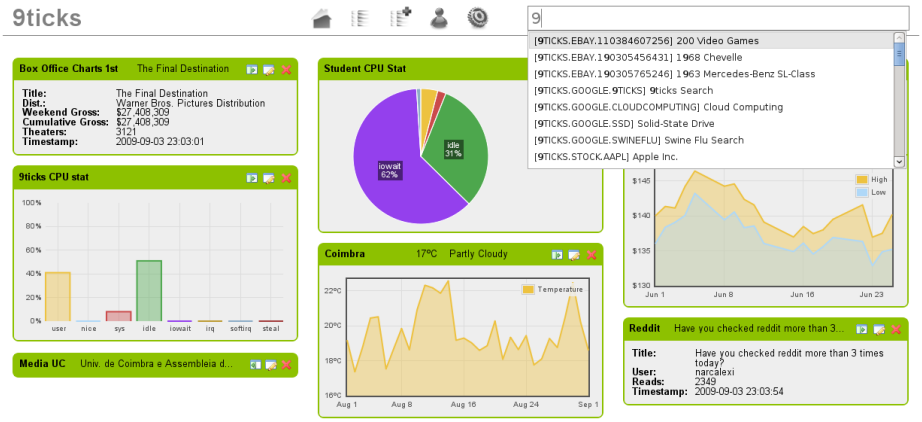


Figura 1.1: Página *web* do *9ticks*

A Figura 1.1 exemplifica a nossa visão para o *9ticks*. No sentido dos ponteiros do relógio, iniciando no canto superior esquerdo, temos o filme que se encontra em primeira posição no *box office* americano, a utilização do processador do servidor dos alunos do nosso departamento, os valores máximos e mínimos ao longo do tempo das acções da *Apple Inc.*, a notícia mais popular no *Reddit*, a temperatura em Londres num intervalo de seis semanas, o último *twit* sobre a Universidade de Coimbra e a utilização do processador do servidor principal do *9ticks*.

O utilizador tem a possibilidade de adicionar novas fontes de informação ao *dashboard* bastando para isso utilizar a caixa de texto no canto superior direito. Trata-se de uma pesquisa por palavras-chave, onde o utilizador é presenteado com uma lista de sugestões, entre as quais deve escolher a que pretende.

Existem centenas de aplicações que partilham o mesmo conceito. No entanto, não é possível adicionar fontes que não sejam suportadas à partida pela aplicação. As opções do utilizador ficam restritas logo à partida.

Por exemplo, existem diversas aplicações que permitem acompanhar em tempo real os resultados da primeira liga do futebol inglês mas não existe nen-

huma que siga a segunda divisão do futebol português, apesar de existirem páginas *web* em que essa informação é disponibilizada em tempo real.

O desafio encontra-se em desenvolver um sistema genérico suficiente ao ponto de permitir extrair pedaços de informação de qualquer página *web*, ou seja, permitir a criação de fontes de informação personalizadas.

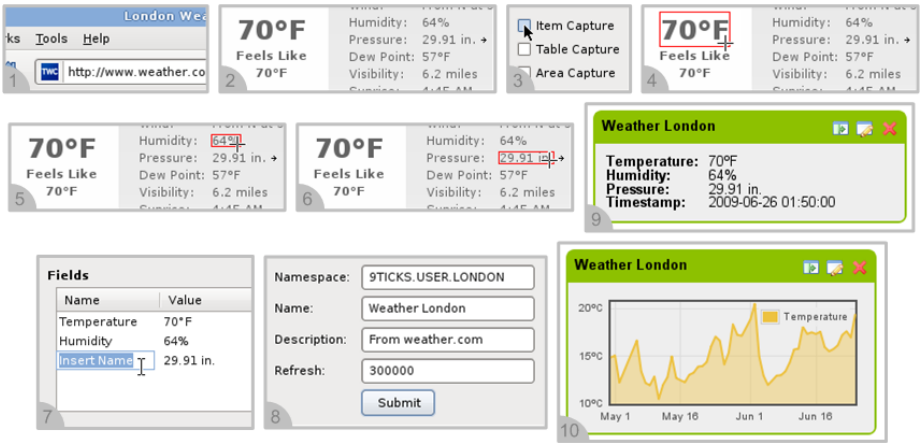


Figura 1.2: Definição de novas fontes

Por exemplo, assumamos que o utilizador deseja acompanhar o estado meteorológico em Londres[1]. Na nossa visão (Figura 1.2), *1)* o utilizador navega até à página *web* de meteorologia de sua eleição e *2)* identifica visualmente onde está a informação que deseja. De seguida abre um *plugin* no seu *browser* e *3)* indica que deseja identificar na página pedaços de informação. De forma automática a página reage ao movimento do rato, identificando quais os elementos possíveis de seleccionar. O utilizador selecciona os elementos do seu interesse: *4)* temperatura, *5)* humidade e *6)* pressão atmosférica. Após terminar a selecção, o utilizador *7)* verifica se os valores que o *plugin* reconheceu estão correctos e designa um nome para cada um deles. Por fim, apenas tem que *8)* preencher a informação que identifique a nova fonte de informação e submetê-la para o nosso sistema. A partir desse momento, é possível *9)* adicioná-la ao *dashboard* como se de uma fonte pré-definida se tratasse. Com o passar do tempo, será possível *10)* consultar a progressão histórica, por exemplo, da temperatura.

Outro grande problema das aplicações já existentes, e desafio para o *9ticks*, é não deixarem ao critério do utilizador o estilo e o grau de detalhe com que ele quer visualizar as diferentes fontes de informação. Por exemplo, voltando à Figura 1.1, o gráfico de barras e o circular apresentam o mesmo tipo de informação (utilização do processador) mas de formas distintas.

Outras vezes, o utilizador deseja visualizar a informação de forma minimalista, organizada de forma a permitir ser consultada apenas com um olhar. É visão do *9ticks* permitir o que se exemplifica na Figura 1.3. Vários tipos de informação, completamente diferentes, apresentados de forma estruturada e

[1]Na realidade, existem inúmeras aplicações que suportam isso à partida. Apenas se trata de um exemplo do dia-a-dia para melhor compreensão.

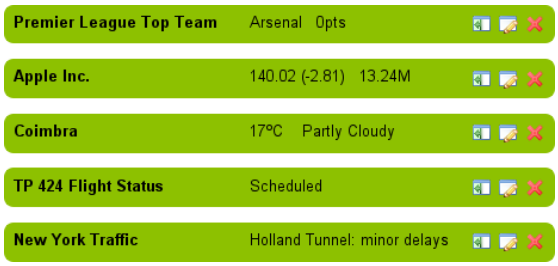


Figura 1.3: Fontes organizadas para leitura rápida

sempre personalizáveis pelo utilizador.

Por fim, pensa-se no *9ticks* como uma plataforma colaborativa, onde cada utilizador pode contribuir com novas fontes de informação e formas de as visualizar. Uma variante de motor de pesquisa com base em dados temporais.

1.4 Objectivos

O objectivo geral deste estágio foi criar uma prova de conceito através do desenvolvimento de um protótipo que recolha, armazene e interprete eventos provenientes de diversas fontes de informação presentes na *web*. Em particular, os objectivos esperados foram os seguintes:

- Levantamento detalhado do estado da arte (Capítulo 2);
- Análise de requisitos detalhada e refinada com o avançar do projecto (Capítulo 3);
- Análise da melhor arquitectura para o sistema, tendo em conta os atributos de qualidade (Capítulo 5);
- Implementação de um protótipo que recolha, armazene e interprete eventos provenientes de diversas fontes de informação (Capítulo 6);
- Elaboração de documentação apropriada (quer ao nível arquitectural como funcional);
- Escrita do relatório final de estágio;
- Escrita de um artigo como resumo do trabalho realizado e sua avaliação. Este poderá ser eventualmente submetido a um fórum apropriado ou publicado como relatório técnico do DEI/FCTUC.

1.5 Planeamento e Gestão

Durante todo o ano lectivo foi seguida uma abordagem iterativa de desenvolvimento, onde em cada ciclo de desenvolvimento (duas semanas) era feito o ponto de situação até ao momento e definidos os requisitos específicos a implementar no ciclo seguinte. Existiram reuniões semanais de acompanhamento do trabalho

estando presentes, sempre que possível, ambos os orientadores.

A distribuição faseada do projecto para o segundo semestre, elaborada e apresentado na defesa intermédia de estágio, encontra-se na Figura 1.4. Notar que a marcação temporal corresponde À semana que nesse dia tem início.

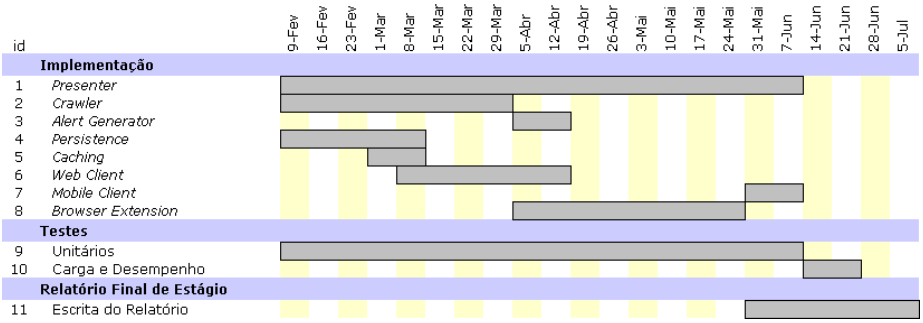


Figura 1.4: Diagrama de *Gantt* previsto

No final do segundo semestre, o diagrama de *Gantt* sofreu algumas alterações (Figura 1.5). De seguida apresentam-se algumas as suas principais diferenças.

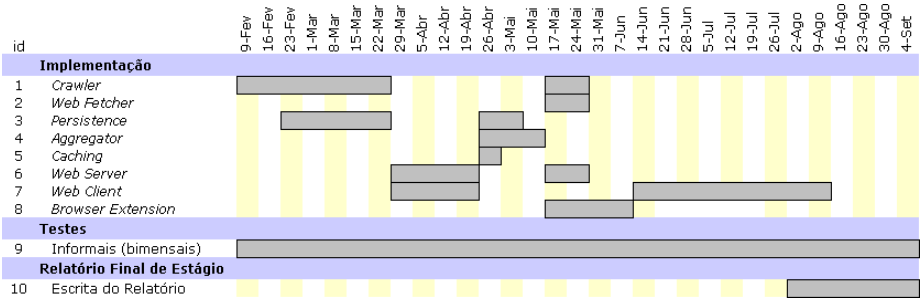


Figura 1.5: Diagrama de *Gantt* efectivo

A diferença mais notória foi a eliminação dos componentes *Presenter*, *Alert Generator* e *Web Client*. Por outro lado, foram adicionados os componentes *Web Fetcher*, *Aggregator* e *Web Server*.

Devido ao estágio consistir numa prova de conceito, os testes unitários, de carga e desempenho foram substituídos por testes informais, ao longo de todo o percurso. Tratando-se de uma aplicação maioritariamente *web*, ela era testada ao longo da semana pelo estagiário e, sempre que possível, os orientadores, debatendo-se os resultados na reunião de acompanhamento semanal.

1.6 Estrutura do Relatório

No Capítulo 2 apresenta-se o resumo do estado da arte relevante para este estágio, descrevendo aplicações similares e tecnologias que poderão servir de

base ao *9ticks*. Notar que a versão completa do estado da arte pode ser encontrada no Anexo D.

No Capítulo 3 são resumidos os requisitos que o sistema necessita de implementar. Apontar ainda que no Anexo E podem ser consultados os requisitos em detalhe.

No Capítulo 4 descreve-se o sistema do ponto de vista do utilizador. É feita uma demonstração com algum detalhe de como funciona a página *web* e a extensão do *web browser*.

No Capítulo 5 introduz-se a arquitectura do *9ticks*, descrevendo sucintamente para que serve e qual a importância de cada um dos seus componentes.

No Capítulo 6 explicitam-se os detalhes de implementação. Indicam-se as escolhas das tecnologias por base e descreve-se como foi implementado cada componente.

No Capítulo 7 faz-se o ponto da situação de situação do *9ticks*, descrevendo os resultados que foram alcançados.

No capítulo 8 descreve-se *roadmap* para a futuro, explicitando o que de mais importante há para fazer.

Finalmente, o Capítulo 9 conclui este relatório.

Nos Anexos A, B e C encontram-se três artigos escritos ao longo do estágio, dois deles tendo sido já publicados.

1.7 Contribuições

O principal objectivo deste estágio foi provar um conceito. Provar que é possível construir um protótipo que cumpre a visão descrita na Secção 1.3. Com base nisso, as principais contribuições foram as seguinte:

1. Dois protótipos funcionais do ponto de vista do utilizador: a página *web* e o *plugin* do *web browser*; Os dois em conjunto permitem solucionar a efemeridade da Internet;

2. Provou-se que as aplicações actuais de *dashboarding* apresentam algumas limitações. Em particular, não permitem criar novas fontes de forma simples nem permitem personalizar a forma como as fontes já existentes são visualizadas;

3. Concluiu-se a ineficácia de algumas ferramentas em casos de uso pontuais. Por exemplo, o facto de sistemas de armazenamento, em particular o *PostgreSQL* e o *HBase*, não estarem preparados para lidar com informação temporal;

4. Deu-se a conhecer o conceito à comunidade científica através da publicação de alguns artigos em conferências (Anexos A e C) e revistas (Anexo B).

"It took us three years to build the NeXT computer. If we'd given customers what they said they wanted, we'd have built a computer they'd have been happy with a year after we spoke to them – not something they'd want now."

Steve Jobs

2
Estado da Arte

Neste capítulo apresenta-se uma versão resumida do estado da arte relevante para o estágio. Notar que a versão completa do estado da arte pode ser consultada no Anexo D.

aplicações similares

As aplicações que apresentam funcionalidades similares às visionadas para o *9ticks* podem ser catalogadas em quatro categorias, abordando-se cada uma em seguida.

custom web pages

A primeira categoria engloba as páginas *web* que utilizam um *dashboard* para apresentam informação. Por exemplo, *Alerts.com* [2], *iGoogle* [39], *Netvibes* [56], *PageFlakes* [59], *My Yahoo!* [55] e *Webwag*[84].

Todas elas permitem ao utilizador compor o seu *dashboard* através da adição de *widgets*. No entanto, todas elas se encontram restritas ao leque de fontes de informação predefinidas pelo sistema. Após adicionadas quase nunca é possível personalizá-las. Ou seja, a mesma fonte é vista por todos os utilizadores de forma igual. Realçar ainda que o *Netvibes* é o único que permite adicionar ao *dashboard clippings* de qualquer página *web* (ver próxima categoria).

Todas as aplicações fazem uso extensivo de *JavaScript* e *AJAX* com vista a melhorar a interacção com o utilizador.

clipping the web

A segunda categoria refere-se às aplicações que permitem fazer *clipping* de páginas *web*. Ou seja, definir pedaços da página que contêm informação de interesse e apenas visualizar essa porção. Existem duas aplicações que merecem ser destacadas a este nível: *Web Slices* [82] e *Web Clip* [80].

Ambas correm fora do ambiente *web* e nenhuma delas dá importância ao conteúdo do recorte, permitindo simplesmente visualizá-lo. A primeira foi introduzida com o *Mac OS X Leopard* e a segunda com o *Internet Explorer 8*. Esta última apenas suportas páginas preparadas explicitamente para esta funcionalidade.

scrapping the web

A terceira categoria abrange as aplicações que fazem *scrapping*, ou seja, extraem informação provenientes de páginas *web*. Por exemplo, *Dapper* [15], *Openkapow* [57], *WebSundew* [83] e *Strata* [70].

Com excepção do *Dapper*, todas elas são aplicações *standalone*. Cada uma tem propósitos diferentes e têm apenas em comum a presença de módulos que fazem *scrapping* de páginas *web*. Para efectuar a extracção todas usam combinações entre *XPath*[88], *XQuery*[89], XSLT[90] e expressões regulares.

mashup editors

A última categoria engloba os *mashup editors*. Consistem em aplicações que combinam conteúdos de várias fontes com vista a criar um produto final. Entre as várias existentes realçar *Yahoo Pipes* [93], *Microsoft PopFly* [52], *Google Mashup Editor* [29] e *Orchestr8 AlchemyPoint* [58].

Nenhuma das aplicações apresenta técnicas para a extracção de informação a partir de páginas *web*, suportando apenas, por exemplo, RSS [64], *Atom* [5], JSON [46], XML [86] e CSV [14].

zoetrope

Fora destas categorias é importante realçar o *Zoetrope* [1, 96]. Esta é a aplicação que mais semelhanças aparenta ter com o *9ticks*. *Zoetrope* permite ao utilizador visualizar informação passada, correlacionar páginas e definir visualizações personalizadas. As desvantagens aparentes são tratar-se de uma aplicação *standalone* e estar limitada a um conjunto de páginas previamente definido.

Não existe muita informação para além de um artigo e de um vídeo, pelo que pouco mais se pode a aferir sobre ele.

tecnologias

É também importante falar das diversas tecnologias relevantes para a construção de uma plataforma com as características do *9ticks*. Em particular os mecanismos de comunicação, os sistemas de armazenamento, tecnologias de *rendering* e extracção de informação e tecnologias para o desenvolvimento *web*. Estas tecnologias não vão ser sumariadas neste capítulo, podendo sempre ser consultadas na Secção D.2.

"Web users ultimately want to get at data quickly and easily. They don't care as much about attractive sites and pretty design."

Tim Berners-Lee

3 Requisitos

Neste capítulo apresenta-se uma versão resumida dos requisitos do *9ticks*. Notar que a versão completa do levantamento de requisitos pode ser consultada no Anexo E.

requisitos funcionais

Os requisitos funcionais do sistema podem ser catalogados em diversas categorias, abordando-se cada uma em seguida.

interface com o utilizador

É essencial que a aplicação consista numa página *web*, em que as fontes de informação seleccionadas pelo utilizador sejam apresentadas por meio de um *dashboard*.

É desejável que o sistema possa também ser acedido através de um dispositivo móvel, podendo através dele consultar e subscrever novas fontes. Deverse-á ter cuidado especial na comunicação com o dispositivo móvel devido às limitações da ligação.

Ambos os sistemas têm que suportar diversos utilizadores, recorrendo a um mecanismo de autenticação e autorização.

fontes predefinidas

O utilizador deve ter a possibilidade de subscrever fontes de eventos predefinidas. Devem existir fontes predefinidas para a bolsa de valores e para a meteorologia. É desejável que haja também para resultados desportivos, trânsito, tráfego aéreo, estado de leilões e *feeds* RSS/*Atom*.

fontes personalizadas

Deve existir a possibilidade de criar e subscrever fontes de eventos personalizadas. É essencial que o utilizador possa seleccionar, dentro de determinada página *web*, elementos soltos de informação. É desejável que também possam ser seleccionados elementos em tabelas e valores repetitivos (por exemplo, todos os títulos de notícias).

É também desejável que o sistema seja robusto ao ponto de permitir extrair informação de páginas que necessitem de um conjunto metódico de passos para serem alcançadas (por exemplo, páginas com autenticação).

Caso a estrutura de uma página seja alterada, é desejável que o sistema se adapte a essa mudança de forma a poder continuar a extrair informação dessa página.

tipagem de dados

Ao definir uma fonte de eventos personalizada, os tipos de dados de cada elemento deverão ser automaticamente sugeridos pelo sistema (por exemplo, números inteiros, decimais, cadeias de caracteres e datas). De qualquer forma, o utilizador poderá sempre alterar os tipos de dados reconhecidos.

fontes de eventos

As fontes de eventos deverão ser catalogadas segundo um espaço de nomes hierárquico. O utilizador poderá pesquisar e adicionar fontes de eventos previamente definidas por outros utilizadores. A informação histórica de cada fonte de eventos deverá ser armazenada com vista a suportar futuras pesquisas e consultas.

visualização de eventos

O utilizador deverá ter a possibilidade de escolher como visualizar cada fonte de informação. É essencial que possa visualizar as fontes de forma textual e por meio de gráficos (por exemplo, linhas, barras e circulares). É desejável que as fontes possam ser visualizados utilizando mapas, *gauges* e *tag clouds*. É também desejável que o utilizador possa aplicar funções de agregação aos eventos, podendo também filtrá-los ou ordená-los.

definição de alertas

É desejável que o utilizador possa definir alertas, mediante condições, sobre os eventos. Estes alertas poderão ter como destino o *dashboard*, o *e-mail* ou um telemóvel através do serviço de SMS.

requisitos não-funcionais

Quanto aos requisitos não-funcionais, é essencial que os eventos e os alertas sejam visualizados em tempo real. A informação antiga deverá ser agregada suportando vários níveis e funções de agregação.

atributos de qualidade

O sistema deve cumprir alguns atributos de qualidade. Em particular é essencial que seja escalável, tenha disponibilidade elevada, seja confiável e seguro.

"And now for something completely different."
Monty Python's Flying Circus

4
Visão do Utilizador

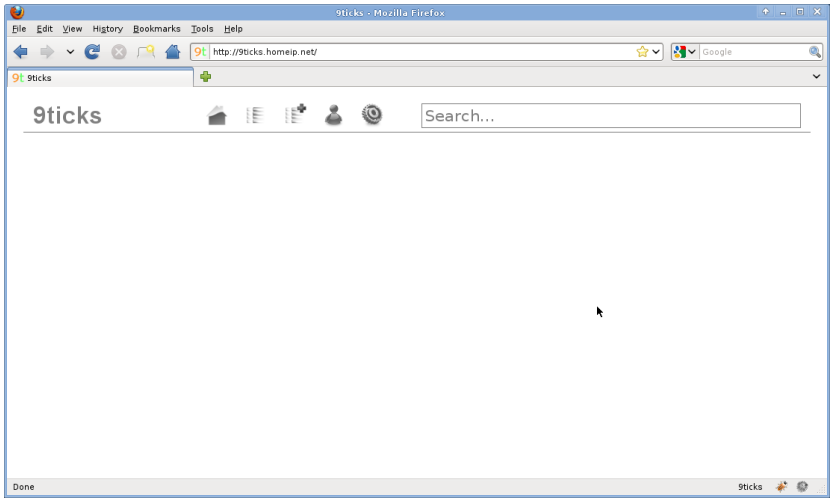


Figura 4.1: *9ticks*: Página inicial

Ao iniciar, o *9ticks* apresenta um *dashboard* vazio, pronto para lá serem colocadas fontes de informação (Figura 4.1).

Para adicionar uma fonte de informação ao *dashboard*, o utilizador escreve as palavras-chaves do que procura na caixa de texto no topo da janela. O *9ticks* irá então apresentar uma lista de fontes de eventos, entre as quais deve escolher uma.

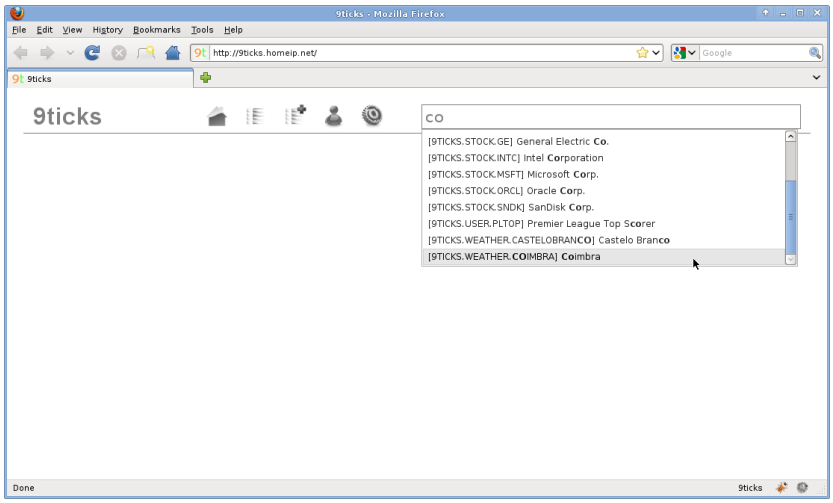


Figura 4.2: *9ticks*: Pesquisar fontes

Imagine-se que se quer adicionar o estado meteorológico em Coimbra ao *dashboard*. Para isso, (Figura 4.2) o utilizador começa por escrever a palavra *coimbra* e, quando ainda não terminou, o *9ticks* apresenta uma lista de sugestões, onde efectivamente se encontra a opção desejada. Após seleccionada, a nova fonte de eventos aparecerá no seu *dashboard* pessoal (Figura 4.4).

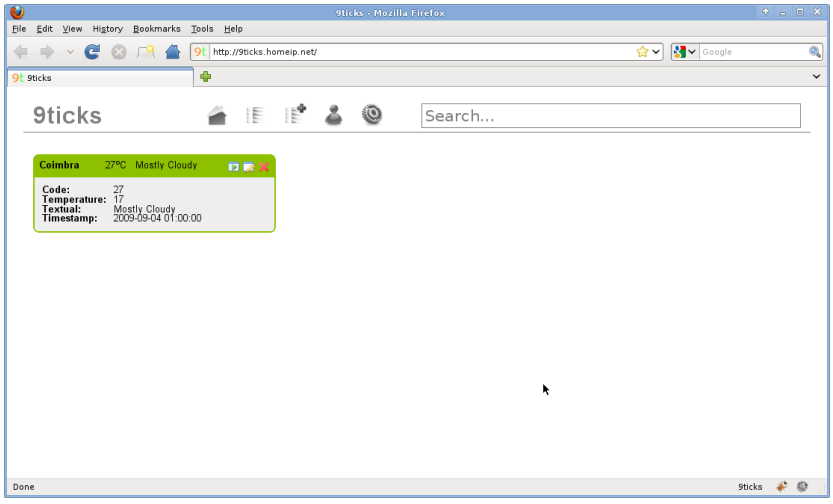


Figura 4.3: *9ticks*: Inserir fonte

Cada fonte adicionada ao *dashboard* apresenta três botões. O primeiro contrai a fonte de forma a ocupar menos espaço (e expande para voltar à forma normal), o segundo abre o modo de edição (onde se pode alterar, por exemplo, a forma como se visualiza a fonte) e por fim o botão para remover a fonte do *dashboard*.

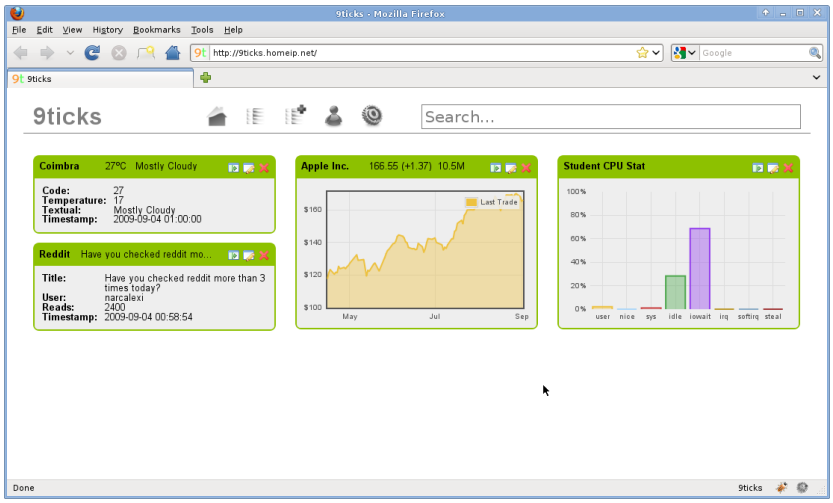


Figura 4.4: *9ticks*: Inserir mais fontes

Para além do estado meteorológico em Coimbra, imagine-se que se quer adicionar também o estado da bolsa de valores para a *Apple Inc.*, o estado do processador do servidor dos alunos do departamento e a notícia mais lida do *Reddit*. Repetindo o processo de procura anterior, obtemos um *dashboard* como o que apresenta a Figura 4.4.

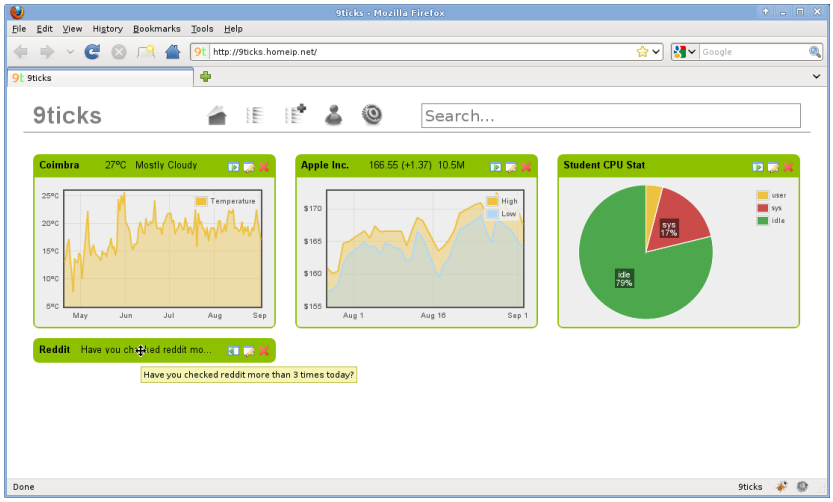


Figura 4.5: *9ticks*: Visualizações diferentes

Na Figura 4.5 observa-se que a mesma fonte de informação pode ser visualizada de formas diferentes. Por exemplo, a temperatura em Coimbra era representada de forma textual passando a ser representada de forma gráfica ao longo do tempo, o estado do processador estava representado por um gráfico de barras e passou a estar por um gráfico circular. Notar também que a fonte do *Reddit* apresenta uma visualização linear.

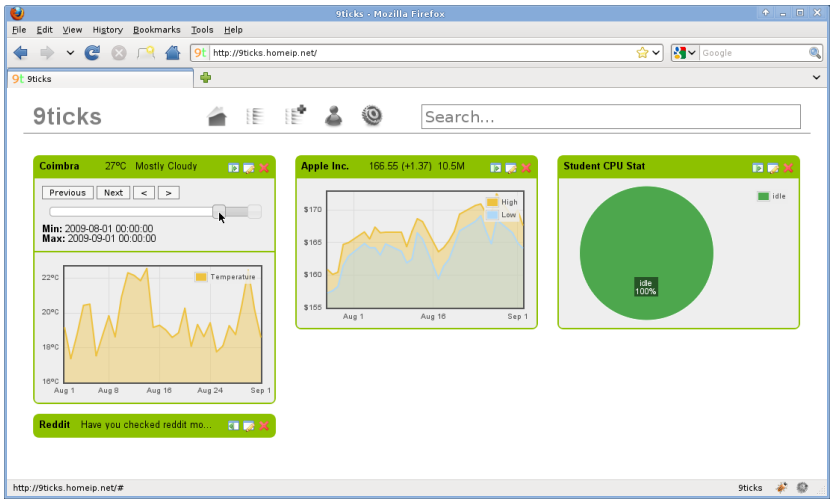


Figura 4.6: *9ticks*: Ajuste da janela de visualização temporal

Como vimos anteriormente, a temperatura em Coimbra encontra-se agora visualizada de forma histórica. Assim, é possível definir a janela de visualização temporal (Figura 4.6). Para isso carrega-se no botão de edição e aparece uma barra de deslocamento com dois botões. Ao arrastar cada um deles, a janela temporal altera-se e o gráfico será ajustado. No exemplo ajustou-se os limites da janela de forma a se visualizar a temperatura em Coimbra durante o mês de Agosto.

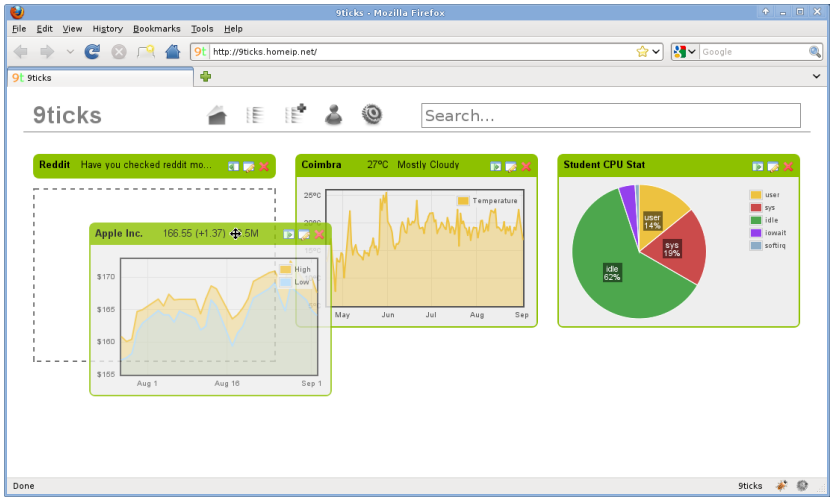


Figura 4.7: *9ticks*: Reorganizar *dashboard*

Na Figura 4.7 demonstra-se que é possível trocar de ordem as fontes de informação adicionadas ao *dashboard*. Para isso basta arrastar cada fonte pegando no seu cabeçalho. À medida que a fonte é movida pelo *dashboard*, o *9ticks* irá indicando a tracejado quais os locais para onde a fonte pode ser movida.

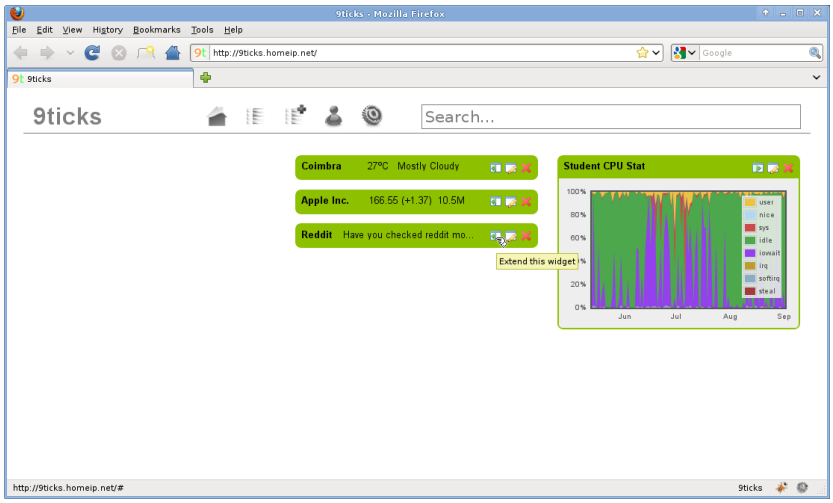


Figura 4.8: *9ticks*: Visualização linear

Na Figura 4.8 observa-se que é possível colocar várias fontes de informação par a par de forma a ser mais simples ler a sua informação. Isto permite que o utilizador, com apenas um breve olhar, consiga inteirar-se do que se passa com as suas fontes.

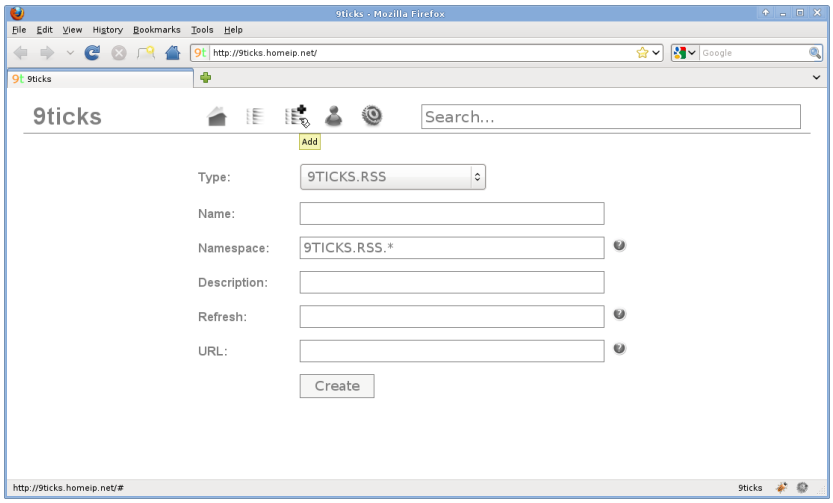


Figura 4.9: *9ticks*: Adicionar fonte a uma categoria

Até ao momento, o utilizador apenas está a adicionar ao seu *dashboard* fontes que já se encontram definidas no sistema. O *9ticks* tem uma lista de categorias às quais o utilizador pode facilmente adicionar mais fontes. Por exemplo, se o utilizador quiser adicionar um novo *feed* que ainda não exista no sistema, ele pode facultar ao *9ticks* o seu URL e criará então uma nova fonte. Para isso, selecciona-se o botão para adicionar fonte no topo da janela (Figura 4.9).

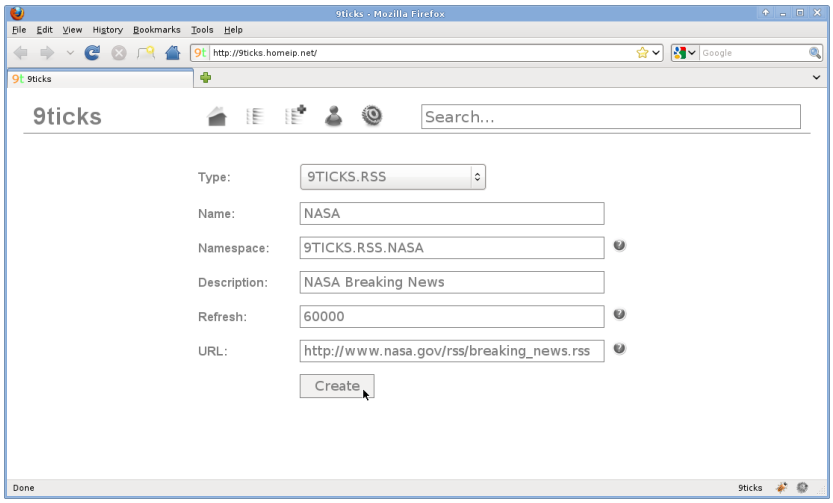


Figura 4.10: *9ticks*: Adicionar a NASA à categoria dos *feeds* RSS

Imagine-se que o utilizador deseja adicionar o *feed* RSS da NASA ao seu *dashboard*. Para isso, basta preencher o formulário na Figura 4.10 indicando o seu URL. Tem também que especificar qual o nome que quer dar à sua fonte e adicionar uma descrição relevante. O *9ticks* sugere automaticamente os valores para os restantes campos. No lado direito de cada campo do formulário aparece como ajuda uma breve descrição.

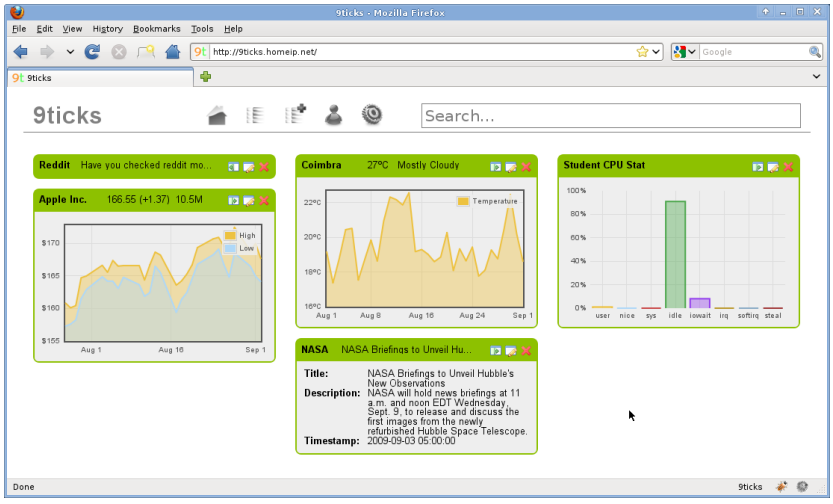


Figura 4.11: *9ticks*: Adicionar a NASA ao *dashboard*

Após adicionar a NASA à categoria dos *feeds* do *9ticks*, a nova fonte é fica disponível para procura, como qualquer outra previamente inserida. Após a sua selecção esta irá aparecer no *dashboard* como é exemplificado na Figura 4.11.

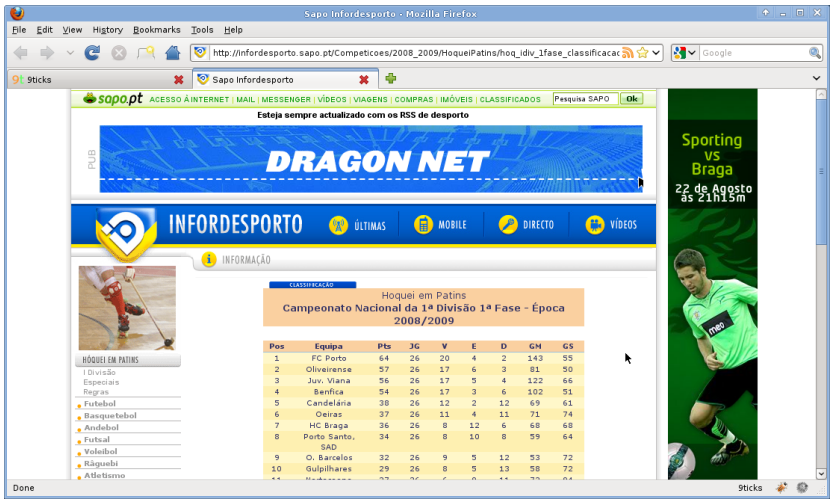


Figura 4.12: *9ticks*: Resultados do hóquei em tempo real

Imagine-se agora que o utilizador que adicionar os resultados da primeira liga de hóquei português. Ele poderia adicionar um *feed*, tal como vez anteriormente mas, acontece que não existe nenhum *feed* desta fonte. Mas, ele tem conhecimento que a informação é disponibilizada em tempo real pela página *web* `http://infordesporto.sapo.pt` (Figura 4.12).

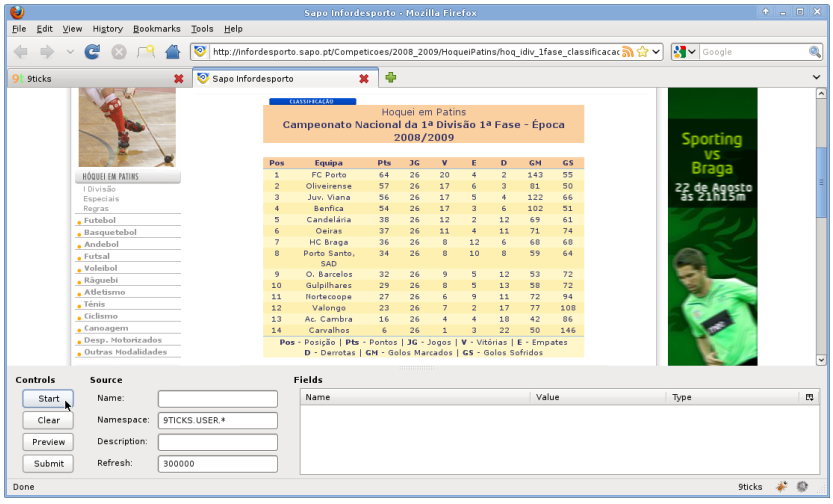


Figura 4.13: *9ticks*: Abrir *plugin* do *9ticks* no *browser*

Estando o utilizador na página onde se encontra a informação, ele pode utilizar um *plugin* para indicar ao *9ticks* qual a informação desejada. Para iniciar a captura dele elementos na página selecciona-se o botão *Start* (Figura 4.13).

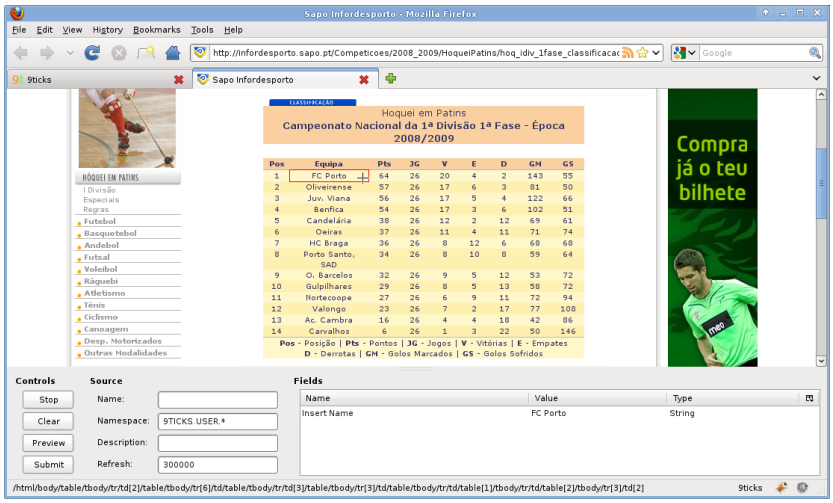


Figura 4.14: *9ticks*: Selecção dos elementos na página

O utilizador, com o auxílio do rato, selecciona quais os elementos na página que contêm informação relevante para a nova fonte de informação. A página vai interagir com o movimento do rato, sobressaindo cada elemento sempre que possa ser seleccionado. Imagine-se que o utilizador deseja saber qual a equipa que se encontra em primeiro lugar e respectivo número de pontos, jogos e vitórias. O utilizador selecciona esses quatro elementos com o rato (Figura 4.14), sendo assim adicionados à tabela na base da janela.

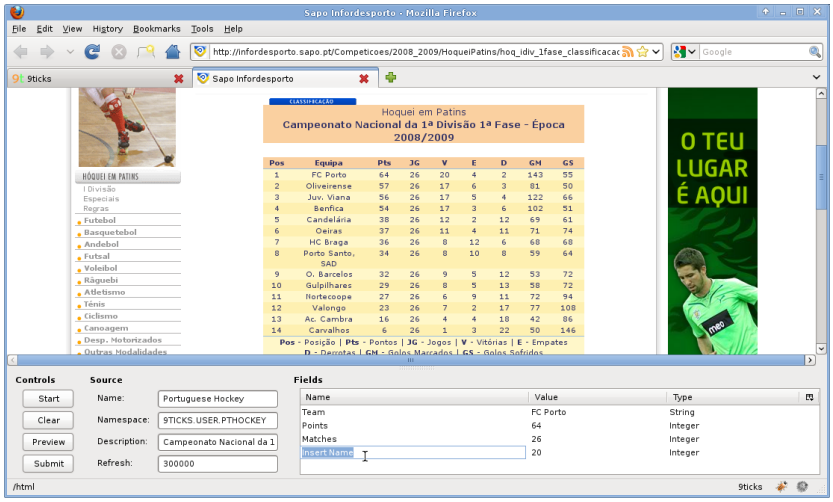


Figura 4.15: *9ticks*: Preenchimento do formulário

Após terminar a selecção dos elementos tem que indicar nomes sugestivos para cada um deles (Figura 4.15). Tem ainda que preencher o formulário relativo à fonte, onde define, por exemplo, o seu nome e descrição, tal como aconteceu anteriormente.

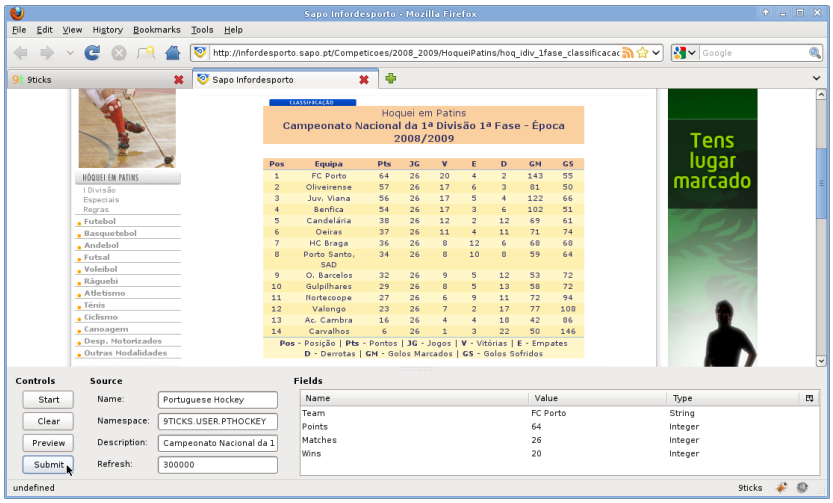


Figura 4.16: *9ticks*: Criação de fonte personalizada

Após o correcto preenchimento do formulário, apenas é necessário seleccionar o botão *Submit* e a fonte acabada de ser descrita tornar-se-á como qualquer outra fonte no *9ticks* (Figura 4.16).

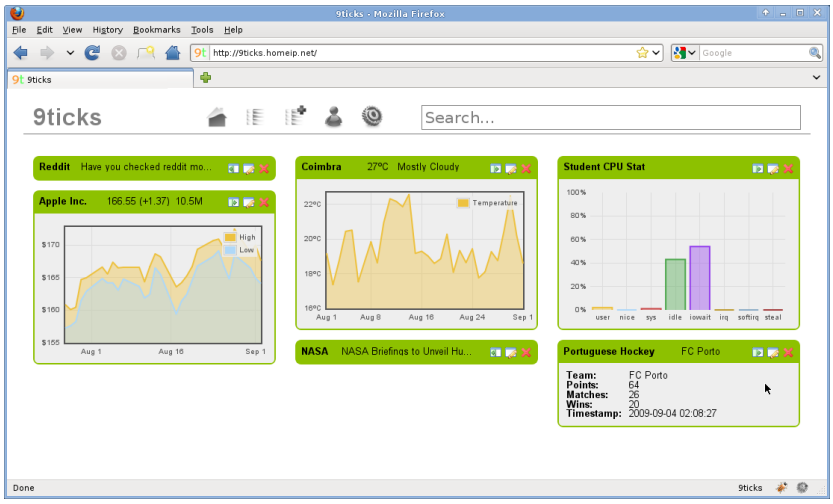


Figura 4.17: *9ticks*: Adição da fonte personalizada ao sistema

Finalmente apenas é necessário procurar pela fonte recentemente criada e seleccioná-la de forma a ser adicionada ao *dashboard* pessoal.

É também possível fazer *browse* pelas diversas categorias e respectivas fontes existentes no *9ticks*. De forma a que o *dashboard* organizado seja guardado e carregado ao voltar, é necessário efectuar registo, carregando para isso no botão apropriado no topo da página.

"It's not trivial. The hard part is building something that people can use and having it scale to millions of people."

David A. Patterson

5 Arquitectura

Nesta secção descreve-se de forma genérica a arquitectura do sistema (Figura 5.1), explicitando a importância de cada um dos módulos. No Capítulo 6, Implementação, encontram-se as partes mais relevantes do sistema descritas pormenorizadamente.

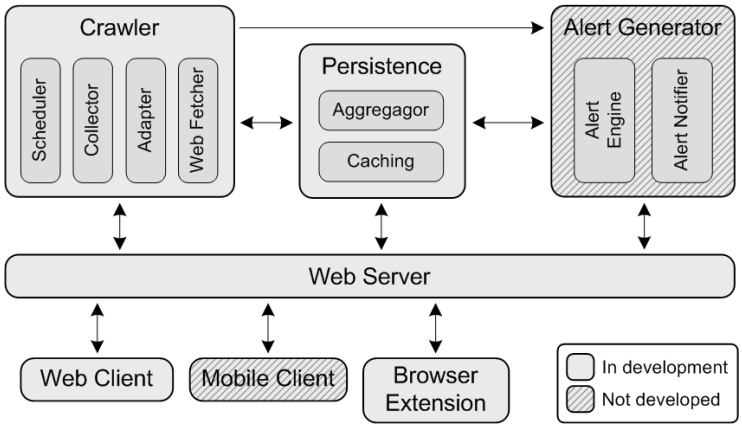


Figura 5.1: Arquitectura do *9ticks*

crawler O *Crawler* é o componente responsável por fazer *polling*, de forma periódica, a cada fonte de eventos, de forma a verificar se existiram actualizações (Figura 5.2). Ele consiste em quatro subcomponentes chave: *Scheduler*, *Collector*, *Adapter* e *Web Fetcher*.

adapter Cada fonte de eventos tem um *adapter* associado a ela. Este *adapter* é responsável por implementar a lógica que permita verificar se ocorreu algum
scheduler evento e ir buscá-lo. O *scheduler* é responsável por distribuir as fontes de in-

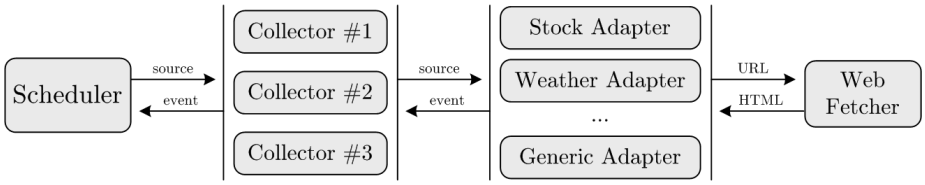


Figura 5.2: *Crawler*

collector formação subscritas pelos diversos *collectors*. O *collector* implementa o mecanismo de *polling*, obedecendo à distribuição de tarefas estabelecida pelo *scheduler*. Pode existir mais que um *collector* a correr num dado momento de forma a balancear a carga. O *web fetcher* é a ligação dos *adapters* com a Internet, sendo responsável por ir buscar as páginas *web* de forma a serem processadas.

web fetcher

persistence *Persistence* é o módulo responsável pelo armazenamento da informação do sistema, maioritariamente constituída pela informação proveniente do módulo *Crawler*.

Devido às grandes quantidades de informação recolhidas, ao longo do tempo, pelo *Crawler* surge a necessidade de aumentar a performance de leitura dos dados. Nessa linha, o componente *Aggregator* tem como função sumariar os dados. Outra forma de diminuir a latência na leitura de informação é usar um mecanismo de *caching*. O componente *Caching* é responsável por essa tarefa.

aggregator

caching

web client O *Web Client* representa a página *web*, a visão principal que o utilizador tem do *9ticks*. De forma similar, o *Mobile Client* representa a aplicação móvel.

mobile client

browser extension *Browser Extension* consiste num *plug-in* de um navegador *web* que permite seleccionar pedaços relevantes em páginas *web*, e identificar o seu tipo de dados, de forma a criar fontes de eventos personalizadas.

web server O *Web Server* estabelece a ponte entre o sistema de processamento e os *frontends* para os utilizadores.

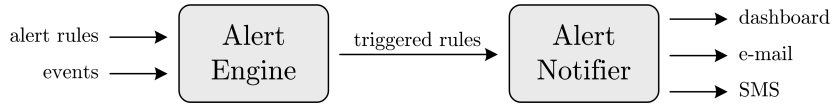


Figura 5.3: *Alert Generator*

alert generator O *Alert Generator* é composto por dois módulos distintos: *Alert Engine* e *Alert Notifier* (Figura 5.3). O *alert engine* é um serviço autónomo, que recebe as regras de alerta (*alert rules*) e os eventos. Sempre que os eventos recebidos disparem regras de alerta um alerta é gerado. O *alert notifier* é responsável por receber todos os alertas gerados pelo *alert engine* e direccioná-los para o ou os destinos correctos. Estes destinos podem ser o *dashboard* do utilizador, E-mail ou, futuramente, SMS. É desejável que cada um dos módulos possa ser

alert engine

alert notifier

composto por várias instâncias consoante o *workload* do sistema.

Notar que os módulos *Mobile Client* e *Alert Generator* não se encontram actualmente implementados.

"How Little Things Can Make a Big Difference"
Malcolm Gladwell – The Tipping Point

6 Implementação

Nesta secção descreve-se a implementação do sistema, evidenciando as suas partes mais importantes. Começa-se por enumerar as diversas linguagens e tecnologias utilizadas, tipos de dados principais e como é que os diversos módulos introduzidos na arquitectura comunicam entre sim. Por fim procede-se a uma descrição módulo por módulo com algum pormenor.

6.1 Notas gerais

6.1.1 Tecnologias

No desenvolvimento do sistema foram usadas diversas linguagens e tecnologias. Estas encontram-se listadas nas Tabelas 6.1 e 6.2.

Módulo	**Linguagens**
Persistence	*Java*
Crawler	*Java*, *Python*, *XPath* [88], HTML e *JavaScript*
Web Server	*Java* e JSP
Web Client	HTML, CSS e *JavaScript*
Firefox Extension	XUL [91], *JavaScript*, *XPath* e CSS
Mobile Client	não implementado
Alert Generator	não implementado

Tabela 6.1: Linguagens de programação usadas

Após a leitura da Tabela 6.1 conclui-se que a linguagem *Java* é usada em todo o *core* do sistema enquanto que nos clientes utiliza-se principalmente HTML e *JavaScript*.

Módulo	Tecnologia	Linguagem
Comunicação	*Apache Thrift* [74]	*cross-language*
	JSON [46]	*cross-language*
Persistence	*Hypertable* [37]	C++
	Memcached [51]	C
Crawler	HTML*Cleaner* [34]	*Java*
	Twitter4J [78]	*Java*
	Eddie RSS/Atom [18]	*Java*
	JSSh [48]	*C++*
Web Server	*Jetty* [42]	*Java*
	Cometd-Jetty [9]	*Java*
	log4j [49]	*Java*
Web Client	*jQuery* [45]	*JavaScript*
	Flot [25]	*JavaScript*
	EasyWidgets [45]	*JavaScript*
	Cometd [9]	*JavaScript*

Tabela 6.2: Tecnologias usadas

De entre as várias tecnologias utilizadas como base do *9ticks* as escolhas mais importantes recaem sob *Hypertable* (armazenamento), *Memcached* (*caching*), *Cometd* (*server push*) e *Thrift* (comunicação).

6.1.2 Tipos de Dados

Foram criados várias tipos de dados como suporte ao 9ticks. De todos eles é importante salientar os mais importantes, fonte (Tabela 6.3) e evento (Tabela 6.5), e dar exemplos da sua utilização. A coluna *Obrigatório* indica se o atributo tem ou não que estar presente no sistema de armazenamento.

6.1.2.1 Source

Atributo	Tipo	Obrigatório
`namespace`	`string`	sim
`name`	`string`	sim
`refresh`	`integer`	sim
`sourceType`	`SourceType`	sim
`fields`	`list<SourceField>`	sim
`properties`	`map<string, string>`	não
`description`	`string`	não

Tabela 6.3: *Source*

namespace Cada fonte é identificada univocamente pelo atributo `namespace`. Este obedece uma hierarquia, permitindo uma catalogação automática de todo o

espaço de fontes e uma fácil identificação por parte do utilizador. Por exemplo, `9TICKS.STOCK.AAPL` (Acções da *Apple Inc.*), `9TICKS.WEATHER.LISBOA` (Tempo em Lisboa), `9TICKS.WEATHER.COIMBRA` (Tempo em Coimbra), `9TICKS.TRAFFIC.LONDON` (Trânsito em Londres) e `9TICKS.USER.JOHNDOE.MYSOURCE` (Definida pelo utilizador).

refresh

O atributo `refresh` simboliza o intervalo de tempo desejado entre actualizações. Ou seja, o intervalo em que sistema deve verificar se existiu algum evento por parte da fonte em causa.

sourceType

O tipo de dados `SourceType` determina a categoria a que a fonte pertence e determina o segundo nível do *namespace*. Por exemplo, actualmente são suportadas as categorias `USER`, `STOCK`, `WEATHER`, `GOOGLE`, `EBAY`, `RSS`, e `TWITTER`.

Atributo	Tipo	Obrigatório
`key`	`string`	sim
`name`	`string`	sim
`dataType`	`DataType`	sim
`aggregationType`	`AggregationType`	sim

Tabela 6.4: *SourceField*

fields

À partida, cada fonte sabe o nome dos atributos dos seus futuros eventos. Estes atributos são descritos pelo campo `fields` através do tipo de dados `SourceField` (Tabela 6.4).

properties

Devido à heterogeneidade das fontes, cada uma delas tem propriedades intrínsecas que nem sempre são fáceis de representar. Optou-se por uma representação genérica através de um array associativo, correspondente ao campo `properties`.

Na Tabela 6.4 são mencionados dois novos tipos de dados. `DataType` indica para cada campo qual o seu tipo de dados simples (ex: *integer*, *float* e *string*). O tipo `AggregationType` representa como os valores desse campo vão ser sumariados e vai ser aprofundado na Secção 6.2.3.

6.1.2.2 Event

Atributo	Tipo	Obrigatório
`namespace`	`string`	sim
`timestamp`	`long`	sim
`properties`	`map<string, string>`	sim

Tabela 6.5: *Event*

namespace
timestamp

Cada evento é identificado univocamente pela composição dos atributos `namespace` e `timestamp`. Ou seja, cada fonte produz vários eventos em instantes

properties temporais diferentes. Uma vez mais, devido à heterogeneidade dos eventos, os seus valores são armazenados num array associativo correspondente ao campo `properties`.

6.1.2.3 Exemplo

Imagine-se o caso onde queremos representar a fonte correspondente ao mercado bolsista da *Apple Inc.*, considerando apenas os valores *última transacção* e *volume de transacções*.

```
Stock(
  namespace:   '9TICKS.STOCK.AAPL',
  name:        'Apple Inc.',
  description: 'Apple Inc. Stock Quote (NASDAQ)',
  refresh:     5000,
  sourceType:  SourceType(STOCK),
  fields:      [
                 SourceField(
                   name:            'Last Trade',
                   key:             'trade'
                   dataType:        DataType(DOUBLE),
                   aggregationType: AggregationType(AVG)
                 ),
                 SourceField(
                   name:            'Volume',
                   key:             'volume'
                   dataType:        DataType(LONG),
                   aggregationType: AggregationType(MAX)
                 )
               ],
  properties:  {
                 'symbol' = 'AAPL'
               }
}
```

Listagem 6.1: Fonte de eventos relativa ao índice da bolsa `AAPL`

```
Event(
  namespace:  '9TICKS.STOCK.AAPL',
  timestamp:  1251129600000,
  properties: {
                trade =  169.06,
                volume = 14533168
              }
)
```

Listagem 6.2: Evento relativo ao índice da bolsa `AAPL`

Na Listagem 6.1 encontra-se a fonte `9TICKS.STOCK.AAPL` representada numa notação JSON-*like*. Para que dentro da categoria *stock* o sistema saiba qual o índice da bolsa se trata, existe no array associativo `properties` a propriedade `symbol`. Notar também que cada atributo dos futuros eventos sabe à partida qual o tipo de valores que admite e como é que esses valores vão ser sumariados com o passar do tempo. Na Listagem 6.2 está descrito um evento da fonte `9TICKS.STOCK.AAPL`.

6.1.3 Orientação a Serviços

O *core* do *9ticks* é composto por diversas entidades autónomas. Estas entidades executam de forma independente, implementam pequenas funcionalidades e disponibilizam interfaces para integração.

Apesar de existirem algumas semelhanças com uma *Service Oriented Architecture* [21] tradicional, alguns dos seus aspectos foram relaxados pela complexidade que introduziriam no sistema. Nomeadamente, as entidades não se encontram ligadas por um *Enterprise Service Bus*, não se encontram registadas num directório de serviços (mas existe intenção de o fazer), não existe orquestração de forma a eliminar um *hop* desnecessário e a comunicação ente os serviços não segue um *standard* aberto.

Ver o *9ticks* como um conjunto de serviços permite a evolução do sistema, quer através da adição de novos serviços, quer através da alteração de serviços já existentes. Ao se tratar de uma arquitectura modular, o mesmo serviço pode ser facilmente utilizado por várias entidades em simultâneo. Por outro lado, a comunicação ente os diversos serviços é o preço a pagar por um sistema modular.

6.1.4 Comunicação entre Serviços

Thrift

Para a comunicação entre cada serviço escolheu-se a *framework Apache Thrift* [74]. Este mecanismo de comunicação abrange apenas o *core* do 9ticks, ou seja, a comunicação entre os componentes *Scheduler*, *Collector*, *Persistence*, e *Web Server*.

Thrift é uma *framework* desenvolvida pelo *Facebook* que permite aplicações escritas num vasto leque de linguagens comunicarem entre si. Actualmente faz parte da incubadora da *Apache Software Foundation*. Permite a definição de tipos de dados e serviços (métodos) através de uma linguagem intermédia (IDL), gerando o código correspondente nas diversas linguagens. A sua baixa latência, flexibilidade e escalabilidade encontra-se bem provada ao ser vastamente usada na implementação do *Facebook*.

```
struct Event {
    1: string namespace,
    2: i64 timestamp,
    3: map<string,string> properties
}

service Datastore {

    // inserts an event
    void insertEvent(1:Event event)
              throws (1:DatastoreException e),

    // selects the last event, if any
    Event selectEvent(1:string namespace)
               throws (1:DatastoreException e),

    // ...
}
```

Listagem 6.3: Exemplo da IDL do *Thrift*

Recorrendo à IDL do Thrift, foram descritos os tipos de dados mencionados na Secção 6.1.2, nos quais todo o sistema assenta. Foram também descritas as diversas interfaces que cada serviço implementa e expõe. Por exemplo, na Listagem 6.3, encontra-se descrito em linguagem intermédia o tipo de dados *event* e parte da interface de acesso à base de dados responsável por persistir e obter eventos. Esta linguagem irá gerar, em particular para *Java*, a classe *Event* e a *interface Persistence*, a implementar.

Na Secção 6.7 – Tentativas Falhadas encontram-se outras tecnologias usadas anteriormente que por algum motivo a sua utilização deixou de ser viável.

6.2 Persistence

Construir um sistema que suporte eficientemente uma alta e contínua *insertion rate* (vários eventos a chegar a todo o momento), *range queries* (consultar visualmente informação histórica) e *window aggregations* (sumariar informação passada dada uma janela temporal) é dos maiores desafios do *9ticks*. Na realidade, o mercado actual encontra-se segmentado em diversos produtos, cada um deles especializado em determinado tipo de operações (por exemplo, bases de dados, *data warehouses*, sistemas de processamento de eventos e sistemas de armazenamento distribuídos). Além disto, o tipo de informação em causa dá extremo ênfase à questão temporal e a maioria dos sistemas de armazenamento não se encontra preparado para lidar eficientemente com questões temporais.

Actualmente, o *9ticks* usa o *Hypertable* [37] como sistema de armazenamento. Antes de se optar por ele foram postos em causa diversos sistemas, os quais se encontram descritos na Secção 6.7 – Outras Abordagens.

6.2.1 Hypertable

Contrariamente às bases de dados relacionais, o *Hypertable* armazena a in-
column store formação segundo um formato vertical (*column store*), sendo ideal para dados esparsos. Trata-se também de um sistema de armazenamento temporal providenciando uma interface optimizada para a leitura de informação com base no seu *timestamp* (por exemplo, seleccionar toda a informação entre dois *timestamps*).

Conceptualmente, o *Hypertable* guarda a informação numa tabela, a qual pode ser pensada como um conjunto de linhas indexadas por uma chave composta. A tabela é guardada de forma esparsa, permitindo que as linhas possam ter um número variável de colunas. Sendo esparsa, para cada linha não é obrigatório que todas as colunas tenham valor. A chave composta corresponde a um tuplo com quatro valores: chave da linha, família da coluna, chave da coluna e *timestamp*.

A Tabela 6.6 exemplifica conceptualmente como o *Hypertable* guarda os eventos provenientes de duas fontes distintas. De forma a tornar a tabela mais concisa, foram realizadas algumas simplificações que em nada alteram a sua compreensão.

row key	timestamp	trade	volume	temp.	press.
STOCK.AAPL	t1	169.06	14533168		
STOCK.AAPL	t2	169.27			
STOCK.AAPL	t3	169.15	15733107		
WEATHER.LISBOA	t6			14.0	1016.9
WEATHER.LISBOA	t7			15.0	1017.0

Tabela 6.6: *Hypertable*: Vista conceptual

Apesar de a nível conceptual as tabelas possam ser vistas como sendo esparsas, fisicamente elas são armazenadas de forma a não existirem células em branco. A representação da Tabela 6.6 ao nível físico pode ser visualizado na Tabela 6.7.

row key	column	timestamp	value
STOCK.AAPL	stock:trade	t1	169.06
STOCK.AAPL	stock:trade	t2	169.27
STOCK.AAPL	stock:trade	t3	169.15
STOCK.AAPL	stock:volume	t1	14533168
STOCK.AAPL	stock:volume	t3	15733107
WEATHER.LISBOA	weather:temperature	t6	14.0
WEATHER.LISBOA	weather:temperature	t7	15.0
WEATHER.LISBOA	weather:pressure	t6	1016.9
WEATHER.LISBOA	weather:pressure	t7	1017.0

Tabela 6.7: *Hypertable*: Vista física

O *namespace* da fonte é sempre que possível usado como chave da linha. O campo `column` é composto por dois valores distintos: a família da coluna (ex: *stock* e *weather*) e a chave da coluna (ex: *trade* e *temperature*). Isto permite ao *Hypertable* armazenar contiguamente as linhas com a mesma família de coluna. Apontar ainda que podemos verificar na Tabela 6.7 que, para uma mesma fonte e atributo, existem vários valores diferentes com o passar do tempo.

6.2.2 Estrutura das tabelas

Tratando-se de tabelas esparsas, poderíamos usar apenas uma grande tabela para armazenar toda a informação relativa ao sistema. No entanto, decidiu-se criar uma tabela para cada tipo de informação distinta: fontes, eventos, visualizações e utilizadores (Tabela 6.8).

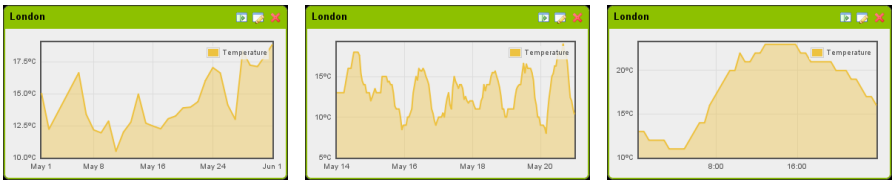
Na Tabela 6.7 podemos encontrar um exemplo prático da tabela *event*. As diferentes visualizações que a informação pode ter serão descritas na Secção 6.4. Existem tabelas auxiliares para sumariar a informação com o passar do tempo, sendo estas abordadas na Secção 6.2.3.

Tabela:	*source*
Row Key:	*namespace*
Column Family:	*source*
Column Key:	*name, refresh, type, fields, properties* e *description*
Timestamp:	–
Tabela:	*event*
Row Key:	*namespace*
Column Family:	*user, stock, weather, google, ebay, rss* e *twitter*
Column Key:	*trade (stock), volume (stock), temperature (weather), results (google), title (rss), description (rss), ...*
Timestamp:	*timestamp* do evento
Tabela:	*visualization*
Row Key:	*namespace*
Column Family:	*visualization*
Column Key:	–
Timestamp:	–
Tabela:	*user*
Row Key:	*username*
Column Family:	*user*
Column Key:	*name, password* e *dashboard*
Timestamp:	–

Tabela 6.8: *Hypertable*: Tabelas

6.2.3 Agregações

Quando se lida com um grande volume de informação histórica, é regra comum sumariá-la com vista a acelerar a sua visualização. Por exemplo, caso um utilizador deseje visualizar a informação histórica relativa a um mês, será apresentada informação sumariada dia-a-dia.



(a) um mês (b) uma semana (c) um dia

Figura 6.1: Exemplos de agregações

Na Figura 6.1 encontra-se a temperatura em Londres vista sob três intervalos de tempo diferentes. Podemos observar que o nível de detalhe aumenta da esquerda para a direita. Ou seja, quanto maior for a janela de visualização, maior é a agregação.

Caso a informação não seja sumariada, é necessário ler do sistema de armazenamento grandes quantidades de informação sempre que fosse necessário visualizar informação histórica abrangente. Com vários pedidos em simultâneo, esta abordagem acabaria por comprometer o desempenho do sistema.

De forma a proceder a essas agregações, foram criadas tabelas auxiliares com a mesma estrutura da tabela *event*. Cada tabela corresponde a um nível diferente de agregação, estando actualmente definidos os níveis descritos na Tabela 6.9.

Tabela	**Nível de Agregação**
event	em bruto
event_minute	minuto a minuto
event_hour	hora a hora
event_day	dia a dia
event_month	mês a mês
event_year	ano a ano

Tabela 6.9: Níveis de agregação

Nem todos os campos podem ser sumariados usando a mesma função de agregação. Por exemplo, para uma acção da bolsa, o atributo *trade* é sujeito a uma média enquanto que o atributo *volume* uma soma. Assim sendo, cada atributo tem que ter informação redundante, especificando qual o tipo de dados e qual o método de agregação (Listagem 6.1).

aggregator

Actualmente, o *Hypertable* não suporta funções de agregação, pelo que todos os cálculos têm que ser realizados pelo componente *aggregator*.

O *aggregator* implementa as funções de agregação AVG (média aritmética), MIN (valor menor), MAX (valor máximo), FIRST (valor com menor *timestamp*), LAST (valor com maior *timestamp*) e MODE (valor mais frequente). O algoritmo tem diferente comportamento mediante o tipo de dados do campo a que a função de agregação está associada. Por exemplo, a média aritmética é trivial quando aplicada a valores numéricos mas não tem significado para *strings*.

6.2.4 Caching

Existem diversos conteúdos interessante para serem alvo de *caching*. Actualmente, apenas se usa este mecanismo em duas situações:

1. Lista de fontes actuais, de forma a acelerar a pesquisa;
2. Último evento de cada fonte, para a aliviar os pedidos ao armazenamento persistente.

Existem muitos outros conteúdos que deviam ser alvo de *caching* como por exemplo os dados de sessão, caso o mecanismo de caching seja distribuído, e o histórico dos eventos das fontes mais subscritas pelos utilizadores.

```
if the operation result is in the cache:
   return the cached result
else:
   do the operation
   save the operation result in the cache
   return the result
```

Listagem 6.4: Algoritmo genérico de utilização do *Memcache*

0 *9ticks* usa como ferramenta de *caching* o *Memcached*. Este segue uma abordagem genérica e possuí clientes em diversas linguagens. Trata-se de uma cache LRU (*Least Recently Used*), distribuída, não replicada e com suporte para *timeouts*. Sempre que é realizado um pedido ao sistema de armazenamento, o algoritmo da Listagem 6.4 é executado.

6.2.5 Como e Quando Esquecer

Estando perante taxas de inserção bastante elevadas, facilmente se alcançará quantidades de informação histórica na ordem dos *terabytes*. Apesar de, actualmente, essa ordem ainda não ter sido alcançada, a existência de um mecanismo de eliminação da informação suficientemente antiga em junção com agregações a vários níveis, é crucial na redução dos requisitos de armazenamento.

Ou seja, queremos armazenar os dados actuais, agregar os mais antigos e eliminar os que já não têm interesse.

Por exemplo, considerando um índice da bolsa, existe o interesse em visualizar com elevada granularidade a informação mais recente e visualizar sumários, com diferentes granularidades, para tempos passados. Por outro lado existem fontes que deixam de gerar eventos, deixando de ter interesse. Por exemplo, após uma encomenda chegar ao seu destino, não interessa mais saber onde é que actualmente ela se encontra.

Actualmente, esta funcionalidade não se encontra implementada mas está prevista para um futuro próximo.

6.3 Crawler

De forma a verificar se ocorreu um evento, é necessário que o sistema faça *polling*, tempos a tempos, a cada fonte de eventos. Este é o módulo responsável por essa tarefa.

6.3.1 Escalonamento das Fontes

A título exemplificativo, considere-se que as fontes de eventos referidas na Tabela 6.10 encontram-se subscritas.

scheduler

Todo o processo tem por base um *scheduler* global (Figura 6.2). Este mantém duas filas: a fila de trabalhos por realizar (*job queue*) e a fila de trabalhos completos (*completion queue*). A *job queue* trata-se de uma fila com *delays* e contém as fontes de eventos que necessitam de ser *polled*. A *refresh rate* de cada fonte é usada como *delay*, permitindo assim definir quanto tempo depois vai

job queue

ID	Tipo	Propriedade	Actualização
1	Stock	AAPL	1 segundo
2	Twitter	johndoe	5 minutos
3	Stock	GOOG	3 segundos
4	Weather	Coimbra	1 hora
5	Stock	INTC	5 segundos

Tabela 6.10: Exemplo de fontes subscritas

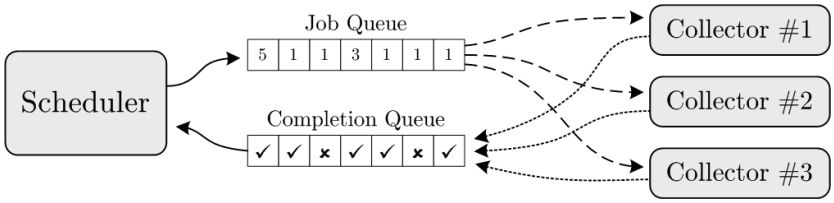


Figura 6.2: *Crawler*: Escalonamento

completion queue

ser o próximo *poll*. A *completion queue* trata-se de uma fila sem prioridades e contém os trabalhos terminados, quer com sucesso quer com insucesso.

collector

Cada *collector* retira trabalhos da *job queue*, executa-os e ao terminar coloca o resultado de cada trabalho na *completion queue*. Trata-se de uma abordagem descentralizada onde podem ser adicionados ou removidos *collectors* sem causar grande *overhead*.

6.3.2 Adapters

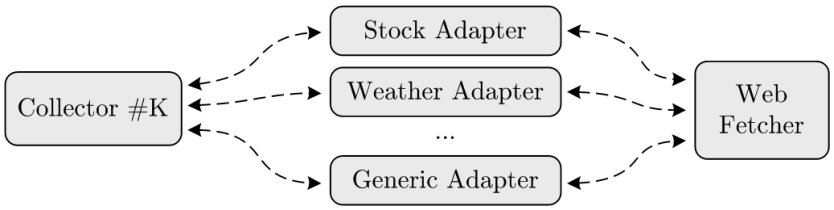


Figura 6.3: *Crawler*: Adaptadores de fontes

adapter

Cada categoria de fontes de informação (`STOCK`, `WEATHER`, `GOOGLE`, `EBAY`, `RSS`, `TWITTER` e genericamente `USER`-*defined*) tem um *adapter* associado (Figura 6.3). Cada *adapter* implementa a lógica que permite detectar se ocorreu um evento de determinada fonte.

```
interface Adapter {
    Event collect(Source source) throws AdapterException;
}
```

Listagem 6.5: *Adapter interface*

De forma aos *collecters* poderem usar genericamente os *adapters*, estes têm que implementar a *interface* apresentada na Listagem 6.5.

A lógica para ir buscar a informação às páginas consiste em fazer um GET ao servidor *web* onde a informação está armazenada e, com o uso de expressões regulares ou de *XPath*, identificar e extrai os pedaços da página relevantes. Esta abordagem pode ter alguns problemas, solucionáveis com a integração do *Web Fetcher* (Secção 6.3.3).

Para os casos em que os eventos não provém de uma página *web* específica ou são fornecidas APIs para o acesso directo à fonte, como é o caso da categoria *RSS*, são utilizadas bibliotecas para extrair a informação.

6.3.3 Web Fetcher

A *web* actual é cada vez mais dinâmica, com páginas que fazem uso de tecnologias *JavaScript* e *CSS*, alterando a estrutura e conteúdo das páginas em *runtime*. Ou seja, abrir uma conexão com o servidor onde a página está alojada, fazer um GET e usar o HTML devolvido pode resultar em incorrecções dado que existem alterações por processar. A extracção do HTML final da página para posterior processamento tem que ter essas alterações em atenção.

web fetcher

O *Web Fetcher* consiste numa aplicação que corre sob a *framework XULRunner*, a qual contém contém o *layout engine Gecko*, usado actualmente pelo *Mozilla Firefox*.

O maior desafio encontra-se em saber quando é que a página está efectivamente num estado adequado para leitura. Muitas das vezes o dinamismo é constante, por exemplo, quando um pedaço da página sofre actualizações segundo a segundo através de *Ajax*.

Sempre que um *adapter* necessitar de extrair informação proveniente de páginas *web*, pede a este módulo o HTML correspondente. Esta funcionalidade encontra-se parcialmente implementada, não estando actualmente integrada com o sistema.

6.3.4 Mudanças na Estrutura das Páginas

XPath

Cada *adapter* sabe dentro de uma página qual a localização (*XPath*) da informação relevante. Com o passar do tempo, a estrutura da página tende a mudar, o que dificulta o *parsing* e identificação da informação. Para que o *9ticks* seja tolerante a este tipo de falhas, tem que reconhecer e adaptar-se a estas alterações.

Este problema pode ser atacado de duas formas:

1. Desenvolver um algoritmo que detecte a semelhança da estrutura actual com outra previamente guardada e a partir daí extrapolar onde deve estar efectivamente a informação na página;

2. Generalizar as *XPaths* fazendo uso de atributos e classes em deterimento à *path* completa até ao elemento. Por exemplo, consideremos a seguinte *XPath*, que identifica a célula de uma tabela:

```
/html/body/div[2]/table[@id='resultados']/tr[2]/td[3]
```

Se a tabela mudar de posição na página, por exemplo, estar dentro de uma *tag* `<span>` em vez de estar dentro da segunda `<div>`, deixaria de funcionar.

```
/html/body/span/table[@id='resultados']/tr[2]/td[3]
```

No entanto, a tabela tem o atributo *id* definido, o qual numa página bem comportada é unívoco. Assim sendo, podemos reescrever a *XPath* indicando que estamos interessados naquela tabela, independentemente da sua posição no documento:

```
//table[@id='resultados']/tr[2]/td[3]
```

6.4 Web Server

Este módulo estabelece a ponte entre o *core* do sistema e os componentes *Web Client* (página *web*) e *Firefox Extension* (*plug-in* do *browser Mozilla Firefox*).

Jetty *Cometd*

A escolha do servidor *web Jetty* recaiu sobre duas das suas características: *footprint* reduzida e inclusão nativa do *Cometd* (uma implementação do protocolo *Bayeux* [6]).

Acompanhar as fontes em tempo real é um requisito fulcral do sistema. É necessário que os eventos cheguem ao *browser* o mais depressa possível. No entanto, o protocolo HTTP não contempla nenhuma forma directa do servidor iniciar uma comunicação com o *browser* quando ocorre um evento. É esta a falha que o *Comet* tenta colmatar.

Comet

Notar que estamos a falar de dois termos distintos: *Comet* e *Cometd*. *Comet* permite simular o inicio de uma comunicação do servidor para o *browser*. *Cometd* é uma implementação do protocolo *Bayeux* e usa *Comet* para o conseguir.

server push

Comet trata-se de um *design pattern*, conhecido também por *Reverse Ajax* [62], *HTTP Streaming* [36] ou *server push*, que permite ao servidor web fazer *push* da informação para o *web browser* sem que este tenha efectuado um pedido em específico. Isto é possível uma vez que o *browser* mantém activa uma conexão com o servidor *web*, o qual apenas responde quando tiver alguma mensagem a transmitir para o *browser* (*long-polling*). O *browser* após receber uma mensagem ou em *timeout* da conexão, a ligação é restabelecida.

long-polling

O protocolo *Bayeux*, e por conseguinte a implementação *Cometd*, tem por base o modelo *publish/subscribe* baseado em canais (ou tópicos). Os *browsers*, através de uma biblioteca com a implementação cliente do protocolo em *JavaScript*, subscrevem os canais que desejam monitorizar. Sempre que alguma mensagem seja publicada num canal, todos os seus subscritores serão notificados.

publish/subscribe

Tipicamente, os servidores *web* tradicionais utilizam uma *thread* por cada pedido que lhe é feito. Acontece que o *Comet* necessita que a conexão se mantenha aberta à espera que algo aconteça. Isto implica que o servidor vai ter abertas,

em média e na melhor das hipóteses, uma ligação constantemente aberta por cada *browser* cliente e, consequentemente, uma *thread*, o que não é escalável.

A solução passa por conseguir dissociar a conexão da *thread* sempre que não exista nada a enviar para o *browser*. Assim, essa *thread* poderá ser reutilizada para outra conexão. O *Jetty* consegue isso com o uso de *Continuations* [10]. De qualquer forma, o servidor tem que manter todas as conexões abertas e encontra-se limitado pelo máximo número de conexões suportadas pelo sistema.

continuations

6.5 Web Client

Este componente corresponde à página *web* do *9ticks*. É nesta página que o utilizador tem o seu *dashboard* e pode personalizá-lo através da adição de fontes de informação e alterando a forma de as visualizar (Figura 6.4).

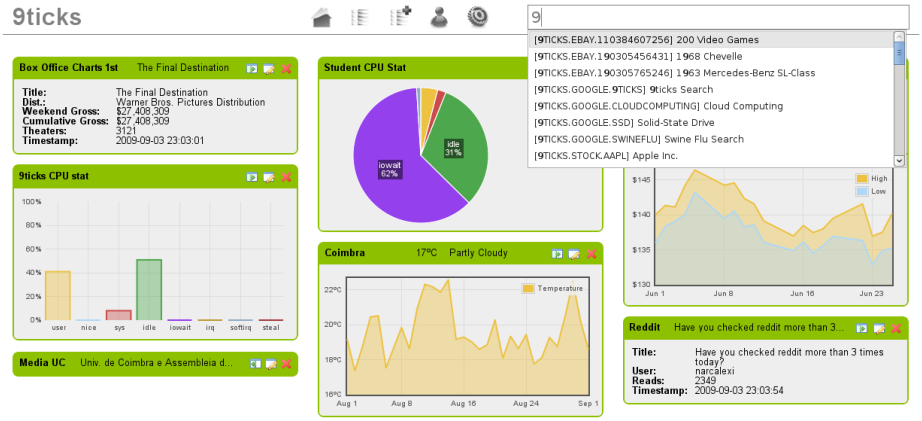


Figura 6.4: Página *web* do *9ticks*

Esta página é completamente dinâmica e construída através de *JavaScript*, o que permite aumentar a interactividade com o utilizador. Para facilitar a escrita do código foi usada a biblioteca de *JavaScript jQuery*. Antes de continuar a ler este capítulo é recomendada a leitura do Capítulo 4 – Visão do Utilizador.

jQuery

6.5.1 Dashboard

O utilizador pode adicionar fontes ao seu *dashboard* através da caixa de texto no canto superior direito do ecrã (Figura 6.4). Consoante a escrita nessa caixa serão apresentadas algumas sugestões, das quais o utilizador deve escolher uma.

Sempre que exista comunicação activa e com o servidor, uma notificação *Gmail-like* com o texto *loading* será apresentada no topo da página. Esta permite dar ao utilizador a sensação que deve aguardar.

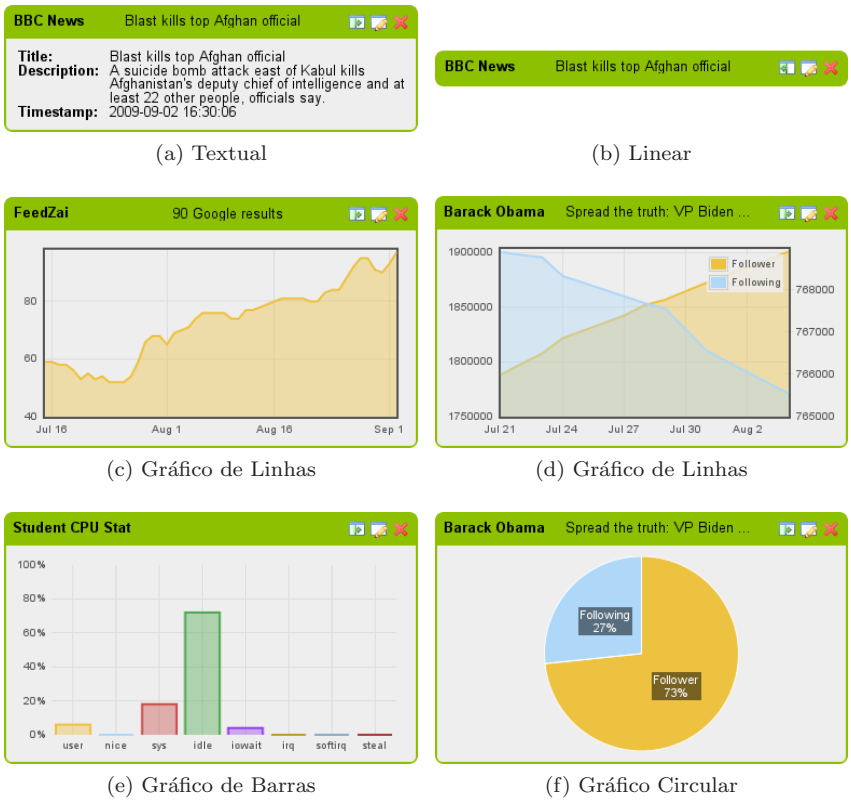
Como base do *dashboard* foi utilizado o *plugin* para o *jQuery EasyWidgets*. Este permite criar o ambiente dinâmico onde se pode mover as fontes de local.

Caso o utilizador esteja autenticado, sempre que o *dashboard* seja alterado (adicionar uma fonte, remover uma fonte, mover uma fonte e alterar visualização de uma fonte) este é automaticamente guardado de forma a poder ser restaurado posteriormente.

6.5.2 Visualizações

Cada fonte tendo associado um *namespace* hierárquico, facilita a definição de formas de visualização para categorias e para fontes em específico. Ou seja, podemos definir como visualizar todas as fontes relativas à bolsa de valores, `9TICKS.STOCK`, bem como definir visualizações em particular para, por exemplo, a Apple Inc., `9TICKS.STOCK.AAPL`.

Os tipos de visualizações podem dividir-se em duas categorias: visualizações pontuais ou históricas. As visualizações pontuais, ou instantâneas, apresentam o estado da fonte de informação num determinado instante temporal, ou seja, apresentam apenas um evento. As históricas apresentam, para uma janela temporal, a evolução da fonte de informação.



(a) Textual

(b) Linear

(c) Gráfico de Linhas

(d) Gráfico de Linhas

(e) Gráfico de Barras

(f) Gráfico Circular

Figura 6.5: Exemplos de visualizações

Na Figura 6.5 encontram-se diferentes visualizações actualmente suportadas pelo *9ticks*. As forma mais básicas de visualizar informação são *textualmente* (Figura 6.5a) e *linearmente* (Figura 6.5b). Graficamente, podemos visualizar informação pontual através de gráficos de barras (Figura 6.5e) ou através de gráficos circulares (Figura 6.5f). A informação histórica apenas pode ser visualizada por meio de gráficos de linhas (Figuras 6.5c e 6.5d).

É de notar que a mesma fonte de informação pode ser visualizada de diferentes formas, tal como é exemplificado pelas Figuras 6.5d e 6.5f.

É de realçar ainda que é possível personalizar os gráficos, por exemplo, a cor (linhas de cor diferente na Figura 6.5d), formatação dos eixos (símbolo de percentagem na Figura 6.5e) e múltiplos eixos para diferentes escalas (Figura 6.5d).

Flot

As visualizações gráficas são geradas pela biblioteca *Flot* [25] recorrendo apenas a HTML, CSS e *JavaScript*, ou seja, não necessita da instalação de *plugins* como *Flash* ou *Silverlight*. Os gráficos ao serem criados por *JavaScript* permitem uma integração uniforme com toda a restante página *web*.

6.5.3 Comunicação com o Servidor

Ajax
JSON

A comunicação entre o *browser* e o servidor é feita exclusivamente em *Ajax* usando a representação JSON. *JavaScript Object Notation* trata-se de um formato textual independente da linguagem de programação. A sua interpretação através de *JavaScript* é trivial, tendo-se tornado o formato de eleição para a transmissão de informação estruturada pela Internet.

A comunicação com o servidor é realizada quase sempre de forma assíncrona, salvo alguns casos em que é aconselhável manter a sincronia, por exemplo, no processo de autenticação.

Cometd

A tecnologia *Cometd*, introduzida na Secção 6.4, é ideal para o *9ticks*, uma vez que permite criar um canal por cada fonte de informação. Estes canais são subscritos pelos browsers cliente consoante as fontes que o utilizador seleccionou. Sempre que ocorre um novo evento é colocado no canal correspondente e todos os clientes nele subscritos são automaticamente notificados.

Na Figura 6.6 encontra-se o diagrama de sequência da comunicação do *browser* com o servidor *web*. Notar que o servidor de *Cometd* faz efectivamente parte do servidor *web*, correndo num URL *mapping* particular, por exemplo, `http://localhost/cometd`. De qualquer forma, encontram-se separados para facilidade de compreensão.

Inicialmente, *a)* o *browser* faz um pedido síncrono ao servidor *web* indicando-lhe qual a fonte de informação que pretende subscrever. O servidor responde indicando os detalhes da fonte pedida e o seu último evento. É com estes dados que vai ser adicionado uma nova entrada ao *dashboard* do utilizador.

De forma a manter o *dashboard* constantemente actualizado, *b)* o *browser* pede ao servidor de *cometd* para subscrever o canal correspondente a essa fonte. A partir desse momento, *c)* sempre que um evento seja publicado no canal subscrito pelo *browser*, este será notificado (bem como todos os outros *browsers*

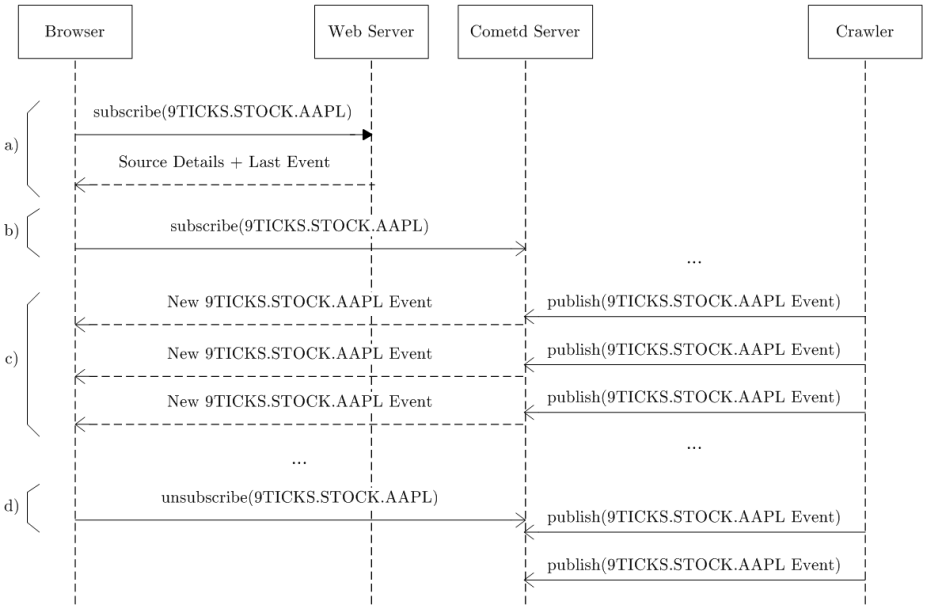


Figura 6.6: Comunicação do *browser* com os servidores

nele subscritos). Notar que o *Crawler*, sempre que detecta que ocorreu um novo evento, coloca-o no canal do servidor *cometd* correspondente.

Por fim, *d)*, quando o utilizador indica que não quer mais subscrever determinada fonte, o *browser* pede ao servidor *cometd* para remover a subscrição do canal correspondente.

Com mais de que um *browser* na equação (Figura 6.7), os passos continuam a ser os mesmos: subscrever canal, receber assincronamente os eventos desse canal e remover subscrição.

Para um número elevado de fontes, a probabilidade de existirem fontes que não se encontram num determinado momento subscritas, é elevada. Assim, um canal só existe quando existem subscritores. Ou seja, o primeiro subscritor de um canal faz com que o servidor *cometd* crie o canal e a último a remover a subscrição resulta na eliminação do canal.

6.6 Firefox Extension

Este componente corresponde a um *plugin* para o *browser Mozilla Firefox*. É com o auxílio dele que é possível criar novas fontes de informação. Desta forma, o utilizador não se encontra limitado às categorias de fontes predefinidas no sistema, permitindo-lhe gerar conteúdos personalizados.

XUL A extensão é desenvolvida em XUL [91] e *JavaScript*. Antes de se ter optado por uma extensão, colocou-se em hipótese outra alternativa. Esta encontra-se descrita na Secção 6.7 – Tentativas Falhadas.

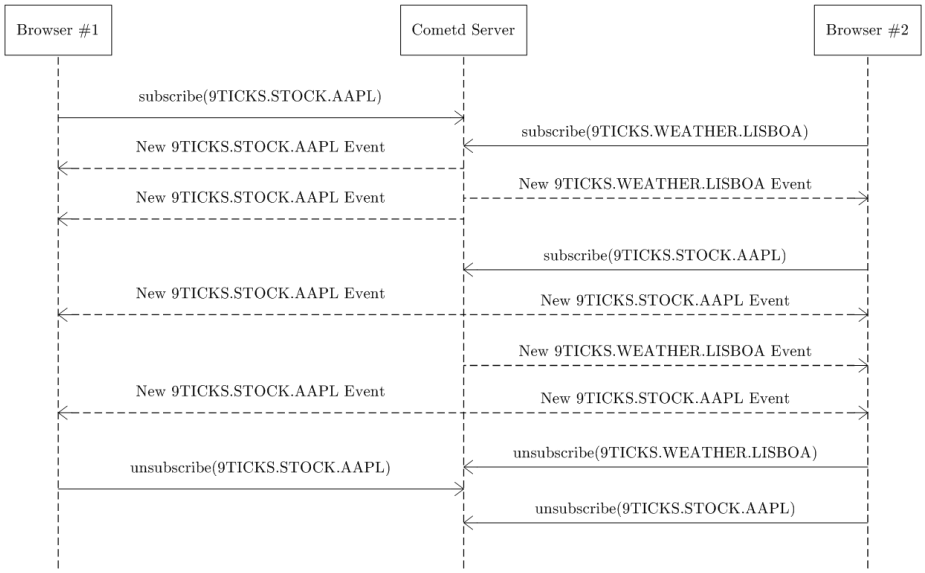


Figura 6.7: Comunicação de vários *browsers* com o servidor de *cometd*

As suas funcionalidades principais, e como se processa a interacção com o utilizador, estão descritas no Capítulo 4 – Visão do Utilizador. A sua leitura é recomendada para a correcta compreensão desta secção.

6.6.1 Identificação dos Elementos HTML

O objectivo principal deste componente é permitir ao utilizador identificar as porções da página *web* que contêm informação do seu interesse.

Para isso, foi desenvolvido um mecanismo que efectua pequenas alterações na página seleccionada pelo utilizador, permitindo que com o auxílio do rato os elementos fiquem sobressaídos.

```
for each HTML element inside the document:
    add on-mouse-over listener
    add on-mouse-click listener

onmouseover:
    prevent event propagation
    highlight element

onmouseclick:
    prevent event propagation
    build the xpath to the the clicked element
    add xpath to the selected elements list
```

Listagem 6.6: Identificação de elementos HTML

Este mecanismo altera o estilo e os eventos de todos os elementos HTML do documento (página *web*). Na Listagem 6.6 encontra-se descrito o algoritmo simplificado deste mecanismo e que está, efectivamente, implementado em *JavaScript*. O que ele faz é injectar *event listeners* em todos os elementos

event listeners

da página que permita detectar quando o rato passa por eles e quando os selecciona. É importante que a primeira coisa a fazer quando ocorre um evento do rato seja evitar a propagação desse evento, uma vez que a própria página pode estar também a observá-los.

6.6.2 Construção da XPath

Quando um elemento é seleccionado com o rato, é necessário determinar a sua localização na página. A forma mais simples de representar essa localização é com o uso de *XPaths*.

XPath

```
let element be the clicked html element

while element:
    add element to xpath

    if element has id attribute:
        add id to xpath
    else:
        if element has siblings:
            add offset to xpath

    element = element's parent
```

Listagem 6.7: Construção da *XPath*

Na Listagem 6.7 encontra-se descrito o algoritmo que constrói iterativamente a *XPath*. Ele começa no elemento seleccionado e sobe na hierarquia do documento, elemento a elemento, até chegar à raiz. Para cada elemento verifica se tem um identificador unívoco e adiciona-o à *XPath*. Caso contrário verifica se ao mesmo nível existem outros elementos do mesmo tipo de *tag* e indicar o *offset* entre eles.

Na realidade, tal como está mencionado na Secção 6.3.4, o algoritmo pode parar a partir do momento em que encontra um elemento com identificador.

6.6.3 Detecção dos Tipo de Dados

A detecção correcta do tipo de dados dos elementos seleccionados é de extrema importância. Actualmente, apesar de serem detectados mais tipos de dados, apenas são efectivamente usados os tipos *integer*, *float* e *string*.

A sua detecção é feita através de expressões regulares. Estas são *matched* com o conteúdo do elemento seleccionado, da mais específica à mais genérica. Por exemplo, para os três tipos de dados mencionados, a ordem é: *integer* → *float* → *string*.

A lista de expressões regulares tem que ser bastante completa ao ponto de abranger as diversas representações que o mesmo tipo de dados pode adquirir. Por exemplo, o número decimal `1234567.89` pode ser representado, a título exemplificativo, por `1234567,89` ou `1 234 567,89` ou `1.234.567,89`, dependendo da região do globo. O mesmo se aplica para outros tipos de dados mais complexos como datas e horas. É preciso ter especial atenção a todos os casos possíveis e dar sempre a possibilidade ao utilizador de fazer *override* do tipo de dados detectado automaticamente.

6.6.4 Comunicação

A comunicação com o servidor é bastante reduzida, limitando-se apenas a enviar a descrição da nova fonte de informação para o servidor *web*. Isto é realizado uma vez mais através de *Ajax* mas desta vez recorrendo directamente a uma instância do objecto *XMLHttpRequest*.

6.7 Tentativas Falhadas

Ao longo do desenvolvimento, foram usadas algumas ferramentas que por qualquer motivo não estavam à altura do problema, tendo que ser substituídas. Nesta secção indicam-se os problemas e respectivas soluções seguidas.

XML-RPC

A comunicação entre os componentes *core* do sistema era inicialmente feita através de XML-RPC. Esta foi usada pela sua simplicidade em representar a informação e pela sua baixa latência na comunicação. Aconteceu que, à medida que o sistema foi crescendo e as representações dos dados começaram a ficar mais complexas, a simplicidade do *XML-RPC* passou de vantagem para limitação.

Actualmente utiliza-se para comunicação o *Apache Thrift*, revelando-se até à data bastante eficaz (Secção 6.1.4).

row store

Antes de ser usado *Hypertable* para o armazenamento, foram experimentadas bases de dados relacionais, em particular, PostgreSQL. No início parecia comportar-se bem mas, à medida que o volume de eventos guardados começou a subir, o seu desempenho começou a decrescer drasticamente. O sistema relacional revelou-se bastante ineficiente para lidar com *queries* temporais. Outro problema é o facto de serem armazenados dados tão heterogéneos, muitas vezes apenas determinados em *runtime*, que uma base de dados horizontal (*row store*) nem sempre é a melhor solução. De qualquer forma, foram ajustados os índices e criados procedimentos PL/SQL e o sistema aguentou durante mais umas semanas até ficar, outra vez, demasiado lento.

column store

Foi necessária a mudança para um sistema de armazenamento vertical (*column store*). Existiu a hipótese entre *HBase* e *Hypertable*, ambos verticais e temporais, optando-se pelo primeiro com base em suas provas dadas.

No entanto, apesar de o *HBase* tratar de forma especial o tempo (todos os registos têm obrigatoriamente um *timestamp*, e pode ser consultado o valor que um campo tinha numa determinada revisão), revelou-se ineficaz em consultar todas as revisões entre dois instantes temporais. É uma funcionalidade prevista para o *HBase* mas que ainda não se encontra implementada.

O *Hypertable* preenche a falha do *HBase* e tem mantido, até à data, uma boa performance à medida que o volume de informação cresce (Secção 6.2).

"Results! Why, man, I have gotten a lot of results. I know several thousand things that won't work."

Thomas A. Edison

7 Resultados

Actualmente, o *9ticks* conta com pouco mais que 100 fontes de informação nas diversas categorias, estando a ser colectadas neste preciso momento. Os eventos destas fontes estão a ser armazenados e agregados aos diversas níveis (todos os minutos, horas, dias, semanas, meses e anos). Notar que esta quota de fontes apenas se observou nos últimos dois meses.

Para cerca de 100 fontes de informação, estão a ser realizados aproximadamente 3 pedidos por segundo (≈259200 pedidos por dia). Estes pedidos resultam em cerca de 86MB de informação diários (≈2.67GB por mês). Actualmente, o sistema de armazenamento *Hypertable* contém aproximadamente 10GB de informação, não comprimida, respeitante apenas aos eventos que estão a ser recolhidos em tempo real (não estamos a contabilizar o espaço ocupado pelas agregações que estão a ser efectuadas).

O *9ticks* corre na sua totalidade em apenas um computador convencional[1]. Estamos a falar não só no *Hypertable*, *Memcached* e *Jetty*, mas também no *Persistence*, *Aggregator*, *Scheduler* e vários *Crawlers*.

De qualquer forma, o sistema mantém-se rápido e eficiente ao ponto de estar mais que um mês ligado a correr sem pausa. Notar que é nesse computador que se realiza, em simultâneo, o desenvolvimento do *9ticks*.

Quanto ao produto final, em termos de *software* para o utilizador, encontra-se funcional a página *web* e o *plugin* para o *browser*. Estes provam por completo a viabilidade do *9ticks* a uma escala maior.

Do ponto de vista científico foram produzidos três artigos no decorrer do estágio. Um foi aceite num *workshop* co-localizado numa conferência (Anexo A), outro publicado numa revista (Anexo B) e o terceiro, com carácter demonstrativo, aguarda por aceitação numa outra conferência (Anexo C).

[1] AMD *Athlon*™ 64 X2 Dual Core (2 × 2.4GHz), 4GB RAM e HDD SATA 160GB @ 7200RPM (≈ 3 anos de idade)

"Dude! We haven't hit legendary yet. We're only at the LE, we've still got the GEN, the DA, the RY."

Barney Stinson – How I Met Your Mother

8 Conclusão e Trabalho Futuro

A cada instante, milhares de fontes de informação estruturada espalhadas pela *web* são alvo de actualização. Actualmente, os motores de pesquisa percorrem a *web* à procura de nova informação para indexar e arquivar. No entanto, eles não estão preparados para capturar fontes de informação com actualização muito frequente. Na maior parte dos casos, a informação que existiu perdeu-se com o passar do tempo.

O *9ticks* permite resolver esse problema, recolhendo, armazenando, agregando e interpretando a informação efémera outrora perdida, possibilitando ao utilizador acompanhá-la em tempo real e visualizar, com calma, o passado.

Actualmente já se encontram desenvolvidos os principais módulos de um sistema com estas características, nomeadamente, o escalonador das visitas às fontes, os colectores, os agregadores, o armazenamento e o cliente que permita a consulta e visualização da informação. A próxima etapa passa por implementar um cliente móvel, um notificador de alertas e optimizar os módulos existentes actualmente. Transversalmente, é de extrema importância garantir que o sistema consegue escalar para milhares de eventos e centenas de utilizadores.

O *9ticks* é um projecto inovador, complexo e ambicioso. Quando aliado às necessidades do mercado, acreditamos que reúne as características chave para se tornar um produto de sucesso.

Bibliografia

[1] E. Adar, M. Dontcheva, J. Fogarty, and D. S. Weld. Zoetrope: Interacting with the ephemeral web. In *UIST*, pages 239–248. ACM, 2008.

[2] Alerts.com website. `http://www.alerts.com/`. Accessed September 1, 2009.

[3] Android documentation. `http://code.google.com/android/documentation.html`. Accessed September 1, 2009.

[4] Internet Archive website. `http://www.archive.org`. Accessed September 1, 2009.

[5] Atom website. `http://www.atomenabled.org/`. Accessed September 1, 2009.

[6] Bayeux Protocol – Bayeux 1.0draft1. `http://svn.cometd.com/trunk/bayeux/bayeux.html`. Accessed September 1, 2009.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *7th Symposium on Operating Systems Design and Implementation (OSDI'06)*, pages 205–218, 2006.

[8] Google Chrome website. `http://www.google.com/chrome`. Accessed September 1, 2009.

[9] Cometd Project website. `http://cometdproject.dojotoolkit.org/`. Accessed September 1, 2009.

[10] Jetty Continuation website. `http://docs.codehaus.org/display/JETTY/Continuations`. Accessed September 1, 2009.

[11] Coral 8 documentation. `http://www.coral8.com/developers/documentation.html`. Accessed September 1, 2009.

[12] CORBA Specification. `http://www.omg.org/technology/documents/corba_spec_catalog.htm`. Accessed September 1, 2009.

[13] Crowbar website. `http://simile.mit.edu/wiki/Crowbar`. Accessed September 1, 2009.

[14] Common Format and MIME Type for Comma-Separated Values (CSV) Files. `http://tools.ietf.org/html/rfc4180`. Accessed September 1, 2009.

[15] Dapper website. `http://www.dapper.net/`. Accessed September 1, 2009.

[16] Django documentation. http://docs.djangoproject.com/en/dev/. Accessed September 1, 2009.

[17] .NET Remoting Overview. http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.71).aspx. Accessed September 1, 2009.

[18] Eddie RSS and Atom Parser for Java website. http://www.davidpashley.com/projects/eddie.html. Accessed September 1, 2009.

[19] ehcache website. http://ehcache.sourceforge.net/. Accessed September 1, 2009.

[20] Enterprise JavaBeans Technology. http://java.sun.com/products/ejb/. Accessed September 1, 2009.

[21] T. Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[22] Esper website. http://esper.codehaus.org/. Accessed September 1, 2009.

[23] Mozilla Firefox. http://www.mozilla.com/en-US/firefox/. Accessed September 1, 2009.

[24] Flickr website. http://www.flickr.com/. Accessed September 1, 2009.

[25] Flot website. http://code.google.com/p/flot/. Accessed September 1, 2009.

[26] Gecko website. http://www.mozilla.org/newlayout/. Accessed September 1, 2009.

[27] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM Press.

[28] GigaSpaces website. http://www.gigaspaces.com/caching. Accessed September 1, 2009.

[29] Google Mashup Editor website. http://code.google.com/gme/. Accessed September 1, 2009.

[30] HBase Documentation. http://hadoop.apache.org/hbase/docs/current/. Accessed September 1, 2009.

[31] HDFS Architecture. http://hadoop.apache.org/core/docs/current/hdfs_design.html. Accessed September 1, 2009.

[32] Hibernate documentation. http://www.hibernate.org/5.html. Accessed September 1, 2009.

[33] HTML 4.01 Specification. http://www.w3.org/TR/html4/. Accessed September 1, 2009.

[34] HTMLCleaner website. http://htmlcleaner.sourceforge.net/. Accessed September 1, 2009.

[35] HtmlUnit website. http://htmlunit.sourceforge.net/. Accessed September 1, 2009.

[36] Ajax Patterns website. http://ajaxpatterns.org/HTTP_Streaming. Accessed September 1, 2009.

[37] Hypertable website. http://www.hypertable.org/. Accessed September 1, 2009.

[38] Internet Explorer website. http://www.microsoft.com/windows/products/winfamily/ie/default.mspx. Accessed September 1, 2009.

[39] iGoogle website. http://www.google.com/ig. Accessed September 1, 2009.

[40] iPhone website. http://www.apple.com/iphone/. Accessed September 1, 2009.

[41] JBoss Seam documentation. http://jboss.com/docs/index. Accessed September 1, 2009.

[42] Jetty website. http://www.mortbay.org/jetty/. Accessed September 1, 2009.

[43] Java ME Technology documentation. http://java.sun.com/javame/reference/apis.jsp. Accessed September 1, 2009.

[44] JMS Documentation. http://java.sun.com/products/jms/docs.html. Accessed September 1, 2009.

[45] jQuery website. http://jquery.com/. Accessed September 1, 2009.

[46] JSON website. http://json.org/. Accessed September 1, 2009.

[47] JSON-RPC Specification. http://json-rpc.org/wiki/specification. Accessed September 1, 2009.

[48] JSSh website. http://www.croczilla.com/bits_and_pieces/jssh/. Accessed September 1, 2009.

[49] log4j website. http://logging.apache.org/log4j/1.2/index.html. Accessed September 1, 2009.

[50] Mac OS X Dashboard. http://www.apple.com/macosx/features/300.html#dashboard. Accessed September 1, 2009.

[51] memcached website. http://www.danga.com/memcached/. Accessed September 1, 2009.

[52] Microsoft PopFly website. http://www.popfly.com/. Accessed September 1, 2009.

[53] MozRepl website. http://hyperstruct.net/projects/mozrepl. Accessed September 1, 2009.

[54] Windows Mobile website. http://www.microsoft.com/windowsmobile/en-us/default.mspx. Accessed September 1, 2009.

[55] My Yahoo! website. http://my.yahoo.com/. Accessed September 1, 2009.

[56] NetVibes website. http://www.netvibes.com/. Accessed September 1, 2009.

[57] Openkapow website. http://openkapow.com/. Accessed September 1, 2009.

[58] Orchestr8 Alchemypoint website. http://www.orch8.net/ap/. Accessed September 1, 2009.

[59] PageFlakes website. http://www.pageflakes.com/. Accessed September 1, 2009.

[60] Ruby on Rails documentation. http://rubyonrails.org/documentation. Accessed September 1, 2009.

[61] RRDtool Documentation. http://oss.oetiker.ch/rrdtool/doc/index.en.html. Accessed September 1, 2009.

[62] Reverse Ajax website. http://directwebremoting.org/dwr/reverse-ajax/index.html. Accessed September 1, 2009.

[63] Remote Method Invocation website. http://java.sun.com/javase/technologies/core/basic/rmi/. Accessed September 1, 2009.

[64] RSS 2.0 Specification. http://www.rssboard.org/rss-specification. Accessed September 1, 2009.

[65] S60 website. http://www.symbian.com/. Accessed September 1, 2009.

[66] Safari website. http://www.apple.com/safari/. Accessed September 1, 2009.

[67] Selenium website. http://seleniumhq.org/. Accessed September 1, 2009.

[68] SOAP Specification. http://www.w3.org/TR/soap/. Accessed September 1, 2009.

[69] Spring documentation. http://www.springsource.org/documentation. Accessed September 1, 2009.

[70] Strata website. http://www.kirix.com/. Accessed September 1, 2009.

[71] Streambase website. http://www.streambase.com/. Accessed September 1, 2009.

[72] Struts 2 documentation. http://struts.apache.org/2.x/docs/. Accessed September 1, 2009.

[73] Terracotta website. http://www.terracotta.org/. Accessed September 1, 2009.

[74] Apache Thrift website. http://incubator.apache.org/thrift/. Accessed September 1, 2009.

[75] Google Trends website. http://www.google.com/trends. Accessed September 1, 2009.

[76] Trident reference. http://msdn2.microsoft.com/en-us/library/aa741317.aspx. Accessed September 1, 2009.

[77] TurboGears documentation. http://docs.turbogears.org/1.0. Accessed September 1, 2009.

[78] Twitter4J website. http://yusuke.homeip.net/twitter4j/en/index.html. Accessed September 1, 2009.

[79] Watir website. http://wtr.rubyforge.org/. Accessed September 1, 2009.

[80] Web Clip website. http://www.apple.com/macosx/features/safari.html. Accessed September 1, 2009.

[81] WebKit website. http://webkit.org/. Accessed September 1, 2009.

[82] WebSlices website. http://www.ieaddons.com/en/webslices/. Accessed September 1, 2009.

[83] WebSundew website. http://www.websundew.com/. Accessed September 1, 2009.

[84] Webwag website. http://www.webwag.com/. Accessed September 1, 2009.

[85] Windows Vista Sidebar. http://www.microsoft.com/windows/windows-vista/features/sidebar-gadgets.aspx. Accessed September 1, 2009.

[86] Extensible Markup Language Specification. http://www.w3.org/TR/REC-xml/. Accessed September 1, 2009.

[87] Xml-rpc specification. http://www.xmlrpc.com/spec. Accessed September 1, 2009.

[88] XML Path Language Specification. http://www.w3.org/TR/xpath20. Accessed September 1, 2009.

[89] XML Query Language Specification. http://www.w3.org/TR/xquery/. Accessed September 1, 2009.

[90] XSL Transformations Specification. http://www.w3.org/TR/xslt. Accessed September 1, 2009.

[91] XML User Interface Language website. https://developer.mozilla.org/en/XUL. Accessed September 1, 2009.

[92] XULRunner website. https://developer.mozilla.org/en/XULRunner. Accessed September 1, 2009.

[93] Yahoo Pipes website. http://pipes.yahoo.com/. Accessed September 1, 2009.

[94] YAML Specification. http://www.yaml.org/spec/. Accessed September 1, 2009.

[95] YouTube website. http://www.youtube.com/. Accessed September 1, 2009.

[96] Zoetrope: Video demontration. http://www.cond.org/zoetrope.html. Accessed September 1, 2009.

[97] Zope documentation. http://www.zope.org/Documentation/. Accessed September 1, 2009.

A

9ticks – The Web as a Stream

Rafael Marmelo, Pedro Bizarro and Paulo Marques

1st International Workshop on Database Architectures for the Internet of Things (DAIT'2009) in conjunction with the 26th British National Conference on Databases (BNCOD'2009)

6th July 2009, University of Birmingham, UK

B

9ticks – The Web as a Stream (extended)

Rafael Marmelo, Pedro Bizarro and Paulo Marques

1st International Workshop on Database Architectures for the Internet of Things (DAIT'2009) in conjunction with the 26th British National Conference on Databases (BNCOD'2009)

September/October 2009, IETE Technical Review journal

C

9ticks – Streaming and Storing the Ephemeral Web

Rafael Marmelo, Pedro Bizarro and Paulo Marques

26th IEEE International Conference on Data Engineering (ICDE'2010) Demo Session

March 2010, Long Beach, California, USA

Waiting for Approval

D
Estado da Arte

O estado da arte foca-se em dois temas distintos. Numa primeira parte, apresentam-se algumas aplicações já existentes no mercado que possuem funcionalidades similares às requeridas para o *9ticks*. O segundo tema aborda as tecnologias que podem ser utilizadas para desenvolver esta aplicação, apresentando os seus pontos fortes e fracos, relacionando-os sempre que possível com os atributos de qualidade referidos na secção E.2.1.

D.1 Aplicações similares

Sendo o *9ticks* uma ferramenta bastante ampla no que toca a funcionalidades, as aplicações que de alguma forma se identificam com ela, foram divididas em quatro subsecções lógicas. Estas são apresentadas de seguida.

D.1.1 *Custom Home Pages*

Um dos requisitos para este estágio é apresentar visualmente a informação sob a forma de um painel, *dashboard*, sendo os conteúdos (*widgets* ou *gadgets*, como são usualmente chamados) nele apresentados escolhidos pelo utilizador. Existem várias aplicações que apresentam essa funcionalidade, destacando as seguintes:

- *Alerts.com* [2]
- *iGoogle* [39]
- *Netvibes* [56]
- *PageFlakes* [59]
- *My Yahoo!* [55]
- *Webwag* [84]

Todas as aplicações mencionadas são aplicações *web* e têm um painel ao qual o utilizador pode adicionar conteúdos predefinidos pelo sistema. Poder-se-ia falar de muitas outras aplicações *web*, mas todas elas seguem o conceito base das apresentadas. No entanto, é importante referir que existem aplicações que apresentam um *dashboard* mas que não são vocacionadas para a *web*. Temos por exemplo o *dashboard* do *Mac OS X* [50] e a barra lateral do *Windows Vista* [85].

O leque de conteúdos varia consoante a aplicação. No entanto, todas elas possuem *widgets* base que permitem consultar o estado meteorológico, visualizar resultados desportivos, aceder a boletins informativos, visualizar a informação de índices na bolsa de valores, manter uma lista de tarefas a realizar e interagir com serviços externos, como o *Flickr* [24] ou o *YouTube* [95].

De todo o conjunto de aplicações, duas incorporam funcionalidades que merecem ser mencionadas. A primeira aplicação, *Alerts.com*, permite definir alertas mediante regras, podendo estes ser recebidos por *E-mail*, SMS ou no computador, através de uma aplicação proprietária. A outra aplicação que se destaca é a *Netvibes*, que permite fazer *clipping* (ver subsecção D.1.2) de uma qualquer página *web* e colocá-la no *dashboard*.

Todas as aplicações fazem uso extensivo de *JavaScript* e AJAX de forma a melhorar a interface com o utilizador.

D.1.2 *Clipping the Web*

Clipping é a palavra utilizada para representar o acto de recortar artigos em revistas, jornais ou qualquer outro meio de comunicação impresso. Ganhou mais recentemente um novo sentido, aplicado ao recorte de páginas *web*. Esta é a forma mais simples de extrair informação da Internet. Existem duas aplicações que merecem ser destacadas a este nível:

- *Web Slices* [82]
- *Web Clip* [80]

A aplicação *Web Clip* foi introduzida com o *Mac OS X Leopard* e permite aos utilizadores criar *widgets* para o *dashboard* a partir de partes de quaisquer páginas *web*. A selecção de partes de páginas *web* é efectuada a partir do navegador *Safari*.

Web Slices apareceu recentemente com o navegador *Internet Explorer 8*, ainda em estado *beta* no momento da escrita deste documento. Esta aplicação apenas permite extrair informação de páginas que tenham sido alteradas de forma a suportarem *Web Slices*, o que constitui a sua maior desvantagem. Toda a aplicação corre dentro do *Internet Explorer*, não havendo integração com o sistema operativo.

Este tipo de aplicações apresenta a página *web*, ou parte dela, sem dar importância ao seu conteúdo. Como tal, o utilizador não tem possibilidade de efectuar qualquer outro tipo de operação que não esteja originalmente definida.

D.1.3 *Scrapping the Web*

Web scrapping consiste numa técnica de *screen scrapping* para extracção de informação a partir de páginas *web*. Esta abordagem é bastante mais poderosa que a descrita na subsecção D.1.2, uma vez que esta produz resultados estruturados. Estes resultados podem ser apresentados aos utilizadores de forma personalizada ou mesmo ser usados por outras aplicações. Quanto a este conceito, é importante referir as seguintes aplicações:

- *Dapper* [15]
- *Openkapow* [57]
- *WebSundew* [83]
- *Strata* [70]

Tratam-se de aplicações para diferentes propósitos, mas todas contêm módulos que efectuam *scrapping* de páginas *web*.

Strata é uma aplicação para a análise de dados de forma interactiva, que suporta, como valores de entrada, tabelas HTML [33] provenientes de qualquer página. Quer o *Openkapow* como o *WebSundew* são ferramentas bastante poderosas mas complicadas de utilizar.

Enquanto que as três aplicações mencionadas anteriormente se tratam de executáveis *standalone*, o *Dapper* corre sob qualquer navegador *web*. Trata-se do exemplo com melhor interacção com o utilizador, sendo bastante simples de usar. Este permite a extracção de conteúdos repetitivos de qualquer página e a partir destes criar um ficheiro de saída, suportando diversos formatos. Notar que a identificação e extracção de conteúdos repetitivos é um dos requisitos deste estágio e o *Dapper* consegue isso de forma bastante eficiente.

Uma vez que a maioria das páginas *web* são construídas à base de HTML e XHTML, estas aplicações baseiam a extracção de dados em tecnologias como *XPath* [88], *XQuery* [89], XSLT [90] e expressões regulares.

Nos dias que correm existem cada vez mais páginas que fazem uso de geração de código dinâmico através de chamadas assíncronas a serviços com vista a aumentar a interacção com o utilizador. Normalmente, com o aumento da interactividade, aumenta também a dificuldade no processamento das páginas por este tipo de aplicações.

Notar ainda que o *Dapper*, ao permitir que utilizador visualize e seleccione os conteúdos que pretende através da sua própria página, abre a porta a um conjunto potencial de problemas. Os dois principais são *Cross-Site Scripting*, XSS, e *Cross-Site Request Forgery*, XSRF.

D.1.4 *Mashup Editors*

Mashup, numa perspectiva virada para a Internet, consiste numa aplicação que combina conteúdos de várias fontes, criando um produto completo. Existem várias editores que permitem fazer este tipo de aplicações, como por exemplo:

- *Yahoo Pipes* [93]
- *Microsoft PopFly* [52]
- *Google Mashup Editor* [29]
- *Orchestr8 AlchemyPoint* [58]

Todos os editores são bastante bons, no entanto, o *Yahoo Pipes* merece ser destacado pela sua simplicidade de utilização. Ele compara o seu funcionamento aos *pipes* em *Unix*: comandos simples que podem ser combinados, criando um resultado personalizado.

Uma desvantagem comum é o facto de apenas suportarem fontes estruturadas (por exemplo RSS [64], *Atom* [5], JSON [46], XML [86] e CSV [14]), não apresentando técnicas robustas para a extracção de informação a partir de qualquer página *web*. No entanto, os resultados produzidos pelas aplicações mencionadas na subsecção D.1.3 podem servir como entrada para a criação de *mashups* por estas aplicações.

D.1.5 *Web History*

A única aplicação que se encontra nesta categoria é o *Zoetrope* [1, 96]. Este permite ao utilizador visualizar informação passada, relacionar fontes de informação e definir visualizações personalizadas (por exemplo, gráficos e *timelines*).

Trata-se de uma aplicação *standalone*, desenvolvida em *Java*. O número de fontes de eventos (páginas *web*) suportadas é actualmente reduzido.

O utilizador tem a possibilidade de definir uma área na página onde haja actualização (*lenses*), controlando depois uma *timeline* que permite voltar atrás no tempo e visualizar informação passada. Este sistema mostra a página tal e qual como ela era apresentada (incluindo imagens), pelo que gera um grande volume de dados.

D.2 Tecnologias

Nesta subsecção são apresentadas várias tecnologias às quais se volta a fazer referência no Capítulo 6 – Implementação, ao ser descrita a organização e funcionamento interno do *9ticks*.

D.2.1 Comunicação

É importante para qualquer aplicação a escolha dos mecanismos de comunicação mais adequados. De seguida apresentam-se alguns dos mecanismos considerados mais relevantes e estabelece-se uma breve comparação.

Web Services

Existem diversos protocolos que podem ser utilizados quando se fala de *web services*. Dentro dessa lista foram escolhidos quatro que se destacam pela sua simplicidade ou pela grande aceitação no mercado.

Ao longo da descrição apresentam-se exemplos de pedido e resposta para cada um dos tipos de *web services*. Imagine-se que se pretende visualizar o último valor do índice da bolsa de valores `GOOG`.

XML-RPC [87] é um protocolo de comunicação muito simples que codifica as mensagens em XML e utiliza HTTP como mecanismo de transporte. Define propositadamente uma lista reduzida de tipos de dados escalares e complexos: *integer*, *double*, *string*, *boolean*, *datetime*, *base64*, *array* e *struct*, que consiste num *array associativo*. Tratando-se de um protocolo bastante simples existem implementações nas mais variadas linguagens. As característica mais importantes do XML-RPC são a sua estabilidade, portabilidade e *performance*. A Listagem D.1 e D.2 apresentam um exemplo de pedido e resposta usando XML-RPC.

```
POST /stock HTTP/1.1
Host: http://www.9ticks.com

<?xml version="1.0"?>
<methodCall>
  <methodName>GetLastTrade</methodName>
  <params>
    <param>
      <value><string>GOOG</string></value>
    </param>
  </params>
</methodCall>
```

Listagem D.1: XML-RPC *request*

```
HTTP/1.1 200 OK

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><double>324.70</double></value>
    </param>
  </params>
</methodResponse>
```

Listagem D.2: XML-RPC *response*

SOAP (*Simple Object Access Protocol*) [68] é o protocolo mais utilizado actualmente. É uma evolução do XML-RPC adicionando-lhe, de grosso modo, a possibilidade de implementar *user defined data types* através de XML *Schema* e tornou agnóstico o mecanismo de transporte. A grande desvantagem deste protocolo é o facto das mensagens terem vários níveis de encapsulamento, o que faz com que seja computacionalmente bastante pesado. A Listagem D.3 e D.4 apresentam um exemplo de pedido e resposta usando SOAP.

```
GET /stock HTTP/1.1
Host: http://www.9ticks.com

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
```

```
  <soap:Body xmlns:nt="http://www.9ticks.com/stock">
    <nt:GetLastTrade>
      <nt:Symbol>GOOG</nt:Symbol>
    </nt:GetLastTrade>
  </soap:Body>
</soap:Envelope>
```

Listagem D.3: SOAP *request*

```
HTTP/1.1 200 OK

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body xmlns:nt="http://www.9ticks.com/stock">
    <nt:GetLastTradeResponse>
      <nt:Trade>324.70</nt:Trade>
    </nt:GetLastTradeResponse>
  </soap:Body>
</soap:Envelope>
```

Listagem D.4: SOAP *response*

JSON-RPC (JavaScript Object Notation RPC) [47] é uma variante do XML-RPC, substituindo a codificação de mensagens de XML para JSON. À semelhança do XML, é também representado por texto. Oferece um conjunto de tipos de dados equivalentes ao oferecido pelo XML-RPC, definidos numa notação ao estilo do *JavaScript*. Isto permite que, em *JavaScript*, um simples `eval()` interprete a *string* JSON. A integração com o *JavaScript*, e o facto de existirem implementações em variadas linguagens, faz com que este protocolo seja ideal para transmitir assincronamente informação com o navegador *web*. Apesar de não ser tão estável como o XML-RPC, partilha com ele as características de portabilidade e *performance*. A Listagem D.5 e D.6 apresentam um exemplo de pedido e resposta usando JSON-RPC.

```
POST /stock HTTP/1.1
Host: http://www.9ticks.com

{ "method": "GetLastTrade", "params": ["GOOG"], "id": 123 }
```

Listagem D.5: JSON-RPC *request*

```
HTTP/1.1 200 OK

{ "result": 324.70, "error": null, "id": 123 }
```

Listagem D.6: JSON-RPC *response*

REST (Representational State Transfer) é uma tecnologia emergente. Utiliza o protocolo HTTP como mecanismo de transporte, usando os seus métodos PUT, GET, POST and DELETE. Todos os recursos estão disponíveis através de um URL. A troca de mensagens consiste na invocação de um URL através de um dos métodos HTTP. A resposta é agnóstica quanto ao conteúdo, não existindo a necessidade de usar os longos envelopes que o SOAP utiliza nas suas mensagens. No entanto, os MIME *types* mais usuais são XML, JSON ou YAML [94]. Tendo por base o HTTP, também o REST é *stateless*, o que lhe garante uma boa escalabilidade e *performance*. A Listagem D.7 e D.8 apresentam um

exemplo de pedido e resposta usando REST.

```
GET /stock/LastTrade/GOOG HTTP/1.1
Host: http://www.9ticks.com
```

Listagem D.7: REST *request*

```
HTTP/1.1 200 OK

<?xml version="1.0" xmlns:nt="http://www.9ticks.com/stock"?>
<nt:Trade>324.70</nt:Trade>
```

Listagem D.8: REST *response*

Apache Thrift

Thrift é uma *framework* desenvolvida pelo *Facebook* que permite aplicações escritas num vasto leque de linguagens comunicarem entre si de forma transparente (C++, *Java*, *Python*, PHP, *Ruby*, *Erlang*, *Perl*, *Haskell*, C#, *Cocoa*, *Smalltalk*, e *OCaml*).

Permite a definição de tipos de dados e serviços (métodos) através de uma linguagem intermédia (IDL), gerando o código cliente e servidor correspondentes nas diversas linguagens.

Outras Tecnologias

Existem muitas outras tecnologias que permitem efectuar a comunicação entre os diversos módulos aplicacionais. Poder-se-ia falar de CORBA [12], *Sockets*, *pipes*, *Shared Memory* ou IPCs. Todos elas permitiriam, com as suas limitações, resolver o problema da comunicação. No entanto, CORBA não é simples de usar e os outros falham na estruturação das mensagens.

Existe também um leque variado de tecnologias que se encontram fortemente ligadas a plataformas ou linguagens, como por exemplo RMI [63], JMS [44] e EJB [20] para o caso de *Java* e .NET *Remoting* [17] para a plataforma .NET. Se é objectivo da aplicação ser modular e escalável, a utilização deste tipo de tecnologias não é de todo apropriada. No entanto, o uso destas tecnologias pode fazer sentido dentro de silos aplicacionais fortemente acoplados.

D.2.2 Persistência

Os requisitos deste trabalho, no que toca à persistência, são conseguir assegurar o armazenamento dos eventos que chegam ao sistema muito frequentemente, permitir efectuar operações de pesquisa e consulta sobre eles e garantir que este seja resiliente a falhas. Neste ponto de vista, foram analisadas várias formas de conseguir cumprir estes requisitos, que se apresentam de seguida.

Sistema de Ficheiros

Utilizar directamente o sistema de ficheiros está, à partida, fora de questão. Qualquer volume de dados elevado tornaria o sistema instável. Esta abordagem não ofereceria formas optimizadas de efectuar operações sobre os dados, não

iria garantir sua resistência a falhas, nem seria escalável. Poder-se-ia utilizar um sistema em configuração RAID com *mirroring* mas, de qualquer forma, o problema não estaria de todo resolvido.

RRDtool

Trata-se de uma ferramenta desenvolvida para armazenar de forma sistemática e eficiente valores que estão a ser constantemente actualizados, permitindo a geração de gráficos a partir deles.

Apesar de não se enquadrar totalmente no que é requerido, apresenta uma funcionalidade necessária ao bom funcionamento deste trabalho: o sumariar a informação. O *RRDtool* [61] garante, através da eliminação da informação mais antiga e da consolidação da informação menos recente, que a informação de uma fonte de dados não cresce com o passar do tempo.

De qualquer forma, se o volume de dados for bastante elevado continua-se a ter o problema do sistema de ficheiro, ou mesmo o disco, não aguentar a carga. Neste caso poder-se-ia optar por utilizar o sistema em configuração RAID com *stripping*, de forma a fazer *load balancing* das escritas.

Outra abordagem passa por escrever os dados para memória RAM (utilizando por exemplo o sistema de ficheiros *tmpfs*), aliviando assim as leituras e escritas no disco. Tratando-se de memória volátil, é necessário a escrita de uma pequena ferramenta que a sincronize regularmente com o disco. Esta abordagem é apenas válida se a informação não for de extrema importância. A falha do sistema, quer por motivos de software, hardware ou externo ao sistema (interrupção de corrente eléctrica, por exemplo), pode levar à perda da informação por sincronizar.

Base de Dados

Uma base de dados é, de grosso modo, um conjunto de dados estruturados e persistentes. Actualmente, o modelo de estruturação mais usual é o modelo relacional, tendo por base tabelas, sendo estas compostas por linhas e colunas.

Quando o *bottleneck* do sistema está na leitura dos dados, a solução costuma ser replicar a base de dados. A configuração mais usual é *master/slave*, onde o *master* é para todos os efeitos quem detém a autoridade e as bases de dados *slave* sincronizam-se com ele. Esta abordagem paraleliza as leituras mas não aumenta o *performance* na escrita.

Quando o problema passa a ser a escrita de dados, opta-se por particionar a informação entre vários nós. A este processo dá-se o nome de *Sharding*. Atenção que *Sharding* não replica informação, limitando-se a distribuir os registos por várias tabelas. Escalar, é adicionar bases de dados e se necessário redefinir os critérios de particionamento. Esta abordagem aumenta drasticamente a *performance* e a *availability* do sistema, descuidando a sua *reliability*.

HBase (BigTable-like)

BigTable [7] é um sistema distribuído proprietário da *Google* para armazenamento de dados estruturados, desenhado para aguentar com muita informação. O *HBase* [30] trata-se de uma abordagem aberta que segue uma arquitectura muito similar à do *BigTable*.

O modo de utilização não podia ser mais simples. Os dados são organizados em tabelas, linhas e colunas. É fornecido uma interface que itera as linhas das tabelas e, fornecendo uma coluna e uma chave, devolve as linhas que contenham essa chave na coluna referida.

Recentemente foi adicionada uma *gateway* REST ao *HBase*, suportando serialização dos dados em XML e em JSON. Isto vem permitir que qualquer linguagem possa interagir de forma ainda mais simples com esta plataforma.

O *BigTable* corre sob o *Google File System* [27]. Similarmente, o *HBase* corre sobre o *Hadoop Distributed File System* [31]. Este último é responsável pela distribuição e replicação das tabelas por vários nós e suporta mecanismos de recuperação automática de falhas.

A junção de todos estes factores fazem com que sistema seja bastante escalável, obtendo valores altos de *performance*, *availability* e *reliability*.

Hypertable (BigTable-like)

Hypertable é um sistema de armazenamento *open source* baseado, tal como o *HBase*, no *BigTable*. Na realidade, os seus criadores trabalharam no *HBase*, tendo-se separado por não concordarem com algumas opções arquitecturais.

É escrito em C++ e pode correr sob diversos sistemas de ficheiros distríbuidos como o HDFS (mencionado na secção anterior) e o *Kosmos File System* (KFS).

É possível aceder directamente ao sistema através de C++ ou a partir de um leque variado de linguagens usando para isso a interface *Thrift*.

D.2.3 *Caching*

Quando a optimização a nível do armazenamento não é suficiente, principalmente ao nível das leituras, *caching* é o próximo passo a seguir.

Existe diversa informação que pode ser alvo de *caching*: os resultados de uma consulta à base de dados, fragmentos HTML (ou mesmo páginas inteiras), imagens e todos os outros recursos que forem bastante acedidos ou que sejam computacionalmente caros de calcular.

Existem várias aplicações que permitem fazer *caching* distribuído, a vários níveis, destacando as seguintes:

- *Memcached* [51] – *cache* genérica, com API em diversas linguagens;
- *GigaSpaces* [28] – *cache* ao nível da base de dados utilizando *Hibernate*;
- *Terracota* [73] – permite a partilha da *heap* do *Java* de forma a que *threads* que executem em diferentes JVMs possam interagir como de uma única JVM se tratasse;

- *Ehcache* [19] – pode ser utilizada como *cache* genérica, como também está optimizada para interagir com o *Hibernate* [32] e com *Java EE Servlets*.

A menos que se procure resolver um problema específico resolvido por outras *caches*, *Memcached* é a escolha mais simples e funcional.

D.2.4 Motor de Alertas

Um dos requisitos deste trabalho é a detecção de condições, sendo estas baseadas nos eventos recebidos. Existem alguns motores destinados ao processamento de eventos, dos quais se destacam três: *Coral 8* [11], *Streambase* [71] e *Esper* [22].

Quer o *Coral 8* como o *Streambase* são proprietários. Ambos são aplicações *standalone* sendo necessário definir formas de interagir com elas. Por outro lado, o *Esper* é *open-source* e tanto pode ser usado de forma *standalone* como *embedded*.

Todos eles têm possuem configurações para *clustering* e *high availability*.

D.2.5 HTML *Rendering*

Um parte central do estágio, senão mesmo a mais importante, é a identificação e extracção de informação de páginas *web*.

Recolher o HTML de um URL e utilizar XPath ou expressões regulares para extrair informação já é suficiente. As páginas actuais são dinâmicas, fazendo uso principalmente de AJAX, de forma a aumentar a interacção com o utilizador.

Usar essas técnicas de extracção continua a ser viável se o HTML a ser processado já tiver as alterações realizadas assincronamente. Este problema é resolvido ao ser utilizado um *layout engine* que suporte os standards HTML, CSS e JavaScript. Existem vários motores que permitem fazer isso:

- *Trident* [76] – utilizado pelo *Microsoft Internet Explorer* (também conhecido por MSHTML);
- *Gecko* [26] – utilizado pelo *Mozilla Firefox* (recentemente incorporado na *framework XULRunner* [92]);
- *WebKit* [81] – utilizado pelo *Apple Safari*;
- *HtmlUnit* [35] – *framework* para efectuar testes unitários a aplicações *web* em modo *headless*.

Para além destes *layout engines* existem outras aplicações que correm sobre eles e dão o controlo da navegação ao programador. Temos o exemplo do *Selenium* [67] e do *Watir* [79], os quais são ferramentas para teste unitário de páginas *web*.

Existem outras abordagens que permitem dar o controlo ao utilizador através da instalação de um *plugin* no *browser*. Por exemplo, para o *Firefox* (ou qualquer plataforma baseada no *XULRunner*), existem o *MozRepl* [53] e o *Crowbar* [13].

D.2.6 Cliente *Web*

Existem diversas frameworks para o desenvolvimento de aplicações web. Actualmentente estão em voga as seguintes, catalogadas por linguagem de programação:

- *Java* – *Struts2* [72], *Spring* [69], *JBoss Seam* [41]
- *Python* – *Django* [16], *Zope* [97], *TurboGears* [77]
- *Ruby* – *Ruby on Rails* [60]

Todas seguem o *design pattern Model-View-Controller* (MVC) para abstrair entre si as camadas de dados, de apresentação e de negócio.

Assumindo que o programador domina as linguagens equitativamente, a curva de aprendizagem das *frameworks* em *Python* e *Ruby* tende a ser menor.

D.2.7 Cliente Móvel

Um dos requisitos desejáveis deste trabalho é desenvolver um protótipo que exemplifique a utilização do *9ticks* num ambiente móvel. As plataformas móveis que actualmente estão em destaque são as seguintes:

- *Microsoft Windows Mobile* [54];
- *Apple iPhone* [40];
- *Google Android* [3];
- *Sun* Java Micro Edition [43];
- *Symbian* S60 [65].

Se o interesse for desenvolver uma aplicação móvel que abranja o maior número de dispositivos possível, *Java* ME é actualmente a opção a seguir.

O S60 encontra-se em grande parte dos telemóveis *Nokia* e *Samsung*, pelo que detém um quota de mercado bastante elevada.

Quanto ao *Windows Mobile*, *iPhone* e *Android* as suas quotas de mercado encontra-se em expansão mas ainda são relativamente baixas.

D.2.8 Extensão ao Navegador *Web*

Existe a necessidade de desenvolver uma aplicação que permita identificar informação de páginas *web*. Uma abordagem possível é criar um *plugin*, ou extensão, para um navegador *web*.

De seguida apresentam-se os navegadores mais utilizados actualmente, apontando algumas das suas características.

- *Microsoft Internet Explorer* [38] – apenas funciona em ambientes *Windows*;
- *Mozilla Firefox* [23] – funciona num leque variado de plataformas e tem uma boa documentação no que toca à criação de extensões;

- *Apple Safari* [66] – apenas funciona em ambientes *Windows* e *Mac*;
- *Google Chrome* [8] – apenas funciona em ambientes *Windows* e *Mac* e não existe ainda suporte oficial para extensões.

Se a portabilidade é essencial, *Mozilla Firefox* é a melhor opção. Apesar do *Internet Explorer* ainda deter a maior quota de mercado, esta tende a diminuir.

E
Requisitos

É objectivo deste anexo especificar os requisitos, quer funcionais como não-funcionais, da aplicação a desenvolver. Para melhor facilidade de leitura, os requisitos foram agrupados por partes lógicas do sistema. Para cada requisito foi atribuído um identificador, uma prioridade e uma descrição. A prioridade compreende dois tipos: Essencial e Desejável. Ao longo deste trabalho, apenas está prevista a implementação dos requisitos de prioridade Essencial.

E.1 Requisitos Funcionais

E.1.1 Interface com o Utilizador

Código:	REQ_0100
Requisito:	O sistema deverá ser acessível através de um navegador *web*.
Prioridade:	Essencial
Descrição:	Todas as operações a que o utilizador tenha acesso devem poder ser realizadas através desta interface.

Código:	REQ_0101
Requisito:	A página inicial do utilizador deverá seguir o estilo de um *dashboard*.
Prioridade:	Essencial
Descrição:	Nesse *dashboard* deverão ser representadas todas as fontes de eventos por ele seleccionadas.

Código:	REQ_0200
Requisito:	O sistema deverá ser acessível através de um dispositivo móvel.
Prioridade:	Desejável

Descrição:	Nesta interface, o utilizador poderá realizar um subconjunto de operações das disponíveis através do navegador *web*. As operações base desejáveis são visualizar as fontes de eventos por ele seleccionadas, adicionar novas fontes de eventos predefinidas e definir alertas. Dever-se-á ter um cuidado adicional ao definir o mecanismo de comunicação, uma vez que podem existir restrições no que toca à largura de banda disponível e a possíveis quotas de tráfego.

Código:	REQ_0201
Requisito:	A partir do dispositivo móvel será possível visualizar as fontes de evento subscritas.
Prioridade:	Essencial
Descrição:	Sub-requisitos mencionados em REQ_0800.

Código:	REQ_0202
Requisito:	A partir do dispositivo móvel será possível adicionar novas fontes de eventos predefinidas.
Prioridade:	Essencial
Descrição:	Sub-requisitos mencionados no REQ_0400.

Código:	REQ_0203
Requisito:	A partir do dispositivo móvel será possível definir alertas.
Prioridade:	Essencial
Descrição:	Sub-requisitos mencionados no REQ_0900.

Código:	REQ_0204
Requisito:	A comunicação com o dispositivo móvel deverá ser feita usando mecanismos adequados.
Prioridade:	Essencial
Descrição:	Dever-se-á ter um cuidado adicional ao definir o mecanismo de comunicação, uma vez que podem existir restrições no que toca à largura de banda disponível e a possíveis quotas de tráfego.

Código:	REQ_0300
Requisito:	O sistema deverá suportar diversos utilizadores através de um mecanismo de autenticação e autorização.
Prioridade:	Essencial
Descrição:	Quer a interface *web* quer a móvel devem estar sujeitas a autenticação e autorização.

E.1.2 Fontes de Eventos Predefinidas

Código:	REQ_0400
Requisito:	O utilizador deverá ter a oportunidade de subscrever fontes de eventos predefinidas.

Prioridade:	Essencial
Descrição:	O sistema deverá oferecer à partida um leque de fontes de interesse geral para os utilizadores. É essencial que as fontes de eventos predefinidas utilizem a interface referida anteriormente, não sendo programadas de forma *hardcoded* de forma a permitir futuras reutilizações.

Código:	REQ_0401
Requisito:	Deverá existir uma fonte de eventos predefinida para a bolsa de valores.
Prioridade:	Essencial
Descrição:	Deverá pedir como valores de entrada o índice e o mercado a que se refere.

Código:	REQ_0402
Requisito:	Deverá existir uma fonte de eventos predefinida para a meteorologia.
Prioridade:	Essencial
Descrição:	Deverá pedir como valor de entrada a localidade desejada. É desejável que o sistema faça uso da localização geográfica associada ao IP do cliente de forma a sugerir a localização.

Código:	REQ_0403
Requisito:	Deverá existir uma fonte de eventos predefinida para resultados desportivos.
Prioridade:	Desejável
Descrição:	Deverá pedir como valores de entrada a informação necessária para a identificação de uma equipa desportiva ou de uma partida em concreto.

Código:	REQ_0404
Requisito:	Deverão existir fontes de eventos predefinidas para o trânsito, tráfego aéreo e leilões.
Prioridade:	Desejável
Descrição:	Sem descrição.

Código:	REQ_0405
Requisito:	Deverão existir fontes de eventos predefinidas para *feeds* RSS e *Atom*.
Prioridade:	Desejável
Descrição:	Deverá pedir como valor de entrada a localização do *feed*.

Código:	REQ_0406
Requisito:	Deverão existir fontes de eventos predefinidas para XML, JSON, *iCal* e *Web Services* genéricos.
Prioridade:	Desejável
Descrição:	Deverá pedir como valor de entrada a localização do serviço.

E.1.3 Fontes de Eventos Personalizadas

Código:	REQ_0500
Requisito:	O utilizador deverá ter a oportunidade de criar fontes de eventos personalizadas e poder subscrevê-las.
Prioridade:	Essencial
Descrição:	O utilizador não deverá estar sujeito ao leque de fontes predefinidas, tendo a possibilidade de criar novas fontes e subscrevê-las. Estas fontes de eventos deverão ser criadas a partir de páginas *web* que apresentem um certo nível de informação estruturada. O utilizador poderá marcar a fonte de eventos como privada caso o deseje. Após a criação de uma fonte de eventos, e caso esta seja pública, deverá ficar disponível para subscrição por qualquer utilizador.

Código:	REQ_0501
Requisito:	O utilizador deverá ter a oportunidade de criar fontes de eventos personalizadas a partir de páginas *web* que necessitem de um conjunto metódico de passos para chegar a elas.
Prioridade:	Desejável
Descrição:	O sistema deverá guardar, com autorização do utilizador, os passos que ele realizou até chegar à página onde a informação se encontra. Como passos entende-se, por exemplo, autenticação, submissão de formulários, selecção de opções e seguir hiperligações.

Código:	REQ_0502
Requisito:	O utilizador deverá ter a oportunidade de identificar variáveis de entrada, caso a página o permita.
Prioridade:	Desejável
Descrição:	O sistema deverá guardar a lista de variáveis de entrada de forma ao alterar os seus valores os resultados obtidos sejam alterados. Por exemplo, ao efectuar uma pesquisa num qualquer motor e recolher os seus resultados, será possível identificar e tornar variável as palavras chave da pesquisa. Isto permitirá que qualquer utilizador possa usufruir desta fonte de informação da forma mais genérica possível.

Código:	REQ_0503
Requisito:	O utilizador deverá poder seleccionar, dentro de uma determinada página *web*, pedaços soltos de informação.
Prioridade:	Essencial
Descrição:	O utilizador deverá ter a oportunidade de seleccionar pedaços de informação. Estes podem ser elementos HTML (por exemplo hiperligações, blocos de divisão e células de uma tabela) ou, em última instância, texto livre.

Código:	REQ_0504

Requisito:	O utilizador deverá poder seleccionar, dentro de uma determinada página *web*, uma tabela HTML.
Prioridade:	Desejável
Descrição:	Após a selecção da tabela desejada, o utilizador deverá poder identificar os títulos das linhas e/ou colunas. Para além disso deverá ter a hipótese de identificar quais as linhas e colunas em que está interessado.

Código:	REQ_0505
Requisito:	O utilizador deverá poder seleccionar, dentro de uma determinada página *web*, valores identificados por estruturas HTML repetitivas.
Prioridade:	Desejável
Descrição:	O utilizador deverá ter a oportunidade de seleccionar pedaços de informação visivelmente repetitivos. Por exemplo, o utilizador poderá dirigir-se a uma página com informação noticiosa e seleccionar o título e descrição de uma notícia. Esta informação deverá ser suficiente para que o sistema detecte a existência, nessa mesma página, de outros elementos que sigam essa mesma estrutura e formatação.

Código:	REQ_0506
Requisito:	Caso a estrutura de uma página, que o utilizador tenha outrora definido como fonte de informação, se altere o sistema deverá adaptar-se de forma a conseguir extrair a informação correcta.
Prioridade:	Desejável
Descrição:	O sistema deverá implementar um mecanismo de comparação, entre a página original e a actual, de forma a identificar as alterações e adaptar a sua forma de extrair informação.

E.1.4 Tipagem de Dados

Código:	REQ_0600
Requisito:	O sistema, ao utilizador definir uma fonte de eventos personalizada, deverá identificar automaticamente os tipos de dados de cada pedaço de informação seleccionada.
Prioridade:	Essencial
Descrição:	O sistema deverá identificar tipos de dados simples como por exemplo números inteiros, números decimais, cadeias de caracteres, datas, horas e hiperligações.

Código:	REQ_0601
Requisito:	O utilizador deverá poder alterar os tipos de dados seleccionados automaticamente pelo sistema.
Prioridade:	Essencial

Descrição:	O utilizador deverá sempre poder redefinir os tipos de dados, seja isso da sua preferência ou o sistema tenha falhado na identificação.

Código:	REQ_0602
Requisito:	Os valores que obedecem a unidades deverão ser automaticamente reconhecidos e deverá ser dada a opção ao utilizador de seleccionar unidades equivalentes.
Prioridade:	Desejado
Descrição:	Por exemplo, caso o utilizador identifique um valor que corresponde a uma quantia monetária expressa em Euros, o sistema deverá sugerir unidades equivalentes (a título exemplificativo, dólar americano, libra e franco Suíço). O mesmo acontecerá, por exemplo, para unidades de medida, temperatura, área, volume, pressão, tempo e velocidade.

E.1.5 Operações sob Fontes de Eventos

Código:	REQ_0700
Requisito:	As fontes de eventos deverão ser catalogadas segundo um espaço de nomes hierárquico.
Prioridade:	Essencial
Descrição:	Por exemplo, retomando o exemplo da bolsa de valores, deverá existir o espaço de nomes base, `STOCK`, para todas as fontes relacionadas com a bolsa. Dentro dele estarão definidos os mercados, `STOCK.NASDAQ` ou `STOCK.PSI20`. Por fim, dentro de cada mercado encontrar-se-ão os índices, `STOCK.NASDAQ.GOOG` e `STOCK.NASDAQ.AAPL`.

Código:	REQ_0701
Requisito:	O utilizador deverá ter a oportunidade de etiquetar fontes de eventos com algumas palavras de forma a facilitar a procura.
Prioridade:	Essencial
Descrição:	Sem descrição.

Código:	REQ_0702
Requisito:	O utilizador poderá pesquisar por fontes de eventos já definidas no sistema, quer sejam predefinidas quer tenham sido criadas por outros utilizadores.
Prioridade:	Essencial
Descrição:	A procura incidirá não só no nomes das fontes como também nas etiquetas a ela associadas.

Código:	REQ_0703
Requisito:	A informação histórica de cada fonte de eventos deverá ser armazenada e poderá ser alvo de pesquisas e consultas.
Prioridade:	Essencial

Descrição:	Um utilizador poderá visualizar a informação histórica, desde o momento em que subscreveu a fonte de eventos.

E.1.6 Visualização de Eventos

Código:	REQ_0800
Requisito:	O utilizador deverá poder escolher como visualizar a informação, para cada fonte de eventos em particular.
Prioridade:	Essencial
Descrição:	Uma vez que os campos dos eventos são tipados à partida, um utilizador poderá definir, com algumas limitações, a forma mais adequada ao visionamento das suas fontes.

Código:	REQ_0801
Requisito:	O utilizador poderá filtrar e ordenar os eventos.
Prioridade:	Desejável
Descrição:	O utilizador terá a possibilidade de filtrar eventos através de uma condição (utilizando operadores como igual, diferente, maior, menor, maior ou igual, menor ou igual, contém e não contém) e ordenar os eventos através dos seus campos.

Código:	REQ_0802
Requisito:	O utilizador poderá unir eventos provenientes de fontes distintas.
Prioridade:	Desejável
Descrição:	Isto permite ao utilizador visualizar informação relativa a diversas fontes de eventos no mesmo ponto e, dependendo do modo de visualização, comparar os seus valores.

Código:	REQ_0803
Requisito:	O utilizador poderá aplicar funções aos eventos.
Prioridade:	Desejável
Descrição:	O utilizador terá a possibilidade de efectuar contagens, calcular máximos, mínimos e médias e efectuar simples operações aritméticas.

Código:	REQ_0804
Requisito:	O utilizador poderá visualizar a informação de forma textual e/ou por meio de tabelas.
Prioridade:	Essencial
Descrição:	Sem descrição.

Código:	REQ_0805
Requisito:	O utilizador poderá visualizar a informação recorrendo a gráficos de pontos, linhas, áreas, barras, e circulares.
Prioridade:	Essencial

Descrição:	Estes gráficos deverão suportar várias entradas de dados, sejam estas da mesma fonte de eventos ou não.

Código:	REQ_0806
Requisito:	O utilizador poderá visualizar a informação recorrendo a mapas, *gauges*, *tag clouds*, *network diagrams*, *scatterplots* e *treemaps*.
Prioridade:	Desejável
Descrição:	Estes gráficos deverão suportar várias entradas de dados, sejam estas da mesma fonte de eventos ou não.

Código:	REQ_0807
Requisito:	As formas de visualização definidas por um utilizador deverão estar disponíveis para que qualquer outro utilizador possa utilizá-las.
Prioridade:	Desejável
Descrição:	Sem descrição.

E.1.7 Definição de Alertas

Código:	REQ_0900
Requisito:	O utilizador deverá poder definir alertas, mediante condições, sobre os eventos.
Prioridade:	Desejável
Descrição:	Uma vez que os campos dos eventos são tipados à partida, um utilizador poderá construir regras. Quando essas regras se verificarem, um alerta será gerado. Nas condições, a título exemplificativo, deverão ser usados operadores como igual, diferente, maior, menor, maior ou igual, menor ou igual, contém e não contém. O alerta poderá ser despoletado através da combinação de regras, mediante os operadores de intersecção, disjunção e negação.

Código:	REQ_0901
Requisito:	O utilizador deverá poder definir como destino do alerta o *dashboard* e/ou o e-mail.
Prioridade:	Essencial
Descrição:	Sem descrição.

Código:	REQ_0902
Requisito:	O utilizador deverá poder definir como destino do alerta um telemóvel, através do serviço SMS.
Prioridade:	Desejável
Descrição:	Sem descrição.

E.2 Requisitos Não-Funcionais

Código:	REQ_1000
Requisito:	Os eventos deverão ser visíveis em tempo real.
Prioridade:	Essencial
Descrição:	Um evento contém informação sensível e que apenas faz sentido se for observado dentro de um intervalo de tempo. Este intervalo de tempo varia consoante o tipo de eventos. Cada fonte de eventos possui uma taxa de actualização e os valores deverão ser actualizados dentro desse intervalo.

Código:	REQ_1001
Requisito:	Os alertas deverão ser emitidos em tempo real.
Prioridade:	Essencial
Descrição:	Da mesma forma que o requisito anterior, o alerta deverá ser emitido dentro do intervalo de tempo entre actualizações, logo após a condição de alerta ocorrer.

Código:	REQ_1002
Requisito:	A informação mais antiga deverá ser sumariada.
Prioridade:	Essencial
Descrição:	Deverão ser definidos critérios que permitam agregar informação mais antiga, suportando vários níveis. Por exemplo, será possível definir que a informação com mais de três meses seja agregada diariamente, com mais de seis meses agregada semanalmente e com mais de um ano agregada mensalmente.

E.2.1 Atributos de qualidade

Código:	REQ_1100
Requisito:	O sistema deverá ser escalável (*scalability*).
Prioridade:	Essencial
Descrição:	Deverá ser desenhado de forma a conseguir lidar com *workloads* elevados, permitindo de forma elegante adicionar e retirar recursos ao sistema. Nomeadamente, o sistema deverá lidar com dezenas de utilizados e dezenas de fontes, sendo estas fontes potencialmente actualizadas múltiplas vezes por minuto.

Código:	REQ_1101
Requisito:	O sistema deverá ter uma disponibilidade elevada (*high availability*).
Prioridade:	Essencial

Descrição:	O sistema deverá implementar mecanismos que permitam exerçer as suas funções durante um período elevado de tempo. Não é estipulada a percentagem mínima de disponibilidade, no entanto esta deve ser o mais elevada possível.

Código:	REQ_1102
Requisito:	O sistema deverá ser confiável (*reliability*).
Prioridade:	Essencial
Descrição:	O sistema deverá ser resistente a falhas e apresentar ao utilizador, sempre que possível, resultados correctos e actualizados. Não é estipulada a percentagem mínima de confiabilidade, no entanto esta deve ser o mais elevada possível.

Código:	REQ_1103
Requisito:	O sistema deverá ter em conta questões de segurança (*security*).
Prioridade:	Essencial
Descrição:	O sistema deverá ser desenhado com vista a minimizar falhas de segurança que possam vir a comprometer a integridade do sistema e a informação confidencial dos utilizadores.

Código:	REQ_1104
Requisito:	O sistema deverá ser implementado tendo em conta a sua futura manutenção (*maintainability*).
Prioridade:	Essencial
Descrição:	O sistema deverá permitir a adição de novas funcionalidades e efectuar a correcção de problemas.

Código:	REQ_1105
Requisito:	O sistema deverá executar num vasto leque de plataformas (*portability*).
Prioridade:	Essencial
Descrição:	O sistema deverá usar mecanismos que o façam abstrair das interfaces de baixo nível, permitindo correr sob várias plataformas distintas.