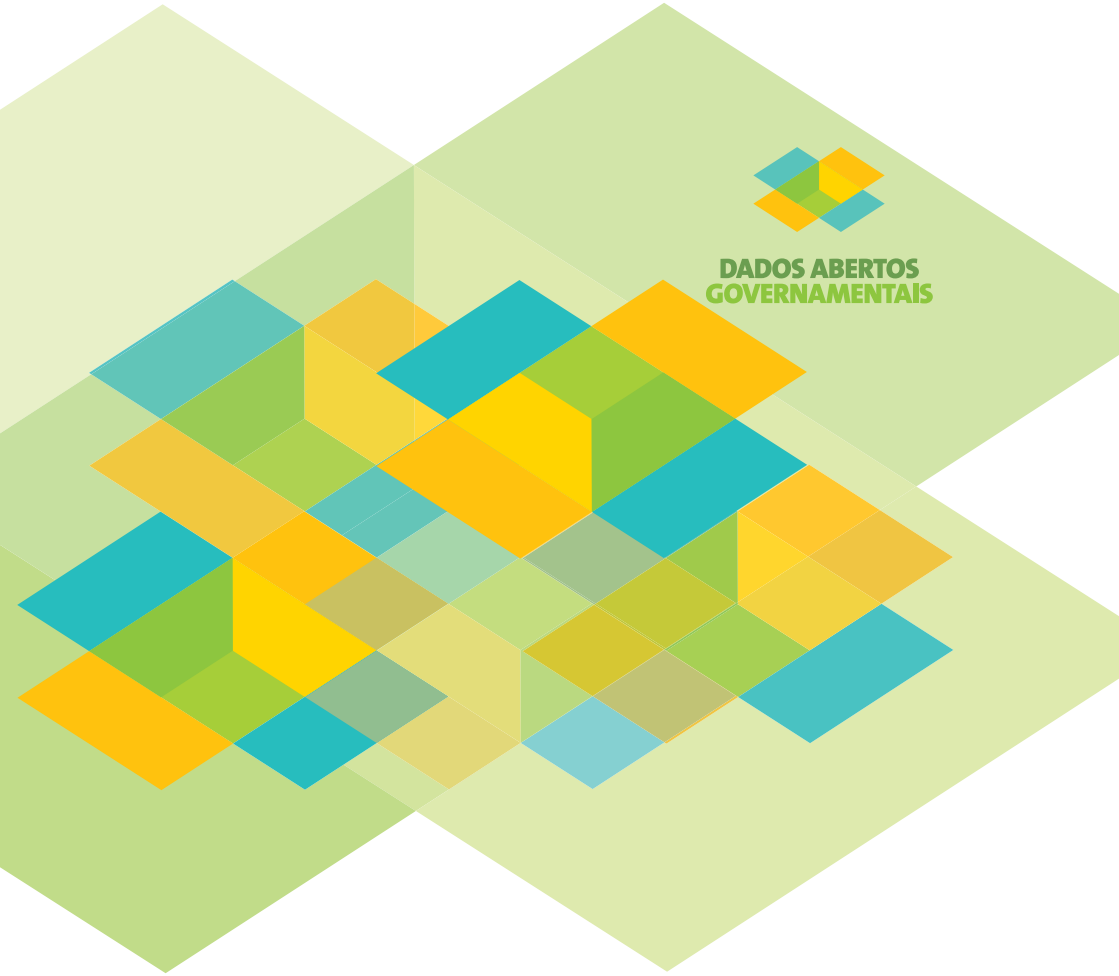
manual dos

dados abertos: desenvolvedores



W3C WORLD WIDE WEB consortium Escritório Brasil

cgi.br Comitê Gestor da Internet no Brasil

nic.br Núcleo de Informação e Coordenação do Ponto BR

laboratório brasileiro de cultura digital

Cartilha para desenvolvedores Transparência Hacker

Este manual foi projetado para desenvolvedores de software que queiram trabalhar com dados abertos. Aqui se discute por que abrir dados, o que são dados abertos, como publicar dados em formato aberto e como criar aplicativos reutilizando dados de governo em formato aberto ou não.

Este manual é uma construção colaborativa, que teve como base uma série de artigos publicados pela Comunidade São Paulo Perl Mongers sob o título *Equinócio de março* (*hack de dados públicos*). O W3C Brasil fez a edição e a reorganização dos textos para o formato de um manual e acrescentou contribuições de comunidades de outras linguagens de programação *web* (Ruby, Lua, Phyton e o cms em PHP Drupal).

Encartado neste manual há uma cartilha desenhada para desenvolvedores, elaborada pela Comunidade Transparência Hacker. É uma publicação resultante do acordo de cooperação técnico-científica entre o Laboratório Brasileiro de Cultura Digital e o Núcleo de Informação e Coordenação do Ponto BR (NIC.br).

Créditos e licenças

Os textos da Comunidade São Paulo Perl Mongers que compõem e deram origem a esse manual estão disponíveis em http://sao-paulo.pm.org/principal

Pela revisão dos artigos, conversas e ideias:

- Thiago Rondon (thiago.rondon@gmail.com)
- Alexei "Russo" Znamensky
- Breno G. de Oliveira (garu@cpan.org)

Comunidade Perl

- Thiago Rondon (thiago.rondon@gmail.com)
- Stanislaw Pusep (stas@sysd.org)
- Renato Cron (http://renatocron.com)
- Hernan Lopes (hernanlopes@mail.com)

Comunidade PythonBrasil

- Flavio Codeço Filho (fccoelho@gmail.com)
- Jonh Edson Ribeiro de Carvalho (jonhedson@gmail.com)
- Giuseppe Romagnoli (giuseppe.romagnoli@gmail.com)
- Luiz Guilherme F. Aldabalde (lg.aldabalde@gmail.com)

Comunidade Ruby do Brasil

- Fabio Akita (fabioakita@gmail.com)

Comunidade LUA

- Gabriel Duarte (confusosk8@gmail.com)

Drupal e Web Semântica

- César Brod (cesar@brod.com.br)
- Joice Käfer (joice@brod.com.br)

Comunidade Transparência Hacker

- Pedro Markun (pedro no esfera.mobi)
- Daniela Silva (daniela no esfera.mobi)
- Diego Casaes (Diego no esfera.mobi)

Outras publicações do W3C da série "Dados Abertos Governamentais"

- Melhorando o acesso ao governo com o melhor uso da web, 2009
- Manual dos dados abertos: governo, 2011

Creative Commons 3.0 – Atribuição
Você pode usar, copiar, modificar, remixar, imprimir, distribuir e traduzir esse material para outras línguas, citando a fonte original.
http://creativecommons.org/licenses/by/3.0/legalcode

W3C Escritório Brasil

Laboratório Brasileiro de Cultura Digital

manual dos

dados abertos: desenvolvedores

Comitê Gestor da Internet no Brasil

São Paulo

2011

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Manual dos dados abertos : desenvolvedores / [cooperação técnica científica entre Laboratório Brasileiro de Cultura Digital e o Núcleo de Informação e Coordenação do Ponto BR (NIC.br)]. -- São Paulo : Comitê Gestor da Internet no Brasil, 2011.

Vários colaboradores.
"Acima do título: W3C Escritório Brasil"
ISBN 978-85-60062-47-8

1. Acesso a dados públicos 2. Administração pública - Recursos de informação - Administração 3. Banco de dados 4. Informação eletrônica governamental 5. Internet (Rede de computadores) 6. Internet na administração pública 7. Linguagem de programação para computadores 8. Serviços de informação on-line 9. Tecnologia 10. Websites - Desenvolvimento.

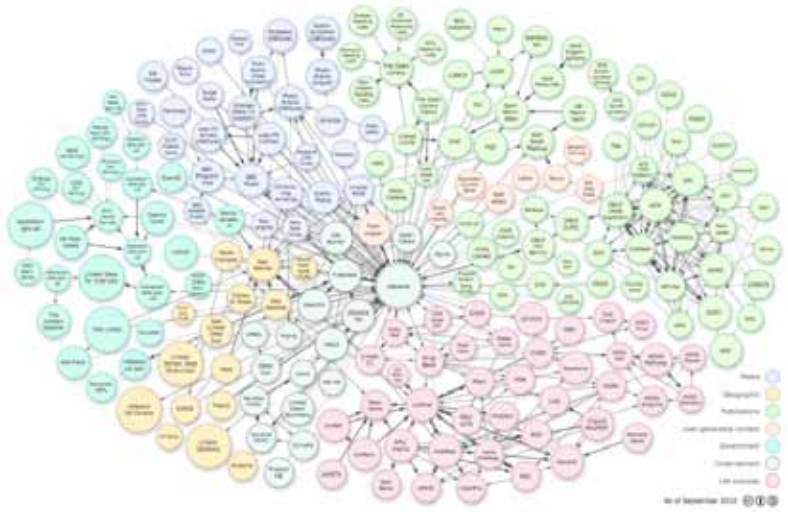
11-09379 CDD-352.380285

Índice para catálogo sistemático:

1. Dados abertos governamentais : Serviços de informação eletrônica : Administração pública 352.380285
2. Dados governamentais abertos : Serviços de informação eletrônica : Administração pública 352.380285

sumário

introdução à publicação de dados abertos



Fonte: Wikipedia.

apresentação

A Internet é uma reunião de pessoas e computadores em escala mundial, dotada de tecnologias que provêm facilidade e grande utilização de documentos por meio da World Wide Web, ou WWW, um sistema de hipermídia que interliga documentos.

Essas tecnologias podem propiciar ao governo uma maneira inovadora de expor os dados já públicos, de forma que muitos consigam usá-los de fato. Sabe-se que há fatores políticos, legais e de cultura que limitam a exposição de dados, porém, existindo esforço para tal implementação, o retorno será positivo. Jornalistas, analistas políticos, desenvolvedores de *software*, organizações, empresários, advogados, médicos e o público em geral terão recursos para acessar e entender todo o repositório de dados gerado pelo governo.

No entanto, apenas colocar documentos na *web* não significa transparência pública. É preciso que as informações estejam disponíveis de modo que humanos e máquinas consigam interpretá-las de forma ágil. Este modo é genericamente designado como "dados abertos". Assim, pode-se esperar colaboração e participação para que exista inovação na criação de aplicativos, como por exemplo para criar distintas visualizações dos dados ou *mash-ups*.

A Constituição brasileira prevê a publicidade de uma grande quantidade de informações de interesse da sociedade, a qual atualmente, devido ao fato de não serem dados abertos, não explora todo o potencial de documentos na *web*.

A principal proposta deste documento é apontar, para desenvolvedores *web*, caminhos que facilitem a publicação dos dados em formato aberto, assim como sua re-utilização por e para a sociedade.

Este manual é uma construção colaborativa e que teve como base uma série de artigos publicados pela Comunidade São Paulo Perl Mongers sob o título "Equinócio de Março (Hack de Dados Públicos)[1]. É o segundo volume da série "Manual de Dados Abertos", publicada pelo W3C. O primeiro volume, "Manual de Dados Abertos – Governo", teve como público-alvo o gestor público, oferecendo uma introdução aos dados abertos e os benefícios de sua utilização. O leitor que necessitar uma abordagem mais básica sobre o tema deve ler este manual, disponível em http://www.w3c.br/divulgacao/pdf/manual_dados-abertos-governo.pdf.

O manual "Manual de Dados Abertos – Desenvolvedor" tem como público-alvo toda a comunidade de desenvolvedores *web* e oferece uma visão mais técnica sobre como publicar e reutilizar dados abertos. Para o leitor que vai se debruçar apenas sobre este volume, há uma brevíssima introdução sobre dados abertos, seus princípios e conceitos. Nos demais capítulos, apresenta-se como expor os dados em formato aberto, como extrair dados públicos e como reutilizar esses dados.

[1] Página *web* da Comunidade São Paulo Perl Mongers: http://sao-paulo.pm.org/principal.

O leitor vai encontrar encartado neste manual uma cartilha de bolso, desenhada especialmente para apresentar, de modo sucinto e para leitura rápida, referências úteis aos desenvolvedores. A cartilha foi elaborada pela Comunidade Transparência Hacker, sendo a segunda publicação resultante do acordo de cooperação técnico-científica entre o Laboratório Brasileiro de Cultura Digital e o Núcleo de Informação e Coordenação do Ponto BR (NIC.br).

o que diz a constituição

constituição da república federativa do brasil

> Art. 37. A administração pública direta e indireta de qualquer dos Poderes da União, dos Estados, do Distrito Federal e dos Municípios obedecerá aos princípios de legalidade, impessoalidade, moralidade, publicidade e eficiência e, também, ao seguinte: (Redação dada pela Emenda Constitucional nº 19, de 1998) [...]
>
> XXII – as administrações tributárias da União, dos Estados, do Distrito Federal e dos Municípios, atividades essenciais ao funcionamento do Estado, exercidas por servidores de carreiras específicas, terão recursos prioritários para a realização de suas atividades e atuarão de forma integrada, inclusive com o compartilhamento de cadastros e de informações fiscais, na forma da lei ou convênio. (Incluído pela Emenda Constitucional nº 42, de 19.12.2003)
>
> § 1º - A publicidade dos atos, programas, obras, serviços e campanhas dos órgãos públicos deverá ter caráter educativo, informativo

ou de orientação social, dela não podendo constar nomes, símbolos ou imagens que caracterizem promoção pessoal de autoridades ou servidores públicos.

§ 2º - A não observância do disposto nos incisos II e III implicará a nulidade do ato e a punição da autoridade responsável, nos termos da lei.

§ 3º A lei disciplinará as formas de participação do usuário na administração pública direta e indireta, regulando especialmente: (Redação dada pela Emenda Constitucional nº 19, de 1998)

I – as reclamações relativas à prestação dos serviços públicos em geral, asseguradas a manutenção de serviços de atendimento ao usuário e a avaliação periódica, externa e interna, da qualidade dos serviços; (Incluído pela Emenda Constitucional nº 19, de 1998)

II – o acesso dos usuários a registros administrativos e a informações sobre atos de governo, observado o disposto no art. 5º, X e XXXIII; (Incluído pela Emenda Constitucional nº 19, de 1998)

III – a disciplina da representação contra o exercício negligente ou abusivo de cargo, emprego ou função na administração pública. (Incluído pela Emenda Constitucional nº 19, de 1998)

§ 4º Os atos de improbidade administrativa importarão a suspensão dos direitos políticos, a perda da função pública, a indisponibilidade dos bens e o ressarcimento ao erário, na forma e gradação previstas em lei, sem prejuízo da ação penal cabível.

§ 5º A lei estabelecerá os prazos de prescrição para ilícitos praticados por qualquer agente, servidor ou não, que causem prejuízos ao erário, ressalvadas as respectivas ações de ressarcimento.

§ 6º As pessoas jurídicas de direito público e as de direito privado prestadoras de serviços públicos responderão pelos danos que seus agentes, nessa qualidade, causarem a terceiros, assegurado o direito de regresso contra o responsável nos casos de dolo ou culpa.

§ 7º A lei disporá sobre os requisitos e as restrições ao ocupante de cargo ou emprego da administração direta e indireta que possibilite o acesso a informações privilegiadas. (Incluído pela Emenda Constitucional nº 19, de 1998)

§ 8º A autonomia gerencial, orçamentária e financeira dos órgãos e entidades da administração direta e indireta poderá ser ampliada mediante contrato, a ser firmado entre seus administradores e o poder público, que tenha por objeto a fixação de metas de desempenho para o órgão ou entidade, cabendo à lei dispor sobre: (Incluído pela Emenda Constitucional nº 19, de 1998)

portal da transparência, governo federal, mantido pela controladoria geral da união (cgu)

"Todo aquele que guarde, administre, gerencie, arrecade ou utilize bens e valores públicos tem o dever constitucional e moral de prestar contas dos recursos públicos. Essa prestação de contas consiste no envio, aos órgãos responsáveis, do conjunto de documentos e informações, obtidos direta ou indiretamente, que permitam avaliar a conformidade e o desempenho da gestão dos responsáveis por políticas públicas, bens, valores e serviços públicos federais. [...]

A CGU e o Ministério do Planejamento, Orçamento e Gestão instituíram, por meio da Portaria Interministerial nº 140, de 16 de março de 2006, a criação das Páginas de Transparência Pública dos órgãos e entidades da Administração Pública Federal. São site[s] que apresentam os dados relativos a execução orçamentária, licitações públicas, contratações, convênios e diárias e passagens da Presidência da República, dos ministérios e dos outros órgãos e entidades do Governo Federal.

A CGU também desenvolve o Programa Olho Vivo no Dinheiro Público, a fim de capacitar agentes públicos municipais em assuntos pertinentes à transparência da gestão, à responsabilização e à necessidade do cumprimento dos dispositivos legais.

A Controladoria ainda oferece regularmente o curso a distância "Controle Social e Cidadania". Os objetivos da capacitação, voltada para conselheiros e agentes públicos municipais, lideranças

locais, professores, estudantes e cidadãos em geral, são incentivar a atuação no controle social das ações de governo e promover a melhor aplicação dos recursos públicos. O curso está estruturado em três módulos: "A participação popular no Estado brasileiro"; "O controle das ações governamentais"; e "O encaminhamento de denúncias aos órgãos responsáveis".[2]

conceitos importantes

O conceito de dados abertos aplica-se a todo o conjunto de dados que podem ser publicados na *web*, e não apenas dados governamentais (para mais detalhes, ver o volume 1, "Manual de Dados Abertos – Governo").[3] Porém os dados de governo, pelo volume e relevância para a sociedade, é que têm sido o foco principal das iniciativas nessa área.

Apenas liberar os documentos não torna eficaz o processo de publicidade dos dados, pois manipular a enorme quantidade de documentos gerados pelo governo pode tornar bem difícil a tarefa.

O que se busca com dados abertos é maior participação da sociedade na vida pública:

- por meio de documentos em formatos de fácil manipulação por humanos.
- por meio de documentos em formatos de manipulação inteligente por máquinas.

[2] Disponível em http://www.portaldatransparencia.gov.br/faleConosco/perguntas-tema-participacao.asp.

[3] Página *web* do W3C Brasil: http://www.w3c.br/divulgacao/pdf/manual_dados-abertos-governo.pdf.

Se houver boa utilização da tecnologia existente e iniciativa da sociedade, os dados governamentais poderão ser cada vez mais benéficos para todos. Sua reutilização poderá garantir maior:

- **Transparência**: provendo melhor acesso aos dados.
- **Participação**: facilitando a educação pública, a democratização do conhecimento e a inovação.
- **Colaboração**: proporcionando contínua realimentação da soiedade e disseminação colaborativa do conhecimento.

princípios dos dados abertos

O conceito de *dados abertos* não é novidade, sendo sempre utilizado em cenários distintos, e requer que um determinado dado esteja disponível para todos, sem restrições. Os princípios listados abaixo caracterizam os dados abertos e expõem melhor o seu significado.

acesso livre

Qualquer pessoa na rede deverá ter acesso aos documentos, sem discriminação de grupos ou pessoa. Algumas tecnologias podem excluir, por exemplo a utilização de um sistema de *captcha* baseado apenas em imagem, em que deficientes visuais seriam impedidos de buscar esses dados.

separar os dados ao máximo

Separar os dados ao máximo, em estruturas distintas, ou seja, deixar as informações armazenadas em formatos da maior uni-

dade possível, facilita o entendimento e a análise da informação, além de evitar resumos.

Em um segundo plano, uma opção interessante pode ser misturar informações distintas, facilitando também o entendimento de determinado dado em um contexto diferente.

responsabilidade

Deve-se promover a responsabilidade dentro dos vários departamentos do governo, para que estes possam publicar dados com qualidade e da maneira mais rápida, de forma segura para o governo e a sociedade.

rápida integração

Deve-se oferecer ferramentas e especificações técnicas para que as integrações sejam realizadas de forma rápida e constante

compartilhamento de boas práticas

Para que exista uma rápida integração, é necessário que os órgãos compartilhem informações e experiências relacionadas a boas práticas.

formatos de arquivos

Formatos de arquivos proprietários podem criar dependência tecnológica para o uso das informações, e isso gerará restrições ao acesso dos dados.

Dessa forma, os dados devem estar estruturados e organizados para facilitar sua manipulação por *softwares* diversos. Por exemplo, alguns documentos são oferecidos pelo governo em formato PDF, o qual não oferece nenhuma estrutura que permita que o documento seja analisado por um *software*.

serialização de dados

Não se deve disponibilizar apenas um único formato aberto de arquivos, pois isso também prejudicaria a utilização por um grupo de pessoas (por falta de conhecimento), e em outros casos faltaria estruturação para manipular os arquivos.

ontologias

Na filosofia, ontologia é o estudo da existência ou do ser enquanto ser, ou seja, a maneira de compreender as identidades e grupos de identidades. Na ciência da computação, é um modelo de dados que representa um conjunto de conceitos sob um domínio e seus relacionamentos, ou, mais formalmente, especifica uma conceitualização dele.

owl (*web ontology language*)

A OWL é uma linguagem para definir e instanciar modelos de dados para informações na *web*. É utilizada em aplicações que precisam processar informações contidas em documentos *web*, não sendo apresentada em formato legível apenas por humanos. É uma recomendação do W3C.

A OWL oferece recursos para elaboração de vocabulários que permitem que a *web* seja mais semântica, oferecendo significados para serem utilizados em *softwares*. Com ela, facilita-se a interpretação de dados por máquinas, utilizando estruturas como XML, RDF e RDFSs, que serão detalhadas no decorrer deste documento.

dados conectados (*linked data*)

Além de oferecer arquivos para serem visualizados ou para *download*, estes devem estar prontos para serem conectados (ou integrados) por meio da *web*. Isso é essencial em muitos cenários.

No documento "Design Issues: Linked Data",[4] Tim Berners-Lee explica como os dados podem ser interligados por meio da *web*. Em resumo:

- Utilizar URI (*Uniform Resource Identifier*) para identificar "coisas" (documentos, arquivos, imagens, serviços, etc.) de maneira única na *web*;
- Utilizar HTTP URI (http://...) para que as "coisas" possam ser localizadas por pessoas ou aplicativos;
- Fornecer informações úteis sobre "coisas", em formatos como RDF/XML;
- Incluir *links* para outros dados como forma de referência, para melhorar a visualização dos dados expostos.

[4] Página *web* do W3C: http://www.w3.org/DesignIssues/LinkedData.html.

metadados

Metadados são comumente definidos como "dados sobre dados". São informações que caracterizam os dados e informam sobre sua natureza, propósito, formato, autoria, etc.

vocabulários

Para representar dados em RDF, é necessário definir vocabulários (muitas vezes também chamados de metadados, ou até mesmo de ontologias). Existem alguns tipos de vocabulários já definidos, que podem ser utilizados para expressar os dados em formato aberto.

FOAF

FOAF é o acrônimo para "Friend of Friend" [Amigo do amigo]. Trata-se de uma ontologia, interpretada por máquina, para descrever dados de pessoas, suas atividades, o relacionamento com outras pessoas e objetos, etc.

Existem alguns módulos para tratar deste tipo de *dataset* no CPAN, XML::FOAF, Gedcom::FOAF e XML::FOAF::Person.

É possível encontrar mais informações sobre esta especificação em http://rdfweb.org/foaf/.

GeoNames

É um tipo de banco de dados que disponibiliza metadados geográficos, e pode ser acessado por vários tipos de *web services*. O GeoNames contém um conjunto de vocabulários para adicionar

a semântica necessária para a distribuição de dados, como RDF/XML, DBpedia, etc.

Existe uma API em Perl para isso, que é Geo::GeoNames, utilizando o *web service* http://ws.geonames.org/.

DBpedia

É uma interface orientada a dados que extrai informações estruturadas criadas no Wikipedia, permitindo-se efetuar consultas sofisticadas e associar documentos e outros conjuntos de dados (*datasets*) para disponibilizá-los na *web*.

Ainda não há um módulo no CPAN para este projeto. Quem sabe o leitor não possa escrever o primeiro?

schema

No contexto de dados abertos, *schema* expressa vocabulários que são compartilhados e orienta como computadores devem lidar com eles.

repositório visual

É recomendada a criação de um repositório visual que esteja hospedado em um órgão de governo, oferecendo um mapa dos recursos disponíveis para sua exploração. O repositório pode utilizar a colaboração dos demais órgãos do governo, aproveitando a plataforma para compartilhar e divulgar dados. Dessa maneira, os dados abertos estariam todos em um mesmo local

para a sociedade, porém seriam mantidos dentro do órgão em que foram gerados.

Assim, seria mais fácil para qualquer pessoa encontrar os dados e identificar locais com problemas. Dois ótimos exemplos são o repositório norte-americano (http://data.gov) e o inglês (vide http://data.gov.uk).

perguntas mais frequentes

de onde devem vir os dados?

Os dados devem vir de seus "donos" ou guardiões, sejam eles organizações privadas ou públicas. Se os dados são de governo, a sua publicidade em muitos casos é estabelecida por lei, como no caso do Brasil com a obrigatoriedade de publicização das receitas e despesas para todos os órgãos públicos.

como deve ser a alimentação de dados?

Os dados, além de disponibilizados em banco de dados internos, que só o governo pode visualizar, também podem ser oferecidos em formatos abertos na Internet, para que exista reutilização das informações de maneira simples pela sociedade.

quais são os formatos?

Assim como os dados, os formatos devem ser abertos, para que não exista dependência tecnológica no acesso aos dados. Há sempre uma alternativa a qualquer formato proprietário. Um exemplo muito comum de uso de formato proprietário é oferecer dados em planilhas Microsoft Excel. Uma alternativa em formato aberto seria o CSV.

CSV

Não há uma especificação formal deste padrão, porém o formato de arquivo CSV armazena dados tabelados em um arquivo de texto cujos valores são separados por um delimitador comum.

Usar um arquivo CSV é muito simples, além de ele ser suportado por quase todos os editores de texto, planilhas eletrônicas e bancos de dados disponíveis no mercado.

Para se distribuir dados governamentais, a sugestão é utilizar, na primeira linha (cabeçalho), o nome das colunas dos dados que estarão dispostos no arquivo. Um exemplo:

```
EMPRESA,CNPJ,TELEFONE,ENDEREÇO,CEP
Empresa de exemplo, 0000-0000-0000/0000-1, 1234-1234, Rua exemplo, 42420-123
Empresa de exemplo 2, 0000-0000-0000/0000-2, 1234-1234, Rua exemplo, 42420-123
```

rdf

Trata-se de uma linguagem para representação de informações na Internet. São arquivos baseados em metadados, com o objetivo de criar um modelo de documento no formato XML com semântica.

Com os dados estruturados, a utilização de linguagens para consulta pode auxiliar muito quem quer pesquisar ou referenciar dados contidos em documentos que estejam na Internet. Uma dessas linguagens é a SPARQL.

Em alguns casos, há dificuldade técnica para implementação, pois os dados já estão formatados em um determinado padrão, ou em um banco de dados que não oferece uma maneira trivial de serializar os dados. Porém existe uma diversidade de tecnologias, disponíveis na própria Internet, para facilitar este trabalho. Um exemplo muito bom é o projeto D2R Server,[5] que trabalha como uma camada na frente de um banco de dados relacional, oferecendo dados em RDF e a capacidade de realizar consultas em SPARQL.

A importância de se ter arquivos em RDF é usufruir do conceito principal do *design* deste *framework*, baseado em URI e XML. Ou seja, documentos RDF podem ser de utilização simples e com uma organização de dados unificada.

Para mais informações sobre publicação de arquivos RDF, veja as referências bibliográficas ao final deste documento.

[5] D2R MAP é uma linguagem declarativa para expressar mapeamentos de um banco de dados relacional e ontologias OWL/RDF. O D2R *Processor* exporta dados de um banco de dados relacional para RDF.

existe um padrão para os dados?

Na Internet, existem ótimos padrões, como o RDF, que define uma estrutura baseada em URI (a localização dos arquivos na *web*) e em XML (como o documento será estruturado).

Além de oferecer os arquivos em formato RDF, é interessante que eles também sejam disponibilizados em outros formatos de fácil manipulação por humanos, como o CSV.

com qual frequência?

Informações públicas devem ser oferecidas conforme a lei estabelece. Não havendo regulamentação, a frequência com que os dados são atualizados oferece um bom parâmetro de definição.

como dividir os dados?

A divisão dos dados é um fator muito importante para sua reutilização de forma proveitosa. Ou seja, dividir as informações no máximo de formatos possíveis é significativo para facilitar o acesso a elas e sua interpretação.

conclusão

O governo já deu grandes passos, pois existem dispositivos legais regulando a publicidade da informações públicos, já havendo dados na *web*. Porém, atualmente, a sociedade enfrenta dificuldades em obter os dados em formatos adequados para serem analisados. Com um trabalho mais elaborado e organizado para disponibilizar dados, haverá benefícios para todos: o governo receberá sugestões e haverá uma participação mais eficiente da sociedade, a qual saberá mensurar melhor o trabalho das autoridades.

A melhor frase para expressar a motivação existente no movimento pelos dados abertos é a afirmação de Eric Raymond, em um dos ensaios do livro *Bazar e catedral* (A Lei de Linus): "Havendo olhos em quantidade suficiente, todos os erros são triviais".[6]

[6] RAYMOND, Eric Steven. *The Cathedral & the Bazaar*, apud O'Reilly (1999).

estruturando e expondo dados abertos

apresentação

Um dos maiores problemas encarados por um projeto de dados abertos está em definir o formato em que a informação deve ser disponibilizada pelo governo. A lei diz que os dados devem ser expostos à população, mas não especifica como. Por essa razão, muitas vezes os esforços de transparência, tão valorizados pela administração pública de nosso país, acabam sem a repercussão ou resultados desejados.

Neste artigo são apresentadas recomendações sobre a estruturação e a exposição de dados públicos, com base nos exemplos do sítio do Instituto de Segurança Pública (ISP) do Estado do Rio de Janeiro e no *dataset* FOAF. São oferecidos exemplos em diferentes *frameworks* de desenvolvimento *web* e distintas linguagens (PERL, Ruby, Phyton e Lua e formatos (JSOJN, YAML e XML).

como não fazer

Infelizmente, boa parte das informações é disponibilizada em bons formatos para impressão analógica, sendo de difícil análise e manuseio por *software*, além de, por vezes, serem expostos em padrões proprietários. Um relatório em PDF pode ser elegante quando impresso, mas isso força um analista a aceitar a informação como está, ou perder muito tempo para realizar referências cruzadas, análises comparativas e estatísticas, desperdiçando e invalidando os esforços de democratização e cidadania.

Embora existam técnicas de leitura e extração automática das informações em arquivos deste tipo – muitos dos quais discutidos no resto deste documento –, o ideal é que as informações sejam expostas de forma mais bem estruturada.

Um exemplo simples é o excelente sítio do Instituto de Segurança Pública – ISP do Estado do Rio de Janeiro (http://www.isp.rj.gov.br), que tornou públicas as informações acerca de incidências criminais no estado. Com isso, a Secretaria de Estado de Segurança (Seseg) espera "assegurar o máximo de transparência e o amplo acesso às informações relativas a notificações criminais".[7]

Ao carregar a página com os dados oficiais, tem-se acesso a informações mensais. O arquivo está em formato XLS, proprietário, mas ao menos soluções livres, como o OpenOffice, permitem a leitura de seu conteúdo estruturado. Quando foi aberto o arquivo de janeiro de 2010, encontrou-se uma coleção completa de dados: 44 planilhas individuais detalhando cada AISP (Área In-

[7] Disponível em http://www.isp.rj.gov.br/Conteudo.asp?ident=150.

tegrada de Segurança Pública). Os autores foram cuidadosos, incluindo planilhas de resumo com o índice estadual de ocorrências e dados condensados por RISP (Região Integrada de Segurança Pública).

No entanto, se alguém quiser saber, por exemplo, qual área possui a maior incidência de homicídios culposos, precisará analisar cada uma das 44 planilhas de AISP, procurando o índice em questão, e anotar o maior manualmente. A automatização é possível, mas depende da informação estar sempre disponível nas mesmas linhas/colunas em todos os documentos, ou com o mesmo rótulo.

Pode ser simples fazer isso manualmente uma vez, mas, se a intenção for obter essa informação mensalmente, ou realizar uma análise estatística de crescimento, ou conhecer a redução da criminalidade no estado ou região, a análise manual pode levar mais tempo do que o desejado para qualquer analista interessado.

A maioria dos portais de transparência, dos sítios municipais aos do governo federal, sofre deste problema. Portanto, se o leitor for responsável pela transparência de dados do poder público, é interessante que evite usar formatos proprietários como PDF, XLS, DOC, HTML etc. Ou melhor, não deve disponibilizar apenas nesses formatos.

Arquivos para *download* ainda são suficientes para quem quiser um dado pontual, resumo ou apenas uma listagem simples. O verdadeiro poder da cidadania, no entanto, está na facilidade de extração e análise da informação exposta, processo dificultado por formatos como os listados acima.

web services, um cenário

Uma alternativa para a disponibilização de dados públicos são os chamados *web services*. Trata-se de um método simples e efetivo de comunicação e interação entre aplicativos *web*. Quando aproveitados em portais de transparência, permitem acesso imediato a informações na forma de estruturas de dados mais bem definidas, facilitando a automatização dos processos de coleta e análise.

Existem diversos tipos de *web services*, sendo o REST um dos mais populares. Há muito material disponível sobre todos os tipos, da estrutura à implementação, mas podemos nos concentrar em mostrar como é simples a exibição de dados e estruturas complexas nos principais formatos para APIs (Application Programming Interface) para *web*.

Suponha que o Instituto de Segurança Pública do Rio queira implementar um *web service* para facilitar o acesso às informações sobre incidências criminais em seu sítio. Os responsáveis poderiam criar um serviço REST exposto com uma URI no formato: http://isp.rj.gov.br/api/<ANO>/<MES>/<No._AISP>/<TIPO_INDICADOR>.

Assim, para acessar a lista de registros de crimes violentos na região AISP7 em maio de 2009, bastaria acessar os dados a partir da URI http://isp.rj.gov.br/api/2009/05/AISP7/crimes-violentos. E daí seria possível importar o resultado para o *software* de análise.

estruturando dados – perl com json, yaml e xml

Em Mojolicious (um *framework* para desenvolvimento *web* em linguagem PERL),[8], para criar esse tipo de rota, basta escrever:

```
use Mojolicious::Lite;
get '/:ano/:mes/:aisp/:tipo' => sub {
...
};
app->start;
```

E, então, só faltaria colocar o aplicativo respondendo em http://isp.rj.gov.br/api.

Para os exemplos a seguir, supõe-se que os dados tenham sido obtidos de um banco de dados e estejam em uma estrutura de dados Perl como esta:

```
my $crimes_violentos = {
'Roubo de aparelho celular' => 30,
'Furto de veiculos'         => 72,
'Extorsao'                  => 6,
'Estelionato'               => 62,
};
```

[8] Página *web* do Mojolicious: http://www.mojolicious.org/.

json

O formato JSON é leve e simples. Sua estrutura é de fácil entendimento e utiliza convenções comuns em muitas linguagens de programação.

Passar a estrutura de dados acima para JSON é simples:

```
use JSON;
my $json = to_json( $crimes_violentos );
```

Agora, é possível "renderizar" a variável $json, lembrando-se de definir o *Content-Type* da página para "application/json".

O resultado será:

```
{"Extorsao":6,"Estelionato":62,"Furto de veiculos":72,"Roubo de aparelho celular":30}
```

yaml

O YAML é outro formato simples de serialização de dados, semelhante em estrutura ao JSON – inclusive com a vantagem de um gerador de JSON poder ser facilmente adaptado para emitir YAML, e vice-versa –, que oferece um padrão de dados simples de ser interpretado tanto por humanos quanto por *softwares*.

Passa-se a estrutura de dados anterior para YAML fazendo:

```
use YAML::XS;
my $yaml = Dump( $crimes_violentos );
```

Não se esqueça de definir o *Content-Type* da página gerada para "text/yaml" ou derivado.

O resultado será algo como:

```
---
Estelionato: 62
Extorsao: 6
Furto de veiculos: 72
Roubo de aparelho celular: 30
```

xml

O XML foi um dos formatos padronizados mais populares para serialização de dados via rede. Atualmente, muitos desenvolvedores evitam seu uso em *web services* REST, optando por alternativas mais simples, como JSON ou YAML. Mas o XML ainda é um bom candidato, especialmente quando se quer validações mais robustas, usando os chamados XML-*schemas*.

Se o arquivo contendo o schema do XML estiver em "dados.xsd", pode-se gerar um XML da seguinte forma:

```
use XML::Compile::Schema;
use XML::LibXML;

my $doc = XML::LibXML->createDocument('1.0', 'UTF-8');
my $schema = XML::Compile::Schema->new( 'dados.xsd' );

my $xml = $schema->compile( WRITER => '{http://isp.rj.gov.br/api}
dados' )
    ->($crimes_violentos, $hash)
    ->toString;
```

Há pouca vantagem em usar XML sem validação, mas isso também pode ser obtido com relativa facilidade. Não se pode esquecer de definir o *Content-Type* da página gerada para "text/xml".

estruturando dados – ruby com json, yaml e xml

Para criar este tipo de aplicação utilizando-se o Sinatra,[9] micro-*-framework* para desenvolvimento de aplicações *web* na linguagem Ruby, comece da seguinte forma:

```
require 'sinatra'
get '/:ano/:mes/:aisp/:tipo' do
...
end
```

E, então, basta colocar o aplicativo respondendo em http://isp.rj.gov.br/api.

Para os exemplos a seguir, supõe-se que os dados tenham sido obtidos de um banco de dados e estejam em uma estrutura de dados do tipo Hash como esta:

```
@crimes_violentos = {
'Roubo de aparelho celular' => 30,
'Furto de veiculos'         => 72,
'Extorsao'                  => 6,
'Estelionato'               => 62,
```


[9] Página web do *framework* Sinatra, em Português: http://www.sinatrarb.com/intro-pt-br.html.

json

Passar a estrutura de dados acima para JSON é simples. No ecossistema Ruby, existem diversas opções de *parsers* e geradores de JSON, sendo que a mais comum é a biblioteca conhecida simplesmente como "json":

```
require 'json'
@json = @crimes_violentos.to_json
```

Agora é possível "renderizar" a variável @json, mas não se esqueça de definir o *Content-Type* da página para "application/json". Complementando o exemplo acima, em Sinatra haveria algo parecido com seguinte:

```
require 'sinatra'
get '/:ano/:mes/:aisp/:tipo' do
...
content_type :json
@crimes_violentos.to_json
end
```

E o resultado será:

```
{"Extorsao":6,"Estelionato":62,"Furto de veiculos":72,"Roubo de aparelho celular":30}
```

yaml

Pode-se passar a estrutura de dados anterior para YAML, fazendo:

```
require 'yaml'
@yaml = @crimes_violentos.to_yaml
```

Não se esqueça de definir o *Content-Type* da página gerada para "text/yaml", semelhantemente ao exemplo de Sinatra com JSON.

O resultado será algo como:

```
---
Estelionato: 62
Extorsao: 6
Furto de veiculos: 72
Roubo de aparelho celular: 30
```

xml

Se o arquivo contendo o *schema* do XML estiver em "dados.xsd", pode-se gerar um XML da seguinte forma:

```
require 'libxml
require 'xmlsimple'
require 'rest_client'

@xml = XmlSimple.xml_out(@crimes_violentos, "RootName" => "CrimesViolentos")
document = LibXML::XML::Parser.string(@xml)
schema = LibXML::XML::Schema.new("dados.xsd")
```

```
RestClient.post('http://isp.rj.gov.br/api',@xml) if document.validate_
schema(schema)
```

Uma das vantagens de se utilizar XML é a opção de uma validação mais rígida de sua estrutura com XML Schemas. Mas não se esqueça de definir o *Content-Type* da página gerada para "text/xml".

a escolha é do freguês

A transformação de estruturas de dados nestes formatos é tão simples, que muitos *frameworks* de desenvolvimento de aplicações para *web*, como Catalyst, Mojolicious e Dancer para linguagem Perl, ou como Ruby on Rails, Sinatra e Merb, permitem especificar seu formato por meio de um parâmetro de extensão na URI, como se fosse a extensão de um arquivo – forma não recomendada, uma vez que a rotina chamada "*content negociation*" em *web* semântica e implementações de serviço REST deve ser via cabeçalho *Content-Type*, e não via extensão. Pode-se oferecer o mesmo serviço em diferentes formatos, conforme a preferência do sistema do usuário.

Ignorando-se o *Content-Type*, ainda assim é possível oferecer a alternativa das extensões. Seguindo o exemplo anterior, ter-se-ia a URI http://isp.rj.gov.br/api/2009/05/AISP7/crimes-violentos.json, que poderia renderizar sua saída em formato JSON. Já a URI http://isp.rj.gov.br/api/2009/05/AISP7/crimes-violentos.yaml poderia renderizar essa mesma informação em YAML, e assim por diante.

começando com rdf

A *web* pode ser considerada um local universal para se armazenar documentos, e isso significa torná-la uma grande tabela de dados dinâmicos em um formato estruturado.

Utilizando a infraestrutura da Internet, os dados podem crescer continuamente, desde que exista uma identificação simples. A melhor forma de fazer isso é por meio do esquema oferecido pelas URIs, ou seja, os dados serão identificados por endereços na *web*.

O RDF é um modelo padrão para troca de informações na rede que facilita tarefas como a mescla de dados, testando *schemas* diferentes e propiciando a evolução dos dados e seus *schemas*. Ele estende o modelo de estrutura de ligação (*links*) da *web* e chega ao relacionamento de documentos via nomes de URIs, interligando-os. Dessa forma, é possível criar estruturas de dados que podem ser misturados, compilados, interpretados e compartilhados utilizando-se aplicativos diversos.

A base do RDF[10] é a "tripla", que define o sujeito, o predicado e o objeto utilizados para descrever uma informação na *web* e o relacionamento entre elas. É assim que será descrito o RDF: um conjunto de dados e seus relacionamentos em forma de triplas.

Os conceitos do RDF são:

- Modelo de dados gráfico;

[10] Mais detalhes sobre o *framework* RDF podem ser encontrados na documentação oficial do W3C (World Wide Web Consortium), "Resource Description Framework (RDF): Concepts and Abstract Syntax", de 10.2.2004, encontrado na página *web* http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

- Vocabulário baseado em URIs;
- Tipos de dados literais;
- Sintaxe de serialização XML;
- Expressão de fatos simples;
- Vinculação.

criando um documento em rdf com perl

Existem vários módulos disponíveis em http://search.cpan.org, mas abaixo é apresentado um exemplo simples.

FOAF

Uma prova do conceito de RDF é criar um documento para demonstrar a facilidade oferecida pela linguagem PERL. Será utilizado o FOAF, um tipo de conjunto de dados já comentado anteriormente. FOAF é o acrônimo para "Friend of Friend" [Amigo do amigo]. Trata-se de uma ontologia para descrever dados de pessoas, suas atividades, o relacionamento com outras pessoas e objetos etc.

Existem alguns módulos para se tratar este tipo de *dataset* no CPAN, XML::FOAF[11], Gedcom::FOAF[12] e XML::FOAF::Person.[13]

[11] http://search.cpan.org/perldoc?XML%3A%3AFOAF
[12] http://search.cpan.org/perldoc?Gedcom%3A%3AFOAF
[13] http://search.cpan.org/perldoc?XML%3A%3AFOAF%3A%3APerson

Porém, neste exemplo, será utilizado o RDF::Simple::Serialiser.[14]

```
use RDF::Simple::Serialiser;

my $ser = RDF::Simple::Serialiser->new;
# Qual é o vocabulário que será usado? Veja o RDF::Simple::NS
# para mais detalhes.
$ser->addns( foaf => 'http://xmlns.com/foaf/0.1/' );

# Gerar um identificador randômico.
my $node1 = $ser->genid;
my $node2 = $ser->genid;

# Informações na estrutura perl.
my @triples = (
[ $node1, 'foaf:name', 'Thiago Rondon'],
[ $node1, 'foaf:know', $node2 ],
[ $node2, 'foaf:name', 'Alice Rondon'],
[ $node2, 'foaf:type', 'foaf:Person'],
);

# Serializando em RDF.
print $ser->serialise(@triples);
```


[14] http://search.cpan.org/perldoc?RDF%3A%3ASimple%3A%3ASerialiser

O resultado do código será:

```
<rdf:RDF
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
<rdf:Description rdf:about="_id:85623079">
<foaf:name>Alice Rondon</foaf:name>
<foaf:type>foaf:Person</foaf:type>
</rdf:Description>
<rdf:Description rdf:about="_id:38921350">
<foaf:name>Thiago Rondon</foaf:name>
<foaf:know rdf:nodeID="_id:85623079"/>
</rdf:Description>
</rdf:RDF>
```

criando um documento em rdf com ruby

Existem vários módulos disponíveis em http://search.cpan.org, mas abaixo é apresentado um exemplo simples.

FOAF

Uma prova do conceito de RDF é criar um documento para demonstrar a facilidade oferecida pela linguagem Ruby. Será utilizado o FOAF, um tipo de conjunto de dados já utilizados no exemplo com PERL.

No mundo Ruby, existem algumas iniciativas para lidar com RDF, como o projeto Spira. A seguir, vê-se um exemplo de como criar, em Ruby, uma representação RDF de FOAF. Mais do que isso, trata-se de uma forma de *design* de alto nível para modelar um RDF na forma de objetos facilmente manipuláveis por um programador:

```
require 'spira'
require 'rdf/ntriples'

repo = "http://datagraph.org/jhacker/foaf.nt"
Spira.add_repository(:default, RDF::Repository.load(repo))

class Person
    include Spira::Resource

    property :name,  :predicate => FOAF.name
    property :nick,  :predicate => FOAF.nick
end

jhacker = RDF::URI("http://datagraph.org/jhacker/#self").as(Person)
jhacker.name #=> "J. Random Hacker"
jhacker.nick #=> "jhacker"
jhacker.name = "Some Other Hacker"
jhacker.save!
```

Se for interessante, é possível serializar o objeto de modelo acima na forma de uma string XML/RDF, da seguinte forma:

```
require 'rdf/rdfxml'
output = RDF::Writer.for(:rdfxml).buffer do |writer|
    jhacker.each_statement do |statement|
    writer << statement
    end
end
```

O resultado do código será:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
xmlns:ns0="http://xmlns.com/foaf/0.1/">
    <rdf:Description rdf:about="http://datagraph.org/
    jhacker/#self">
        <ns0:name>Some Other Hacker</ns0:name>
        <ns0:nick>jhacker</ns0:nick>
    </rdf:Description>
</rdf:RDF>
```

buscando informações

Com os dados disponíveis em RDF, é necessário um padrão para buscá-los de forma inteligente, e uma das soluções é a linguagem de consulta SPARQL (um SQL para a *web*, uma linguagem de procura e pesquisa por dados estruturados). Em resumo, se

as informações desestruturadas da *web* tivessem representações alternativas, estruturadas como RDF, isso seria o equivalente ao Google da *web* não estruturada.

Um exemplo de esforço em estruturar informações é a DBpedia.org, uma versão em RDF que tenta estruturar a gigante Wikipédia de forma a ser possível realizar pesquisas com o SPARQL. Para ver como funciona, tente descobrir como escrever o nome da cidade de Tóquio usando *kanjis* (caracteres japoneses). Pergunte, via SPARQL, à DBpedia.org, que conseguirá pesquisar na base estruturada do Wikipédia com RDF:

```
require 'sparql/client'

    @client = SPARQL::Client.new('http://dbpedia.org/sparql')

    @query = "SELECT ?name WHERE { <http://dbpedia.org/resource/
    Tokyo> <http://dbpedia.org/property/nativeName> ?name }"

    result = @client.query(@query, :content_type =>
    SPARQL::Client::RESULT_JSON).first
```

E o resultado será:

```
    puts result[:name]

    => "東京"
```

conclusão

A democratização e a transparência dos dados da administração pública é sem dúvida louvável, e os órgãos responsáveis estão realizando um trabalho cada vez melhor na disponibilização de dados completos para análise pela população. No entanto a escolha de formatos apropriados para coleta, estruturação e exposição de dados via Internet é fundamental para que esse processo ocorra com a agilidade e a eficiência necessárias.

extraindo dados públicos

apresentação

Depois de expostos os dados, o desafio é extraí-los das páginas *web* ou de repositórios de dados de forma que possam ser reutilizados de acordo com o interesse e conveniência do usuário.

Se os dados estão em formato aberto, a tarefa é bem mais simples; mas nem sempre os dados estão expostos em formato compreensível por máquina, e isso torna a tarefa do interessado bem mais complexa.

Neste capítulo são apresentados exemplos práticos de como extrair dados de formatos mais comuns na *web* como CSV, XLS ou até mesmo fazer *scrapping* do conteúdo da página.

lendo csv: entre pipe e dbms

O formato CSV (*comma-separated values*) é o verdadeiro "idioma inglês" para transferência de dados: quase todo aplicativo de planilha ou DBMS (*database management system*) oferece CSV como formato de importação ou exportação.

O único formato que pode ser considerado mais simples do que CSV é o TSV (*tab-separated values*), que nada mais é do que um caso particular de CSV. O formato TSV está de certa forma embutido em comandos para sistema operacional (*scripts* em *shell*). Por exemplo, *xinput -list | cut -f 2* retorna os identificadores dos dispositivos de entrada do X11. O próprio Perl já lida muito bem com os dados delimitados, via *inline*:

```
# extrai o PID e a linha de comando dos processos operantes
ps auwx | perl -anE '$,="\t"; say @F[1,10]'
```

Do ponto de vista da eficiência, separar pelo delimitador é muito melhor, entretanto alguns dados são suficientemente elaborados, a ponto de empregarem o próprio delimitador como parte dos dados. Por outro lado, colunas separadas por caractere invisível definitivamente não são uma forma compreensível por pessoas de representar os dados que, especialmente quando existem muitas colunas e/ou colunas de valor indefinido. É aí que o CSV se mostra relevante.

Apesar de não existir uma padronização rígida do que seja um CSV válido, o formato é bem intuitivo, e a definição em RFC 4180[15] já ajuda bastante.

Embora seja possível implementar um *parser* de CSV a partir do zero – é só tratar o *escaping* em delimitadores –, felizmente, o CPAN[16] já tem soluções robustas para todos os gostos.

módulos de cpan para tratar csv

Uma busca por "CSV" no CPAN retorna muitas variações sobre o tema. Sem o intuito de desmerecer outras soluções, dar-se-á atenção apenas a Text::CSV, Tie::Handle::CSV e DBD::CSV, por serem abordagens bastante ortogonais.

text::csv

É a opção mais flexível, pois grava e lê tanto arquivos como *strings*, e serve como base para muitos outros módulos (inclusive os outros dois que serão explicados adiante).

Quando são definidas os *column_names* (por meio de um *ArrayRef*), pode-se usar *getline_hr*, que retorna uma *HashRef* para cada linha processada. Já *getline* retorna diretamente *ArrayRef*. Quando estiver trabalhando, é interessante saber que a primeira linha do CSV frequentemente traz os nomes das colunas. Assim, leia estes com *getline* e passe-os para *column_names*. O resto do arquivo pode ser lido com *getline_hr*.

[15] Disponível em http://tools.ietf.org/html/rfc4180.

[16] CPAN (Comprehensive Perl Archive Network) é um repositório virtual de módulos programados na linguagem PERL, disponível em http://www.cpan.org/.

Para gerar CSV, pode ser usado *print*, que grava diretamente em um *FileHandle*, ou uma combinação de *combine* e *string* para gerar um *buffer*.

Vejamos um exemplo prático. Tem-se o seguinte CSV, e é preciso filtrá-lo, deixando apenas código identificador, nome da cidade e UF:

```
estado,cidade,longitude,latitude,link
    MG, Santa Maria do Salto, -40.14935, -16.24953,
http://www.sidra.ibge.gov.br/bda/territorio/infomun.asp?codmun
=3158102
    ES, Marilândia, -40.54236, -19.41355,
http://www.sidra.ibge.gov.br/bda/territorio/infomun.asp?codmun
=3203353
    GO, Estrela do Norte, -49.07341, -13.86851,
http://www.sidra.ibge.gov.br/bda/territorio/infomun.asp?codmun
=5207501
 ...
```

O script que emprega o Text::CSV para isso seria:

```
#!/usr/bin/perl
use strict;
use utf8;
use warnings 'all';

use open ':locale';

use Data::Dumper;
use Text::CSV;
```

```
# Não é necessário definir EOL quando só se vai ler o CSV,
# pois o valor de $/ será usado automaticamente.
# Já para gravar, é preciso definir EOL.

my $csv = new Text::CSV({ eol => "\n" })
or die "Erro com Text::CSV: " . Text::CSV->error_diag;
# Assume que a codificação do arquivo é UTF-8.
open my $fh, '<:utf8', '7marco2011.csv';

# Pega a primeira linha do CSV e extrai os nomes das colunas.
$csv->column_names($csv->getline($fh));

# Pega todas as outras linhas e retorna para cada uma HashRef
# onde as chaves são os nomes das colunas.
while (my $row = $csv->getline_hr($fh)) {
    ($row->{codigo}) = ($row->{link} =~ m{\bcodmun=([0-9]+)\b}i);
    print STDERR Dumper $row;
    $csv->print(\*STDOUT, [ map { $row->{$_} // '' } qw(codigo ci-
    dade estado) ]);
}
$csv->eof or $csv->error_diag;
close $fh;
```

E o *output* será:

```
...
$VAR1 = {
    'link' => 'http://www.sidra.ibge.gov.br/bda/territorio/infomun.
    asp?codmun=4109609',
    'cidade' => 'Guaratuba',
    'longitude' => '-48.57544',
    'latitude' => '-25.88355',
    'codigo' => '4109609',
    'estado' => 'PA'
    };
4109609,Guaratuba,PA
$VAR1 = {
    'link' => 'http://www.sidra.ibge.gov.br/bda/territorio/infomun.
    asp?codmun=3541109',
    'cidade' => 'Presidente Alves',
    'longitude' => '-49.43844',
    'latitude' => '-22.10054',
    'codigo' => '3541109',
    'estado' => 'SP'
    };
3541109,"Presidente Alves",SP
...
```

Vale a pena estudar também os *bind_columns*, que associa diretamente os campos do CSV às variáveis e aos atributos do método *new*. A configuração padrão é:

```
$csv = new Text::CSV({
    quote_char          => '"',
    escape_char         => '"',
    sep_char            => ',',
    eol                 => $\,
    always_quote        => 0,
    quote_space         => 1,
    quote_null          => 1,
    binary              => 0,
    keep_meta_info      => 0,
    allow_loose_quotes  => 0,
    allow_loose_escapes => 0,
    allow_whitespace    => 0,
    blank_is_undef      => 0,
    empty_is_undef      => 0,
    verbatim            => 0,
    auto_diag           => 0,
});
```

É fácil ajustá-la para processar arquivos TSV (cujas colunas são separadas por *tabs*):

```
$csv = new Text::CSV({
    quote_char          => undef,
    escape_char         => undef,
    sep_char            => "\t",
    eol                 => "\n",
```

```
    quote_space         => 0,
    quote_null          => 0,
});
```

tie::handle::csv

É a versão do Text::CSV que acessa mais facilmente dados em CSV, pois combina muito melhor com um código orientado a objetos, fazendo bom uso de *overloading* e normalizando os *headers* para caixa alta/baixa.

Não é tão suscetível a *tuning* quanto o módulo que encapsula, mas em 99% dos casos é exatamente isso que se quer. Segue um exemplo com funcionalidade similar ao anterior:

```
#!/usr/bin/perl
use strict;
use utf8;
use warnings 'all';

use open ':locale';
use Data::Dumper;
use Tie::Handle::CSV;

my $csv = new Tie::Handle::CSV(
    file        => '7marco2011.csv',
    header      => 1,
    key_case    => 'lower',
    open_mode   => '<:utf8',
```

```
);

while (my $row = <$csv>) {
    $row->{link} =~ s/^.*=//;
    print STDERR Dumper $row;
    print $row . "\n";
}

close $csv;
```

dbd::csv

Muitas vezes, quando um projeto está na fase de prototipagem, não compensa "matar passarinho com bazuca", criando *schemas* em um RDBMS (Relational Data Base Management Systems). Outros projetos simplesmente não atingem o patamar mínimo para ser necessário utilizar o SQLite,[17] sendo muito mais prático manter as tabelas com apenas um editor de texto. Para aproveitar o melhor dos dois métodos, existe o DBD::CSV, um *driver* para interfaces de banco de dados que trabalha diretamente com arquivos CSV, usando o Text::CSV como *backend*. A grande vantagem é que o código pode ser facilmente escalado, trocando o *driver* por qualquer outro e importando as tabelas. Quanto ao subconjunto de SQL implementado, ele é suficientemente completo.

Voltando ao exemplo com o mesmo conjunto de dados do exemplo anterior, dessa vez, a sigla de UF será expandida para o

[17] A SQLite é uma biblioteca que implementa linguagem de pesquisas em banco de dados relacional.

nome do estado (por meio de um *JOIN*), e serão mostrados somente os municípios que fazem parte da Grande São Paulo:

```
#!/usr/bin/perl
use strict;
use utf8;
use warnings 'all';

use open ':locale';

use DBI;

my $dbh = DBI->connect('dbi:CSV:', undef, undef, {
    f_encoding  => 'utf8',
    csv_tables  => {
        llcb    => { file => '7marco2011.csv' },
        estados => { file => 'estados.csv' },
    },
    RaiseError  => 1,
    PrintError  => 1,
}) or die "Erro com DBI/DBD::CSV: " . $DBI::errstr;

my $sth = $dbh->prepare(<<SQL_QUERY);
    SELECT cidade, estados.estado AS cidade_uf
    FROM llcb
    JOIN estados
        ON llcb.estado = estados.uf
```

```
        WHERE
        llcb.estado = 'SP'
        AND (latitude   > -23.80)
        AND (latitude   < -23.20)
        AND (longitude  > -47.10)
        AND (longitude  < -46.10)
    ORDER BY cidade
SQL_QUERY

$sth->execute;
while (my $row = $sth->fetchrow_arrayref) {
    printf("%s, %s\n", @{$row});
}
$sth->finish;
$dbh->disconnect;
```

Além de *SELECT* (com *JOIN* e *ORDER!*), *INSERT*, *DELETE* e *UPDATE* também são implementados. A documentação completa está em SQL::Statement::Syntax.

acelerando com o text::csv_xs

Quando o módulo Text::CSV_XS está instalado, o Text::CSV automaticamente faz o uso do mesmo, proporcionando performance consideravelmente melhor:

```
                s/iter Text::CSV_PP Text::CSV_XS
Text::CSV_PP    23.6          --         -90%
Text::CSV_XS    2.27         941%          --
```

retornando csv via http

Em várias ocasiões, é interessante que os usuários possam exportar dados dos sistemas *on-line* diretamente para seus *desktops*. Para isso, basta configurar os *headers* de forma que o sistema do usuário encaminhe a planilha diretamente para o aplicativo especializado (Excel, LibreOffice), ao invés de exibir o conteúdo do CSV na tela do navegador.

Por exemplo, se fosse usado o Catalyst:

```
$c->res->headers->content_type('application/vnd.ms-excel;
charset: iso-8859-1');
$c->res->header('Content-Disposition' =>
'attachment;filename=' . $filename);
```

Previsivelmente, tem-se um *caveat* relativo à codificação: o Excel espera que o CSV esteja em ISO-8859-1 (ou, possivelmente, Windows-1252), sendo então necessário um *downgrade* para latin1.

google fusion tables

Como se tem uma quantidade razoável de dados em formato CSV, fazer um *overview* de forma rápida e intuitiva pode ser um problema, principalmente quando se quer mostrar os dados para leigos (que não têm obrigação de saber SQL). Neste caso, é de grande ajuda o Fusion Tables, um experimento do Google Labs.

Segundo a descrição do próprio Google:

> Fusion Tables é um serviço para gerenciar grandes coleções de dados tabulares na nuvem. Pode-se enviar tabelas de até 100 MB e compartilhá-las com colaboradores, ou torná-las públicas. É possível aplicar filtros e agregação aos dados, visualizá-los em mapas e em outros gráficos, mesclar os dados de diversas tabelas e exportá-los para a *web* ou para arquivos CSV. Pode-se também conduzir discussões sobre os dados em diversos níveis de granularidade, como linhas, colunas e células individuais.

Nada mais é do que um "Excel sob o efeito de esteróides", pois, além dos filtros, é possível fazer *joins* e criar *views*. Para uma demonstração, o arquivo usado como exemplo no decorrer desta cartilha, o *dump* do banco de dados do projeto "Latitudes e longitudes das cidades brasileiras", de Thiago Rondon, encontra-se em http://j.mp/hyoYqi.

mysqldump/mysqlimport

O MySQL oferece importação/exportação otimizada para arquivos CSV localizados no mesmo *host* em que o servidor (mysqld) está rodando. Ou seja, tendo-se certos privilégios, é possível "passar por cima" do sistema de *queries*:

```
mysqldump \
    --fields-terminated-by="," \
    --fields-optionally-enclosed-by="\"" \
    --lines-terminated-by="\n" \
    -u $USERNAME -p -t -T/caminho/para/diretorio $DA-
    TABASE
```

Este comando grava todas as tabelas de $DATABASE em arquivos CSV individuais no diretório /caminho/para/diretorio. É necessário que este diretório tenha permissão para gravação aberta para o usuário que executa o daemon, pois os arquivos CSV são gerados diretamente pelo mesmo, e não pelo mysqldump.
A importação é feita da seguinte maneira:

```
mysqlimport \
    --fields-terminated-by="," \
    --fields-optionally-enclosed-by="\"" \
    --lines-terminated-by="\n" \
    -u $USERNAME -p $DATABASE table.csv
```

O nome da tabela onde será feita a importação é deduzido do nome do arquivo CSV (*table*, no caso).
Para mais detalhes, ver:

* mysqldump — A Database Backup Program
* mysqlimport — A Data Import Program
* LOAD DATA INFILE Syntax

lendo xls e xlsx

Geralmente, boa parte das informações dos clientes está em planilhas de Excel, e, neste trecho, o leitor verá como extrair dados de XLS e XLSX.

Até 2000, a única forma de ler XLS com Perl era usando o Win::OLE; no Linux, não era possível. Mas, em 2001, Takanori Kawai e John McNamara criaram o Spreadsheet::WriteExcel e o Spreadsheet::ParseExcel, que, além de oferecer mais operações para manipulação dos dados, era mais rápido e funcionava no Linux. E quem usava Windows também saiu ganhando, pois não era mais necessário ter o Excel instalado no computador.

spreadsheet::parseexcel

É um módulo que permite leitura de XLS. Na sua forma mais básica, é possível ler os arquivos XLS e acessá-los dizendo qual a planilha, linha e coluna:

workbook:
é o documento inteiro, e nele existem várias *worksheets*.

worksheet:
a planilha que será de fato utilizada; equivale a cada folha (ou aba) exibida no Excel.

Cada *worksheet* disponibiliza os métodos *row_range()* e *col_range()*, e ambos retornam *arrays*, com o mínimo e máximo de seus respectivos nomes (linhas/colunas).

Outro método disponibilizado pelo *worksheet* é o *get_cell($row, $col)*, que retorna a célula daquela posição/planilha. O retorno do *get_cell* é um objeto Spreadsheet::ParseExcel::Cell, e pode-se, entre outros, utilizar os métodos *value()* e *unformatted()*. A principal diferença é que o *unformatted()* retorna os valores da forma que eles foram salvos, e o *value()* é o valor que o Excel exibe.

Quem usa bastante Excel sabe que as datas são como números, mudando apenas a exibição da célula.

Vejamos um exemplo de código.

```
#!/usr/bin/perl -w

use strict;
use Spreadsheet::ParseExcel;

my $parser   = Spreadsheet::ParseExcel->new();
my $workbook = $parser->parse('nome_do_arquivo.xls');

if ( !defined $workbook ) {
    die $parser->error(), ".\n";
}

for my $worksheet ( $workbook->worksheets() ) {

    my ( $row_min, $row_max ) = $worksheet->row_range();
    my ( $col_min, $col_max ) = $worksheet->col_range();

    for my $row ( $row_min .. $row_max ) {
        for my $col ( $col_min .. $col_max ) {
```

```
            my $cell = $worksheet->get_cell( $row, $col );
            next unless $cell;

            print "Row, Col    = ($row, $col)\n";
            print "Value       = ", $cell->value(),       "\n";
            print "Unformatted = ", $cell->unformatted(), "\n";
            print "\n";
            }
        }
    }
```

Um grande problema em XLS é a quantidade de memória utilizada. Por isso, se é desejado apenas extrair as informações básicas do XLS (como o valor da célula), não é necessário montar tudo na memória. O manual informa que um arquivo de 10 Mb é um arquivo grande para uma máquina com 256 MB de RAM, portanto uma planilha de 60 Mb é um arquivo bem grande para uma máquina com 1 GB de RAM.

A solução usada pelo modulo é chamar uma função para cada célula lida, passando como parâmetro a *workbook*, o *index* da planilha, a linha, a coluna e o objeto da célula.

Exemplo:

```
    #!/usr/bin/perl -w

use strict;
use Spreadsheet::ParseExcel;
```

```
my $parser = Spreadsheet::ParseExcel->new(
    CellHandler => \&cell_handler,
    NotSetCell  => 1
);

my $workbook = $parser->parse('file.xls');

sub cell_handler {

    my $workbook    = $_[0];
    my $sheet_index = $_[1];
    my $row         = $_[2];
    my $col         = $_[3];
    my $cell        = $_[4];

    # pula alguns registros
    return if $sheet_index == 3;
    return if $row == 10;

    # imprime o valores formatados
    print $cell->value(), "\n";

}
```

lendo arquivos com cabeçalhos variáveis

Um problema surge quando o cliente envia a planilha numa ordem diferente, ou até mesmo totalmente diferente, sendo necessário recriar o *parsing*. Se for impossível alinhar isso com o cliente, ainda há uma solução: procurar pelos cabeçalhos.

Para isso, cria-se uma planilha conforme abaixo:

NOME	IDADE	TWITTER
Renato	19	@renato_cron
Eden		@edenc
Glupo	33	

Porém, dependendo da vontade do desenvolvedor, as informações podem estar em qualquer posição (por exemplo, o Twitter pode estar antes da idade).

```
#!/usr/bin/perl -w

use strict;
use Spreadsheet::ParseExcel;

my $parser   = Spreadsheet::ParseExcel->new();
my $workbook = $parser->parse('teste.xls');
if ( !defined $workbook ) {
    die $parser->error(), ".\n";
```

```
}

my %expected_header = (
    twitter => qr /\btwitter\b/io,
    idade   => qr /\bidade\b/io,
    nome    => qr /\bnome\b/io
);

# apenas para exibir
my $reg_num        = 0;

for my $worksheet ( $workbook->worksheets() ) {

    my ( $row_min, $row_max ) = $worksheet->row_range();
    my ( $col_min, $col_max ) = $worksheet->col_range();

    my $header_map     = {};
    my $header_found   = 0;

    for my $row ( $row_min .. $row_max ) {

        if (!$header_found){
            for my $col ( $col_min .. $col_max ) {
            my $cell = $worksheet->get_cell( $row, $col );
        next unless $cell;
            foreach my $header_name (keys %expected_header){
            if ($cell->value() =~ $expected_header{$header_name}){
```

```
            $header_found++;
            # mais informações poderia ser salvas, como por exem-
            plo
            # qual é o valor que está escrito e bateu com a regexpr

            $header_map->{$header_name} = $col;
            }
        }
    }
}else{

# aqui você pode verificar se foram encontrados todos os campos
que você precisa
# neste caso, achar apenas 1 cabeçalho já é o suficiente

my $registro = {};

foreach my $header_name (keys %$header_map){
    my $col = $header_map->{$header_name};

    my $cell = $worksheet->get_cell( $row, $col );
    next unless $cell;

    my $value = $cell->value();

    # aqui é uma regra que você escolhe, pois as vezes o valor
    da célula pode ser nulo
```

```
            next unless $value;

            $registro->{$header_name} = $value;
        }

            # se existe alguma chave, algum conteudo foi encontrado
            if (keys %$registro){
                    $reg_num++;
                    print "row $row, registro $reg_num\n";
                    print "$_ = $registro->{$_}\n" for keys %$registro;
                    print "----------\n";
                }
            }
        }
    }
```

E o resultado será algo como:

```
row 5, registro 1
nome = Renato
twitter =  @renato_cron
idade =  19
----------
row 6, registro 2
nome = Eden
twitter =  @edenc
idade =
```

```
----------
row 7, registro 3
nome = Glupo
twitter =
idade =  33
----------
```

O código deve ser alterado de acordo com a necessidade do desenvolvedor e do cliente.

Notem que foi adicionado um *next unless $value*, pois geralmente vem muita linha completamente em branco. São linhas criadas (quase) automaticamente pelo Excel.

encoding

Abra a planilha e troque o cabeçalho "nome" por "nome para validação". Se for executado o programa, tudo continua funcionado; assim, deve-se trocar a *regexpr*:

```
qr /\bnome para validação\b/io
```

Lembre-se de salvar o arquivo em UTF-8. Se executar novamente, o campo não será encontrado, pois é um texto em Perl (que está, portanto, em UTF8), e o padrão do Perl é considerar o código como sendo Latin1. Para resolver este problema, adicione no topo (junto com o *use strict*) o *use utf8*:

```
use strict;
use Spreadsheet::ParseExcel;
```

```
use utf8;

...
```

Uma dica é deixar com acento as *regexpr* do cabeçalhos apenas quando necessário, pois o cliente pode mudar sem querer:

```
qr /\bnome\s+(para\s+)?valida(r|[çc][ãa]o)\b/io
```

Assim, não importa se for escrito 'nome para validar', 'nome validaçao', 'nome validacao', pois todos serão aceitos.

xlsx

Até pouco tempo atrás, existia apenas o módulo SimpleXlsx. Mas McNamara também criou o Spreadsheet::WriteExcel, e com pequenas e rápidas modificações, é possível ler arquivos XLSX usando algumas classes do XLS.

Uma das poucas desvantagens do XLSX é que não se consegue utilizar o método para economizar memória, portanto, se estiver trabalhando um arquivo maior, prepare-se para fazer telas de "*loading*" bonitas para seu cliente.

Abaixo, vejamos o código de exemplo modificado, explicando-se cada alteração.

```
#!/usr/bin/perl -w


use strict;
use utf8;
```

```
use Spreadsheet::XLSX;

use Text::Iconv;
my $converter = Text::Iconv -> new ("utf-8", "latin1");

my $excel  = Spreadsheet::XLSX->new('teste.xlsx', $converter);

my %expected_header = (
    twitter => qr /\btwitter\b/io,
    idade   => qr /\bidade\b/io,
    nome    => qr /\bnome\s+(para\s*)?valida(r|[çc][ãa]o)\b/io
);

# apenas para exibir
my $reg_num        = 0;

for my $worksheet ( @{$excel -> {Worksheet}} ) {

    my ( $row_min, $row_max ) = $worksheet->row_range();
    my ( $col_min, $col_max ) = $worksheet->col_range();

    my $header_map     = {};
    my $header_found   = 0;

    for my $row ( $row_min .. $row_max ) {

        if (!$header_found){
```

```
        for my $col ( $col_min .. $col_max ) {
            my $cell = $worksheet->get_cell( $row, $col );
            next unless $cell;
            foreach my $header_name (keys %expected_header){
            if ($cell->value() =~ $expected_header{$header_
            name}){
                $header_found++;
                $header_map->{$header_name} = $col;
            }
        }
    }
}else{

# aqui você pode verificar se foram encontrados todos os campos
necessários
# neste caso, achar apenas um cabeçalho já é suficiente
my $registro = {};

foreach my $header_name (keys %$header_map){
    my $col = $header_map->{$header_name};

    my $cell = $worksheet->get_cell( $row, $col );
    next unless $cell;

    my $value = $cell->value();

    # aqui é uma regra que você escolhe, pois às vezes o valor
```

```
            da célula pode ser nulo
            next unless $value;

            $registro->{$header_name} = $value;
        }

            if (keys %$registro){
                $reg_num++;
                print "row $row, registro $reg_num\n";
                print "$_ = $registro->{$_}\n" for keys %$registro;
                print "-------------\n";
                }
            }
        }
    }
```

Pois bem, além do *use utf8*, recomenda-se utilizar o módulo Text::Iconv para não ter problemas com *encoding*.

```
use Spreadsheet::XLSX;
use Text::Iconv;
my $converter = Text::Iconv -> new ("utf-8", "latin1");
my $excel  = Spreadsheet::XLSX->new('teste.xlsx', $converter);
```

Pode parecer meio estranho, mas o único jeito de funcionar é trocando o windows-1251 por latin1 (iso-8859-1).

O windows-1251 está na documentação do Spreadsheet::XLSX, porém as *regexpr* não funcionam nem com *use utf8*. Mesmo removendo o *iconv*, o erro persiste, além de receber um *warning*, sendo assim preferível trocar para latin1.

Outra alteração, é que não existe mais a variável *$parser*, e sim *$excel*, que contém o Worksheet inteiro.

```
for my $worksheet ( @{$excel -> {Worksheet}} ) {
```

extração de dados, exportando em odf

Este artigo tem como objetivo apresentar um exemplo prático que envolva:

- Utilização de *Crawlers/Spider* para automatizar a navegação em *sites* com Perl;
- Xpath, para determinar onde estão os dados que se procura dentro de uma página *web*;
- Expressões regulares para reformatar *strings* de conteúdo de acordo com o que se precisa;
- Exportar dados do *site* em arquivo formato ODF (*openoffice spreadsheets*).

Para começar, acesse a URL

http://www.portaltransparencia.gov.br/despesasdiarias/resultado?consulta=rapida&periodoInicio=01%2F01%2F2011&periodoFim=31%2F01%2F2011&fase=EMP&codigoOS=51000&codigoFavorecido=.

Para obter esta URL, acessa-se o site http://www.portaltransparencia.gov.br/despesasdiarias e seleciona-se o período de 01/01/2011 a 31/01/2011, Empenho, Ministério do Esporte > consultar.

Analisemos alguns pontos importantes de uma página:

- Possui paginação?;
- Formatação dos dados, valores e *strings*: padrões dos dados devem ser observados para poder criar *regex* e arrumar os dados de acordo;
- *Layout* da página: como os dados estão dispostos? É uma tabela? São *divs*? Onde estão os dados necessários em uma determinada página?

Observe a página e note que ela possui paginação e tem os dados formatados em uma *table*. Com isso, já é possível perceber que será necessário acessar cada página e ler os dados que estão dentro da tabela. Quanto à formatação dos dados, navegando em algumas páginas, nota-se que os preços muito provavelmente precisarão ser reformatados para que o banco de dados os aceite como valores válidos.

obtendo os dados de uma página

Para acessar os dados, será utilizado o XPath, que é uma sintaxe para navegar entre atributos e elementos em um documento XML. O XPath está presente e é utilizado nas mais diversas linguagens

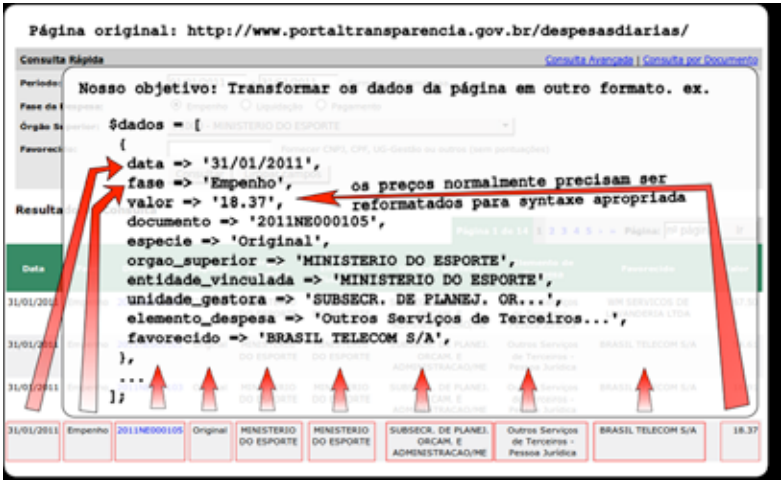
de programação. Sua sintaxe é similar a: "//html/body/div/span", que retornaria o valor do texto que está dentro do *span*. Leia mais sobre xpath em http://www.w3schools.com/xpath/default.asp.

Exemplos de notação XPath:

- //td[1] : retorna o primeiro <td>
- //td[position()=1] : retorna o primeiro <td>
- //table[@class='tabelao'] : retorna a table com class = 'tabelao'
- //table/td[1] : retorna o primeiro td de uma <table>
- /body/html//table/td[1] : retorna o primero td de uma tabela dentro do <html>

Fonte: http://sao-paulo.pm.org/equinocio/2011/mar/3

O objetivo agora é carregar os dados da página dentro de uma estrutura de dados criada especificamente para o *site*.



Fonte: http://sao-paulo.pm.org/equinocio/2011/mar/3.

analisando texto com *regex*

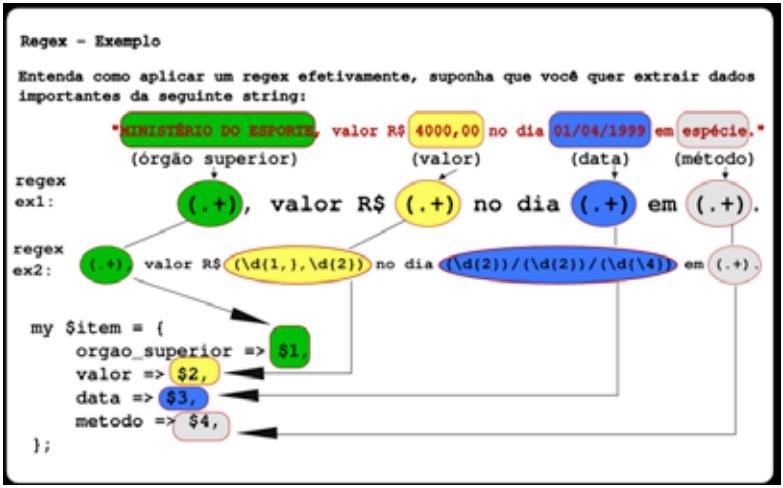
Depois de obter os dados, eles serão "parseados" e será feita uma reformatação de alguns. O objetivo é adaptá-los aos formatos necessários para o desenvolvedor. Para isto, serão utilizadas expressões regulares de forma a analisar todo o conteúdo obtido e transformá-lo de acordo com o que se precisa (isso pode ser feito já na hora da leitura dos dados). Dois exemplos de como aplicar *regex* em um texto:

```
use strict;
use warnings;
my $string = 'Ministerio do esporte, valor R$ 4000,00 no dia 01/02/1999 em espécie.';
```

```
$string =~ m/(.+), valor R\$ (.+) no dia (.+) em (.+)./ ;
    print $1,"\n";
    print $2,"\n";
    print $3,"\n";
    print $4,"\n";
    print "-----","\n";

$string =~ m/(.+), valor R\$ (\d{1,},\d{2}) no dia (\d{2})\/(\d{2})\/
(\d{4}) em (.+)./ ;
    print $1,"\n";
    print $2,"\n";
    print $3, '/', $4, '/', $5,"\n";
    print $6,"\n";
    print "-----","\n";
```

Veja na imagem:



Fonte: http://sao-paulo.pm.org/equinocio/2011/mar/3.

A tabela abaixo apresenta os *regex* simples mais utilizados.

[A-Za-z0-9]	Apenas caracteres alfanuméricos
[A-Za-z0-9_]	Alfanuméricos e _
\w	Alfanuméricos e _
[0-9]	Apenas dígitos
\d	Apenas dígitos
\D	Não dígitos
\s	Espaço
\S	Não espaço
^	String começando em
$	String terminando em
[^a-zA-Z0-9_]	Negação/Não alfanuméricos
\n	Nova linha
\r	Carriage return
{min,max}	Quantidade de ocorrências
()	Agrupa itens dentro do parêntesis
|	Separa alternativas
.	Algum caracter
(.+)	Alguma coisa obrigatoriamente
(.*)	Alguma coisa opcionalmente

Tabela 1. Tabela dos caracteres mais comuns para compor expressões regulares.

navegação e web crawler com perl

Normalmente, as duas opções de navegação via Perl mais comuns são:

- WWW::Mechanize (utilizado no exemplo). Mais detalhes em http://search.cpan.org/perldoc?WWW::Mechanize;
- LWP, mais detalhes em: http://search.cpan.org/perldoc?LWP.

como criar um arquivo openoffice usando perl

Um ótimo módulo para fazer este tipo de trabalho é OpenOffice::OODoc.

No entanto, não se consegue colocar os dados obtidos na tabela (*spreadsheet*) inicial do arquivo, sendo necessário criar uma nova tabela dentro do arquivo OpenOffice, e ai sim é possível salvar os dados por completo. Segue um exemplo do código:

```
sub exportar_e_salvar_odf {
    my ( $items ) = @_;
    my $document = odfDocument(file => 'portal_transparencia.ods',
    create => 'spreadsheet');
    my $table = $document->appendTable("TABELA DADOS",
    1000,50);
    my $line = 0;
    my $all_fields = get_fields();
    my $col = 0;
    foreach my $field ( qw/data fase documento especie orgao_su-
```

```
    perior entidade_vinculada unidade_gestora elemento_despesa
    favorecido valor/ ) {
        $document->updateCell( $table, $line, $col++, $all_fields->{
        $field }->{ label } );
    }

    foreach my $item ( @$items ) {
        $line++;
        $col = 0;
        foreach my $field ( qw/data fase documento especie or-
        gao_superior entidade_vinculada unidade_gestora elemento_
        despesa favorecido valor/ ) {
            $document->updateCell( $table, $line, $col++, $item->{
            $field } );
        }
    }
    $document->save;
}
```

E agora finalmente tudo isso será aplicado no código para gerar um arquivo de planilha com a extensão ods (Open Office). Este código tem por objetivo baixar todos os dados da página do Ministério dos Esportes.

```
#!/usr/bin/perl

use strict;
use warnings;
```

```
use WWW::Mechanize;
use HTML::TreeBuilder::XPath;
use HTML::LinkExtor;
use OpenOffice::OODoc;

my $mech = WWW::Mechanize->new();
$mech->agent_alias( 'Windows IE 6' );
my $url = 'http://www.portaltransparencia.gov.br/despesasdiarias/
resultado'.
    '?consulta=rapida&periodoInicio=01%2F01%2F2011&periodoFi
    m=31%2F01%2F2011'.
    '&fase=EMP&codigoOS=51000&codigoFavorecido=';
my @items = ();
my $url_visited = {};
scan_page( $url );

@items = reformata_dados_com_regex( \@items );
exportar_e_salvar_odf( \@items );

use Data::Dumper;
print "Resultados","\n", Dumper @items, "\n";
print "Para ver o arquivo em formato openoffice, abra o arquivo
criado: \n";
print "'portal_transparencia.ods' e veja a tabela de nome 'TABELA
DADOS'\n";

sub scan_page {
```

```
    my ( $link ) = @_;
    $mech->get( $link );
    my @links_paginacao = search_links_paginacao(
        $mech->res->content, '//p[@id="paginacao"]'
    );
    foreach my $url ( @links_paginacao ) {
        if ( ! $url_visited->{ $url } ) {
            print "Acessando url: ", $url, "\n";
            $mech->get( $url );
            my $pagina_html = $mech->res->content;
            parse_data( $pagina_html );
            $url_visited->{ $url } = 'visited';
            scan_page( $url );

            } else {
                #       print "url visitada","\n";
            }
        }
}

sub search_links_paginacao {
    my ( $html, $pagination_xpath ) = @_;

    my $tree= HTML::TreeBuilder::XPath->new;
    $tree->parse( $html );
    my $pagination_div = $tree->findnodes( $pagination_xpath );
    $pagination_div = $pagination_div->[0]->as_HTML;
```

```
    my $url_list  = HTML::LinkExtor->new();
    $url_list->parse( $pagination_div );
    my @links= ();
    foreach my $item ( @{ [ $url_list->links ] } ) {
        push( @links, 'http://www.portaltransparencia.gov.br' .
        @$item[2]);
    }
    $tree->delete;
    return @links;

    }

sub parse_data {
    my ( $html ) = @_;

    my $tree= HTML::TreeBuilder::XPath->new;
    $tree->parse( $html );

    my $tabela = $tree->findnodes( '//table[@class="tabela"]' );

    my $tabela_com_dados_html =  $tabela->[0]->as_HTML; #

    my $table_rows = $tree->findnodes( '//table[@class="tabela"]/
    tr' );
    my $count = 0;
    foreach my $row ( $table_rows->get_nodelist ) {
        if ( $count > 1 ){
```

```
        my $tree_tr = HTML::TreeBuilder::XPath->new;
        $tree_tr->parse( $row->as_HTML );
        my $row_data = {
            data              => $tree_tr->findvalue( '//td[1]' ),
            fase              => $tree_tr->findvalue( '//td[2]' ),
            documento         => $tree_tr->findvalue( '//td[3]' ),
            especie           => $tree_tr->findvalue( '//td[4]' ),
            orgao_superior    => $tree_tr->findvalue( '//td[5]' ),
            entidade_vinculada => $tree_tr->findvalue( '//td[6]' ),
            unidade_gestora   => $tree_tr->findvalue( '//td[7]' ),
            elemento_despesa  => $tree_tr->findvalue( '//td[8]' ),
            favorecido        => $tree_tr->findvalue( '//td[9]' ),
            valor             => $tree_tr->findvalue( '//td[10]' ),
    };
    push( @items, $row_data );
    $tree_tr->delete;
    }
    $count++;
    }

    $tree->delete;
    }

sub reformata_dados_com_regex {
    my ( $items ) = @_;
    for ( my $i = 0; $i < scalar( @$items ) ; $i++ ) {
        @$items[$i]->{ data } = join ( '/', $1, $2, $3 )
```

```
            if ( @$items[$i]->{ data }  =~ m/^(\d{4})-(\d{2})-(\d{2})/ ) ;
            #4dígitos traço 2 dígitos traço 2 dígitos
        @$items[$i]->{ valor } = join ( '.', $1, $2 )
            if ( @$items[$i]->{ valor }  =~ m/(\d{1,})\.(\d{2})(\d{2})$/ );
            #ao menos 1 dígito seguido de . e 2 dígitos de precisão.
        }
        return @$items;
}

sub exportar_e_salvar_odf {
    my ( $items ) = @_;
    my $document = odfDocument(
        file => 'portal_transparencia.ods',
        create => 'spreadsheet');
    my $table = $document->appendTable("TABELA DADOS",
    1000,50);
    my $line = 0;
    my $all_fields = get_fields();
    my $col = 0;
    foreach my $field ( qw/data fase documento especie orgao_su-
    perior entidade_vinculada unidade_gestora elemento_despesa
    favorecido valor/ ) {
        $document->updateCell( $table, $line, $col++, $all_fields->{
        $field }->{ label } );
    }

    foreach my $item ( @$items ) {
        $line++;
```

```
        $col = 0;
        foreach my $field ( qw/data fase documento especie or-
        gao_superior entidade_vinculada unidade_gestora elemento_
        despesa favorecido valor/ ) {
            $document->updateCell( $table, $line, $col++, $item->{
            $field } );
        }
    }
    $document->save;
}

sub get_fields {
    my ( $self ) = @_;
    return {
        data => {
            field => 'data',
            label => 'Data',
        },
        fase => {
            field => 'fase',
            label => 'Fase',
        },
        documento => {
            field => 'documento',
            label => 'Documento',
        },
        especie => {
            field => 'especie',
```

```
            label => 'Espécie',
        },
        orgao_superior => {
            field => 'orgao_superior',
            label => 'Órgão Superior',
        },
        entidade_vinculada => {
            field => 'entidade_vinculada',
            label => 'Entidade Vinculada',
        },
        unidade_gestora  => {
            field => 'unidade_gestora',
            label => 'Unidade Gestora',
        },
        elemento_despesa => {
            field => 'elemento_despesa',
            label => 'Elemento Despesa',
        },
        favorecido => {
            field => 'favorecido',
            label => 'Favorecido',
        },
        valor => {
            field => 'valor',
            label => 'Valor',
        },
    };
}
```

scrapping fácil e criação de *feeds* atom

A suíte Mojolicious de módulos para desenvolvimento *web* é mais conhecida por permitir a criação rápida e fácil de *sites* dinâmicos, mas oferece muitas outras facilidades, em particular para varredura e coleta de dados de outros sites, ou *web scrapping*.

Aqui será mostrado como é fácil escrever um *scraper* em Perl sabendo apenas seletores CSS. Para completar, há um pequeno desafio – a criação de *feeds* Atom – e será mostrado como o CPAN ajuda a desenvolver *softwares* como se fossem peças de Lego esperando para serem conectadas.

o cenário

Quer-se obter uma lista de todos os artigos publicados no *site* da São Paulo Perl Mongers[18]. A primeira coisa a fazer é entender a estrutura do documento HTML em que as informações estão. Para isso, abre-se a página que lista os artigos e olha-se o código fonte. Abaixo, um trecho do que foi encontrado:

```
<div class="top">
<h2>Artigos</h2>
<div class="whois">
    <h3>2010</h3>
    <ul>
        <li>
        <a
        href="/artigo/2010/comoescreverperlescalavelparaoseu
        clustermysql">Como Escrever Perl Escalável para o seu Cluster
        MySQL</a><span class="autor"> [Luis Motta Campos]</
        span>
        </li>
        <li>
        <a href="/artigo/2010/orliteumaabordagemsimplesparau
        tilizarsqlite">ORLite - Uma abordagem simples para utilizar
        SQLite</a><span class="autor"> [Daniel Vinciguerra]</
        span>
        </li>
```


[18] Disponível em http://sao-paulo.pm.org/artigos

```
    ...
</ul>
<h3>2009</h3>
<ul>
...
```

A página parece ter toda a informação necessária.

obtendo a página

É muito simples fazer um programa que acessa a página desejada:

```
use strict;
use warnings;
use Mojo::UserAgent;

my $client = Mojo::UserAgent->new->get( 'http://sao-paulo.pm.org/artigos' );
```

O Mojo::UserAgent é um cliente HTTP 1.1 e WebSocket completo, com E/S assíncrona e suporte transparente a TLS, *epoll e kqueue*. Sua API é bastante simples e direta, como se pôde observar, e a partir desse ponto do código o objeto *$client* já carregou o *site* e é possível acessar seu conteúdo.

o DOM de ler o conteúdo de *sites*

Após uma requisição *web*, o que interessa é a resposta obtida, ou, mais especificamente, o "DOM" da página recebida como resposta. O modelo de objetos de documentos (DOM, *Document Object Model*) é uma especificação do W3C que possibilita acesso e atualização dinâmicas do conteúdo, estrutura e estilo de documentos. É o que seu navegador constrói e interpreta para exibir o conteúdo de *sites*, e que vai ajudar imensamente na obtenção dos dados desejados.

Se o leitor não estiver familiarizado com o conceito, imagine um documento HTML como uma grande árvore hierárquica de *tags*, agrupadas e tratadas mais ou menos como a árvore de diretórios do seu sistema de arquivos. Veja novamente o trecho da página que se quer analisar:

```
<div class="top">
<h2>Artigos</h2>
<div class="whois">
    <h3>2010</h3>
    <ul>
        <li>
        <a  href="/artigo/2010/comoescreverperlescalavelparaoseu
        clustermysql">Como Escrever Perl Escalável para o seu Cluster
         MySQL</a><span class="autor"> [Luis Motta Campos]</
        span>
        </li>
        <li>
        <a href="/artigo/2010/orliteumaabordagemsimplespara
        utilizarsqlite">ORLite - Uma abordagem simples para utilizar
```

```
        SQLite</a><span class="autor"> [Daniel Vinciguerra]</
        span>
        </li>
        ...
    </ul>
    <h3>2009</h3>
    <ul>
    ...
```

No código acima, o primeiro possui a classe "top" como atributo e podem-se ver dois filhos: o <h2>, sem atributos, com o texto "Artigos" e nenhum filho; e o outro , com a classe "whois" como atributo e vários filhos. Desses filhos, veem-se dois <h3> e dois <ul>, e quando se abre o código completo da página encontram-se muitos outros. *Tags* no mesmo nível são irmãs, mesmo tendo tipos diferentes. Seguindo-se pela estrutura, vê-se que cada <ul> tem vários filhos do tipo <li>, os quais por sua vez contêm *tags* <a> com o *link* e título dos artigos, e *tags* com o nome dos autores entre colchetes.

acessando o DOM

Sabendo a nomenclatura, pode-se solicitar o DOM da resposta obtida pelo cliente *web* e acessar cada um desses elementos facilmente por meio de buscas por seletores CSS.

```
my $dom = $client->res->dom;
```

A chamada acima retorna um objeto Mojo::DOM, que oferece o conteúdo do *site* já devidamente processado como uma árvo-

re DOM XML/HTML5 minimalista e bastante relaxada, ou seja, funcional mesmo que o *site* em questão não tenha um html em conformidade com os padrões do W3C – o que, infelizmente, parece praxe nos dias de hoje.

O Mojo::DOM fornece dois métodos principais para a obtenção de elementos de uma página:

```
$dom->at( 'seletor' )
```

Retorna objeto representando o primeiro elemento que casa com o seletor especificado.

```
$dom->find( 'seletor' )
```

Retorna uma lista de objetos representando todos os elementos que casam com o seletor especificado.

Portanto, para obter a lista de elementos representando artigos publicados, pode-se fazer algo como:

```
my $artigos = $dom->find( 'div[class="whois"] > ul > li' );
```

Ou seja, quer-se todos os elementos "li" filhos de *tags* "ul" que sejam, por sua vez, filhas de *tags* "div" com a classe "whois". Confuso? Experimente ler da esquerda para a direita: quer-se ir de tags "div" com a classe "whois" para os "ul" filhos dela, e destes para os "li". Melhor?

Note que, assim, o DOM vai pegar todos os "ul" disponíveis no documento, ou seja, todos os artigos independentemente do ano – exatamente o que se procura.

seletores css

Seletores como o acima são utilizados em CSS para associar um estilo de formatação a uma determinada *tag* – ou conjunto de *tags* – HTML. É uma notação simples e ao mesmo tempo muito versátil, que há alguns anos vem sendo usada com sucesso por bibliotecas javascript como o jQuery, especialmente porque pode-se combinar seletores da forma que se achar mais adequada para acessar o(s) elemento(s) desejado(s).

Esta, naturalmente, não é a única forma de se chegar aos elementos em questão, mas é a mais utilizada aqui. Sinta-se à vontade para experimentar outros seletores simples.

extraindo as informações

Antes de tudo, lembre-se de que foram colocados na variável *$artigos* todos os elementos <li> e o que eles contêm. Assim, é possível utilizar os métodos auxiliares do Mojo::DOM para obter, de cada um deles, o título, a url e o autor:

```
foreach my $artigo (@$artigos) {
        my $titulo = $artigo->at('a')->text;
        my $url    = $artigo->at('a')->attrs->{'href'};
        my $autor  = $artigo->at('span')->text;

        # vamos retirar os "[" e "]" dos nomes dos autores,
        # e aproveitar para eliminar espaços desnecessários.
        $autor =~ s/\s*[\[\]]//g;
```

```
        # agora vamos exibir o que encontramos
        print "$titulo ($url) '$autor'\n";
}
```

A primeira linha do laço acima diz para se colocar em *$titulo* o texto que estiver dentro da *tag* <a>. Em outras palavras, quando se tem algo como:

```
<a href="link">isso é um texto</a>
```

A chamada *at('a')->text* retornará "isso é um texto".

Para obter a URL, é necessário o attributo "href" desta *tag*, e assim se escreve *at('a')->attrs->{'href'}*.

Por fim, como o autor está envolvido em uma *tag* , utiliza-se *->text*, tal qual foi feito para o título.

O código completo de *crawler* fica:

```
use strict;
use warnings;
use Mojo::UserAgent;

my $client = Mojo::UserAgent->new->get( 'http://sao-paulo.pm.org/
artigos' );
my $dom = $client->res->dom;
my $artigos = $dom->find( 'div[class="whois"] > ul > li' );

foreach my $artigo (@$artigos) {
        my $titulo = $artigo->at('a')->text;
```

```
        my $url = $artigo->at('a')->attrs->{'href'};
        my $autor = $artigo->at('span')->text;
        # vamos retirar os "[" e "]" dos nomes dos autores,
        # e aproveitar para eliminar espaços desnecessários.
        $autor =~ s/\s*[\[\]]//g;

        # agora vamos exibir o que encontramos
        print "$titulo ($url) '$autor'\n";
}
```

pulando etapas

O leitor deve ter reparado que foram feitas algumas chamadas encadeadas, como *->new->get* e *->res->dom*. De fato, Mojo::UserAgent e Mojo::DOM retornam sempre o próprio objeto, de modo que se pode encadear todas as chamadas. Mais ainda, o método *find()* possui seus próprios iteradores para que não se precise fazer o *foreach* manualmente. Assim, o mesmo código acima poderia ser escrito desta forma:

```
use strict;
use warnings;
use Mojo::UserAgent;

Mojo::UserAgent->new->get( 'http://sao-paulo.pm.org/artigos' )
        ->res->dom->find( 'div[class="whois"] > ul > li' )
        ->each( sub {
```

```
    my $artigo = shift;
    my $titulo = $artigo->at('a')->text;
    my $url = $artigo->at('a')->attrs->{'href'};
    my $autor = $artigo->at('span')->text;

    # vamos retirar os "[" e "]" dos nomes dos autores,
    # e aproveitar para eliminar espaços desnecessários.
    $autor =~ s/\s*[\[\]]//g;

    # agora vamos exibir o que encontramos
    print "$titulo ($url) '$autor'\n";
});
```

Existe sempre mais de uma maneira de se fazer as coisas.

o desafio: transformando os dados em um *feed* rss/atom

Quando se fala de dados abertos, pensa-se em dados facilmente disponíveis e em formato livre. Suponha que o *scrapping* deste *site* tenha sido feito justamente porque foi notado que ele não oferece um *feed* listando os artigos disponíveis.

O *web crawler* já está pronto, mas apenas imprime-se o conteúdo na tela. O desafio é, portanto, transformar os dados obtidos em um *feed* Atom.

Mais uma vez, a solução é o CPAN, com o XML::Atom::SimpleFeed.

O processo é bem simples. Primeiro, cria-se o objeto de *feed* no início do programa. Depois, para cada artigo encontrado, adiciona-se uma nova entrada. Ao terminar, imprime-se todo o *feed*.

Vale notar que *feeds* Atom precisam de um identificador único, que não pode ser alterado. Para criar um no formato padrão, utiliza-se um UUID gerado pelo módulo Data::UUID a partir de um *namespace* ('sao-paulo.pm.org') e um nome ('artigo').

É preciso apenas colocar os trechos relacionados ao *feed*, sem modificar em nada o *crawler*. Foi colocado um "+" no início de cada linha nova para que as modificações fiquem mais visíveis:

```
use strict;
use warnings;
use Mojo::UserAgent;
+ use XML::Atom::SimpleFeed;
+ use Data::UUID;

+ # criamos nosso feed Atom
+ my $feed = XML::Atom::SimpleFeed->new(
    +     title => 'Artigos Publicados na SPPM',
    +     id    => 'urn:uuid:' . Data::UUID->new->create_from_name_
    str('sao-paulo.pm.org', 'artigos'),
+ );

# crawling pelo site da SPPM

Mojo::UserAgent->new->get(      'http://sao-paulo.pm.org/artigos'
)->res
```

```
    ->dom->find( 'div[class="whois"] > ul > li' )
    ->each( sub {
        my $artigo = shift;
        my $titulo = $artigo->at('a')->text;
        my $url = $artigo->at('a')->attrs->{'href'};
        my $autor = $artigo->at('span')->text;

        # vamos retirar os "[" e "]" dos nomes dos autores,
        # e aproveitar para eliminar espaços desnecessários.
        $autor =~ s/\s*[\[\]]//g;

        + # agora vamos adicionar o que encontramos
        + $feed->add_entry(
        + 'author' => $autor,
          +                 'title'  => $titulo,
          +                 'link'   => $url,
    +);
});

# agora que temos os artigos no feed,
# podemos imprimir
+ $feed->print;
```

resumo

Foi criado um *webcrawler* que extrai artigos de um *site* e gera um *feed* Atom completo, com direito a UUID – tudo isso em apenas 36 linhas, incluindo linhas em branco, comentários e formatação. Imagine agora o que você também pode fazer!

seletores css

seletores simples

Qualquer elemento

```
'*'
E
```

Um elemento E qualquer do tipo especificado – title, a, head, div, span, tr, ...

```
 'td'
E[foo]
```

Um elemento E qualquer com um atributo *foo* qualquer. Por exemplo, "a[alt]" retorna os elementos "<a>" que possuem o atributo "alt".

```
'a[alt]'
E[foo="bar"]
```

Um elemento E qualquer com um atributo *foo* possuindo valor exatamente igual a *bar*.

```
'input[class="obrigatorio"]'
E[foo~="bar"]
```

Um elemento E cujo atributo *foo* é uma lista de valores separados por espaço, e um desses valores é exatamente igual a *bar*.

```
'input[class~="obrigatorio"]'
E[foo^="bar"]
```

Um elemento E cujo atributo *foo* começa exatamente com a *string bar*.

```
'input[name^="obrig"]'
E[foo$="bar"]
```

Um elemento E cujo atributo *foo* termina exatamente com a *string bar*.

```
'input[name$="orio"]'
E[foo*="bar"]
```

Um elemento E cujo atributo *foo* contém a *substring bar*.

```
'input[name*="gato"]'
E[foo=bar][bar=baz]
```

Elemento E cujos atributos casam com o que for especificado – seguindo as mesmas regras de busca por atributo definidas acima.

```
'a[foo^="obrig"][foo$="ado"]'
```

E F

Um elemento F descendente de um elemento E.

```
'div h1'
```

E > F

Um elemento F filho de um elemento E.

```
'html > body > div > h1'
```

E + F

Um elemento F precedido imediatamente de um elemento E.

```
'h1 + h2'
```

E ~ F

Um elemento F precedido, em algum momento, de um elemento E.

```
'h1 ~ h2'
```

E, F, G

Elementos do tipo E, F e G.

```
'h1, h2, h3'
```

E:root

O elemento E raiz do documento. Em HTML4, é sempre o elemento <html>.

':root'

E:checked

Um elemento E marcado (ou ativado) no momento, como um *radio-button* ou uma *checkbox*.

':checked'

'input:checked'

E:empty

Um elemento E que não possui texto nem *subtags* ("filhos").

':empty'

'span:empty'

E:nth-child(n)

Um elemento E, n-ésimo filho de sua *tag* "pai".

```
'div:nth-child(3)'     # terceiro
'div:nth-child(odd)'   # impares
'div:nth-child(even)'  # pares
'div:nth-child(-n+3)'  # 3 primeiros
```

E:nth-last-child(n)

Um elemento E, n-ésimo filho de sua *tag* "pai", contando de trás para frente.

```
'div:nth-last-child(3)'     # terceiro
'div:nth-last-child(odd)'   # impares
'div:nth-last-child(even)'  # pares
'div:nth-last-child(-n+3)'  # 3 últimos
```

E:nth-of-type(n)

Um elemento E, o n-ésimo elemento ("irmão") do mesmo tipo.

```
'div:nth-of-type(3)'     # terceiro
'div:nth-of-type(odd)'   # impares
'div:nth-of-type(even)'  # pares
'div:nth-of-type(-n+3)'  # 3 primeiros
```

E:nth-last-of-type(n)

Um elemento E, o n-ésimo de seu mesmo tipo, contanto de trás para frente.

```
'div:nth-last-of-type(3)'     # terceiro
'div:nth-last-of-type(odd)'   # ímpares
'div:nth-last-of-type(even)'  # pares
'div:nth-last-of-type(-n+3)'  # 3 últimos
```

E:first-child

Um elemento E, primeiro filho de seu pai.

```
'div p:first-child'
```

E:last-child

Um elemento E, último filho de seu pai.

```
'div p:last-child'
```

E:first-of-type

Um elemento E, primeiro irmão de seu tipo.

```
'div p:first-of-type'
```

E:last-of-type

Um elemento E, último irmão de seu tipo.

```
my $last = $dom->at('div p:last-of-type');
```

E:only-child

Um elemento E, filho único de seu pai.

```
'div p:only-child'
```

E:only-of-type

Um elemento E, único irmão de seu tipo.

```
'div p:only-of-type'
```

E:not(s)

Um elemento E que não casa com o seletor *s*.

```
'div p:not(:first-child)'
```

introdução à reutilização de dados públicos

apresentação

Depois de extrair os dados, o passo seguinte é poder reutilizá-los de acordo com o interesse e a conveniência do usuário.

Existem diversas linguagens de programação no mercado bastante eficientes para desenvolver aplicações *web* de excelente qualidade. Não existe a melhor linguagem, embora haja fãs ardorosos (e algumas vezes combativos) de cada uma delas – e todas elas com comunidades ativas que ajudam na evolução da linguagem.

Cada usuário deve buscar aquela linguagem que lhe pareça mais fácil e a mais adequada ao projeto que deseja desenvolver. Não será incomum misturar tecnologias. Para ajudar, neste capítulo são apresentadas as três linguagens mais comumente utilizadas (Perl, Ruby e Phyton) e uma linguagem bem brasileira, LUA.

por que perl?

Este não é um artigo de "evangelização". A escolha de Perl 5 como linguagem principal (não única) para o projeto OpenData-BR[19] (saiba mais sobre o projeto no Apêndice 1 deste Manual) não foi movida por fanatismo. Foi uma escolha consciente. E neste artigo estão expostos os motivos que levaram o grupo responsável pelo projeto a essa decisão.

constante renovação

O primeiro ponto é que estamos usando o termo "Perl 5", ou seja, Perl moderno. O Perl é uma linguagem em constante renovação, que se orgulha de acompanhar de perto o que está sendo feito em outras linguagens de programação, sem medo de adotar o que funciona e descartar o que não dá certo.

Atualmente em sua 12ª versão, o Perl 5 é uma linguagem de uso geral, flexível e poderosa, que permite diferentes paradigmas de programação – orientada a objetos, funcional, procedural ou orientada a eventos. Ao adotar o Perl, não se fica preso a um único paradigma, e os colaboradores têm liberdade para escolher a forma que mais se adéque a seu problema.

Ao mesmo tempo, a preocupação do Perl com retro-compatibilidade é muito importante, permitindo que projetos como o OpenData-BR utilizem bibliotecas em versões atuais ou antigas do Perl,

[19] OpenData-BR é um projeto voluntário de um grupo com interesse em desenvolver material técnico em relação a dados abertos. Ver http://opendatabr.org.

sem nenhuma dificuldade ou necessidade de realizar modificações.

O Perl pode ser utilizado em mais de 100 plataformas diferentes, de dispositivos portáteis a *mainframes*, e sua poderosa infraestrutura de testes para módulos oferece ao OpenData-BR e seus subprojetos acesso imediato a uma detalhada matriz de compatibilidade entre versões, sistemas operacionais e arquiteturas, de modo que se pode facilmente verificar e garantir a qualidade do código disponibilizado.

redução dos riscos de negócio

O Perl é um *software* livre, e não depende de uma única empresa. Qualquer pessoa pode verificar seu código-fonte, enviar relatórios de erro, escrever correções ou compilar sua própria versão.

O Perl possui uma comunidade ativa de mais de 8 mil colaboradores em todo o mundo, sendo mais de 200 responsáveis diretamente no núcleo da linguagem. Novas versões do Perl são lançadas regularmente, uma por ano, com pelo menos duas atualizações de manutenção nesse intervalo, sem contar os lançamentos mensais da versão em desenvolvimento, permitindo testar seu código e identificar problemas antes de as versões terem sido lançadas oficialmente.

A estabilidade e a robustez do Perl são resultado de anos de maturidade e uma suíte de mais de 92 mil testes para a linguagem, sem contar os milhares de testes em seus diversos módulos auxiliares. O código interno do Perl foi certificado por ter baixa densidade de defeitos, e passou na validação de segurança do Department of Homeland Security dos Estados Unidos. O Perl possui ainda o modo "*taint*", que analisa e discrimina entradas

externas ao código – de usuários, arquivos ou via Internet – para evitar problemas mais comuns de segurança.

Tudo isso fez com que Perl fosse adotado para projetos de missão crítica em todo o mundo, de sistemas de telecomunicações e finanças a sítios na lista dos 100 mais acessados do planeta.

Por ser uma linguagem de alto nível, desenvolvedores desperdiçam menos tempo com problemas como gerenciamento de memória ou tipificação estática de variáveis, investindo mais tempo nos objetivos do *software* em desenvolvimento. Agilidade no desenvolvimento é fundamental para projetos em larga escala como o OpenData-BR, e o Perl provou ser uma ótima opção nesse quesito.

tratamento de informações

O Perl possui suporte nativo a Unicode, manipulação eficiente de *strings* e um dos mais poderosos sistemas de expressões regulares em uso na atualidade.

A linguagem oferece uma série de módulos para realizar *parsing* eficiente de conteúdo HTML, geração e interação com *web services* em JSON, XML ou YAML. Isso é fundamental para o sucesso de um projeto como o OpenData-BR, que lida exatamente com o tratamento de informações. Mesmo formatos mais exóticos conseguem ser tratados com Perl.

Mas coleta e separação de informações públicas são apenas metade da equação. Outra vantagem significativa está na formatação dos dados coletados para exibição e interpretação. Além de ser capaz de converter entre formatos facilmente, o Perl permite a geração de diversos gráficos estáticos e dinâmicos.

Outro desafio para o OpenData-BR é a comunicação entre sistemas e interfaces. Para isso, além da sua natural habilidade de conversão, processamento distribuído e interconectividade, o Perl oferece uma interface de integração de bancos de dados compatível com MySQL, Oracle, Sybase, PostgreSQL, SQLite e outros. É possível converter SQL entre diferentes sintaxes específicas de determinados bancos.

programa praticamente definitivo

Com mais de 22 mil distribuições e 90 mil módulos, e uma média acima de 200 *uploads* por mês, encontram-se no repositório CPAN todos os componentes necessários para reduzir o ciclo de desenvolvimento de *software*, permitindo que o desenvolvedor se concentre apenas na lógica do programa. Isso torna o desenvolvimento do OpenData-BR e seus subprojetos uma atividade rápida, gratificante e prazerosa para desenvolvedores de *software*.

facilidade para aprender baby-perl

dominar qualquer linguagem de programação é algo que requer anos de experiência prática. Mas em Perl a barreira de entrada é baixa, e isso é importante para projetos como o OpenData-BR.

Quem nunca programou antes, em poucas horas, pode escrever programas simples de coleta, manipulação e armazenamento de dados. Para quem já é proficiente em outra linguagem de programação, esse intervalo será de alguns minutos, apenas o tempo suficiente para se familiarizar com sua sintaxe.

comunidade vibrante e pronta para ajudar

Os "Perl Mongers"[20] possuem comunidades de desenvolvedores de *software* e entusiastas espalhados em diversos países, crescendo diariamente, com listas de discussão onde iniciantes podem solucionar suas dúvidas, comentar novos projetos, marcar encontros, participação em redes sociais, fóruns de tecnologia, eventos de *software* livre e mais.

Esse apoio comunitário é fundamental para a longevidade de projetos como o OpenData-BR, movidos pela própria comunidade.

conclusão

Perl não é melhor ou pior que qualquer outra linguagem, e pode não satisfazer às necessidades de seu projeto específico. Mas, para o OpenData-BR, as características acima indicadas, aliadas à enorme capacidade de integrar soluções em diferentes linguagens e interfaces, fazem do Perl moderno uma excelente escolha para projeto tão ambicioso e multifacetado.

[20] Página *web* da Comunidade Perl Mongers-SP: http://sao-paulo.pm.org/.

por que python?

A Python foi criada em 1991 por Guido Van Rossum. Ao longo dos anos, a linguagem evoluiu e atualmente possui um alto grau de maturidade, que pode ser observado nas inúmeras empresas que a utilizam. Um dos motivos desse sucesso é o aumento da produtividade que seu uso propicia aos projetos em que é aplicada.

Uma das principais virtudes é sua capacidade de funcionar como uma "*Glue language*", isto é, a Python possibilita uma fácil troca de informações entre sistemas heterogêneos, por meio de diversos módulos disponíveis que facilitam a interação com bancos de dados, com arquivos em diversos formatos (XML/HTML/RDF/CSV), com *web services* (JSON, REST, SOA, XMLRPC), além de permitir que se integrem módulos feitos em outras linguagens (como C).

A Python é uma linguagem que possui vasta biblioteca de funções, as quais permitem que se agilize o processo de desenvolvimento. Elas são muito úteis porque simplificam a implementação e acabam possibilitando ao programador, pela utilização de módulos confiáveis e testados, escrever menos linhas de código. Encontramos algumas bibliotecas[21] em Python que são utilizadas para facilitar a implementação de especificações, como o LinkedData.[22]

A Python não prende o desenvolvedor a um só padrão de código, e pode ser usada tanto para gerar um simples *script* como em

[21] Ferramentas Python para tratar de linked data:
http://www.readwriteweb.com/hack/2011/03/overview-of-python-tools-for-w.php
http://www.rdflib.net/
http://yergler.net/talks/pythonrdf/

[22] http://www.w3.org/wiki/LinkedData e http://www.w3.org/DesignIssues/LinkedData.html.

soluções mais elaboradas, que exijam padrões de orientação a objetos. Dessa forma, o conhecimento de Python possibilita ao programador utilizar esta linguagem para manipulações de dados simples, soluções rápidas ou para a criação de aplicações mais complexas.

Seu aprendizado é fácil, devido à sintaxe simples, clara e elegante. Possui um modo interativo que permite ao desenvolvedor realizar testes e validações diretamente com a máquina virtual Python por meio de linha de comando – esta capacidade possibilita ao iniciante explorar facilmente a linguagem e tornar-se produtivo em muito pouco tempo. Atualmente, muitas universidades têm utilizado Python em sala de aula como forma de diminuir a distância entre um problema proposto e sua solução, evitando assim que a própria linguagem seja um dos fatores de erro no estudo. Isso tem criado soluções mais elaboradas nos projetos de final de curso, melhorando sua qualidade.

A comunidade Python é muito forte. Possui uma lista de discussão no Brasil com mais de 2 mil membros, que trocam experiências e tiram dúvidas de maneira rápida e detalhada. Também existem grupos espalhados pelo país[23] que ajudam na divulgação e na promoção de debates e treinamentos. Para aumentar a troca de conhecimento, existe uma grande conferência anual (Conferência Python Brasil, http://www.pythonbrasil.org.br), com participações internacionais, em que se debatem novidades e se difunde o uso da linguagem. Para consolidar todas essas interações, existe um portal nacional (http://python.org.br) com uma enorme quantidade de informações em português, tais como dicas, trechos de códigos e o tutorial oficial da linguagem devidamente traduzido.

[23] Como http://pythonrio.org/ e http://www.python.org.br/wiki/GrupoDeUsuarios.

Há muitos *frameworks web* que simplificam a disponibilização das informações coletadas.[24] São ambientes seguros e muito flexíveis, que permitem, na maioria das vezes, ao desenvolvedor trabalhar de maneira intuitiva. Essas ferramentas também possuem comunidades que oferecem suporte ao seu uso.

A Python foi concebida como *software livre*. Por trás dela e de seus milhares de usuários no mundo, existe a Python Software Foundation (http://www.python.org/psf/), cujo objetivo é preservar livre a linguagem e promover seu uso e atualizações.

É uma linguagem fácil de ser encontrada, já que está instalada, nativamente, em todas as distribuições Linux e no MacOS, além de possuir executáveis que permitem sua utilização nos ambientes Windows e Android.

python em projetos dados abertos

A Python é a tecnologia fundamental de um dos mais proeminentes projetos de dados abertos do mundo: o CKAN (Comprehensive Knowledge Archive Network, http://ckan.org/), desenvolvido pela Open Knowledge Foundation, é basicamente um catálogo de dados abertos desenvolvido como um *web service*, codificado inteiramente em Python, com base no *framework* Pylons (http://pylonsproject.org/). Graças à flexibilidade da linguagem Python, o CKAN oferece uma interface única para interação com dados abertos, onde as informações são disponibilizadas para acesso manual ou automatizado e os metadados dos "pacotes de dados" são disponibilizados em múltiplos formatos, tais como JSON e

[24] Exemplos: https://www.djangoproject.com/ e http://web2py.com/.

RDF. Sendo completamente *open source*, a comunidade de catálogos locais CKAN está crescendo rapidamente, já tendo inclusive membros no Brasil (http://ckan.emap.fgv.br).

The Zen of Open Data

(disponível em http://sciblogs.co.nz/seeing-data/2010/10/12/the-zen-of-open-data/)

por Chris McDowall

Open is better than closed.
Transparent is better than opaque.
Simple is better than complex.
Accessible is better than inaccessible.
Sharing is better than hoarding.
Linked is more useful than isolated.
Fine grained is preferable to aggregated.
Although there are legitimate privacy and security limitations.
Optimise for machine readability – they can translate for humans.
Barriers prevent worthwhile things from happening.
Flawed but out there is a million times better than perfect but unattainable.
Opening data up to thousands of eyes makes the data better.
Iterate in response to demand.
There is no one true feed for all eternity – people need to maintain this stuff

Aberto é melhor do que fechado.

Transparente é melhor do que opaco.

Simples é melhor do que complexo.

Acessível é melhor do que inacessível.

Compartilhar é melhor do que acumular.

Conectado é mais útil do que isolado.

De granulação fina é preferível a agregado.

Apesar de haver privacidade legítima e limitações de segurança.

Otimize para leitura por máquinas – elas podem traduzir para humanos.

Barreiras impedem que coisas valiosas aconteçam.

Falho mas disponibilizado é um milhão de vezes melhor do que perfeito mas inalcançável.

Abrir dados para milhares de olhos melhora os dados.

Iterar em resposta à demanda.

Não há um único alimento para toda a eternidade – as pessoas precisam manter essa coisa.

Este texto foi inspirado nos aforismos do Zen of Python (http://www.python.org/dev/peps/pep-0020/), produzido por Tim Peters, que funciona como uma referência aos princípios da linguagem Python.

por que ruby?

A implementação de uma plataforma de *open data*, como defendida neste documento, não traz restrições quanto à tecnologia a ser usada em sua implementação. Quaisquer linguagens, *frameworks* e plataformas atualmente em utilização no mercado têm recursos suficientes para entregar o valor esperado.

Entre as principais linguagens utilizadas para desenvolvimento de aplicações *web*, em ordem especulativa de utilização, estão Java, C#, PHP, Python, Perl, Ruby. Na verdade, a Ruby é uma das linguagens menos utilizadas.

ruby on rails

A linguagem Ruby existe há mais de uma década, porém não foi antes do lançamento do *framework* para desenvolvimento *web* Ruby on Rails, em 2004, que o mercado passou a lhe dar atenção.

Uma das vantagens de ser uma comunidade formada muito recentemente é que ela já se iniciou com desenvolvedores de *software* experientes de todas as outras plataformas. Muito se aprendeu na primeira onda da Internet. Mais do que isso, em 2004 já se estava numa época pós-Agile, em que paradigmas de engenharia de *software* considerados muito experimentais finalmente ganhavam experimentação em projetos reais (por exemplo, metodologias e técnicas ditas ágeis, como desenvolvimento orientado primeiro a testes).

Entre os visionários que iniciaram o movimento de Ruby, junto com o Ruby on Rails, estavam diversos programadores experientes e reconhecidos no mundo da tecnologia e engenharia de *sof-*

tware, como Dave Thomas, Andy Hunt, Martin Fowler. Graças à influência dessas pessoas, o Ruby on Rails já nasceu com o que havia de mais moderno em paradigmas de desenvolvimento de aplicações web modernas e com as melhores práticas de engenharia de *software*.

Em pouco tempo, um ecossistema se formou ao seu redor, dando muita ênfase aos fatores qualidade e agilidade. Foram lançados grandes serviços, como o repositório público de código aberto, Github.

Centenas de ferramentas foram construídas, desde pequenas bibliotecas utilitárias até ambientes completos de desenvolvimento e infraestruturas inteiras de automatização de *data centers*. Mais do que isso, novas ondas de quebra de paradigma surgiram em paralelo, como a utilização crescente de bancos de dados não relacionais (conhecidos como NoSQL). Rapidamente, este ecossistema se estabeleceu como exemplo de desenvolvimento produtivo, atenção à qualidade e primor por boas práticas de engenharia de *software*, tudo isso aliado à utilização de tecnologias inovadoras (que eram experimentadas na prática por centenas de empresas *startup* do Vale do Silício e ao redor do mundo).

O repositório de bibliotecas do mundo Ruby chama-se RubyGems.org e, recentemente, ultrapassou a marca das 30 mil bibliotecas publicadas – número que compete até mesmo com a antiga e robusta CPAN do mundo Perl.

aceleração da evolução do mercado

sendo utilizada nos dias hoje por diversas empresas que efetivamente puxam o mercado adiante, como Amazon, Groupon e até mesmo a Nasa, além de empresas tradicionais, como JP Morgan,

Cisco ou IBM, a Ruby está nas mãos de todos os desenvolvedores de *software* que têm interesse em evoluir a área de engenharia de computação.

E a atual vantagem do mundo Ruby é justamente ela ser um nicho, a linguagem de menor mercado. Analistas do Gartner Group[25] acreditam que atualmente existam cerca de 1 milhão de programadores Ruby no mundo, com potencial para atingir 4 milhões até 2013. É menor, mas suficientemente grande para causar grandes inovações disruptivas, como o próprio lançamento do Ruby on Rails, ferramentas como Puppet e Chef – que administram centenas de servidores em grandes *data centers* –, *frameworks* avançados para lidar com bancos de dados modernos (como MongoDB ou Redis), além de trazer de volta técnicas outrora esquecidas de linguagens como Smalltalk ou Lisp, cujos paradigmas estão retornando ao uso no atual mundo de computação distribuída.

A Ruby não roda na maior parte das plataformas de *hardware* ou sistemas operacionais, mas isso também não é necessário, pois ela executa na maioria que está efetivamente em operação.

destaque no desenvolvimento *web*

Por causa do crescimento derivado com Ruby on Rails, a linguagem Ruby surgiu, em 2004, como incapaz de ser utilizada decentemente na *web* e transformou-se em uma das linguagens mais bem suportadas na rede em 2011. Hoje em dia, sistemas desenvolvidos em Ruby já foram instalados em diversos tipos de

[25] Disponível em http://blogs.gartner.com/mark_driver/2008/10/10/merb-10-for-ruby-emerges-real-competition-for-rails/

ambiente, suportando o tráfego de milhões de acessos por dia, processando quantidades massivas de dados ao redor do mundo, e é utilizada comercialmente nas mais diversas atividades, de *startups* a projetos de pesquisas.

Por causa disso, o ciclo de vida de uma aplicação *web* em Ruby é extremamente bem controlado, à escolha da equipe de desenvolvimento, com a maioria dos processos relevantes automatizados para aumentar produtividade e mitigar risco de erros humanos em instalação da aplicação e sua manutenção em máquinas de produção. Mais do que isso, existem dezenas de alternativas para monitoramento pró-ativo, instrumentação, análise e resolução de problemas e gargalos. Serviços como Hoptoad e New Relic ajudam a constantemente refinar os serviços oferecidos com decisões baseadas em dados.

Projetos de dados abertos são focados em disponibilização de conteúdo em ambiente *web*, coisa que o ecossistema Ruby tem dominado com disponibilidade de tecnologia robusta, que aguenta os ambientes mais pesados da Internet.

programa quase pronto

A comunidade Ruby nasceu de uma só vez, aproveitando tudo que fora aprendido até a metade da primeira década do século XXI. É a única comunidade que quase inteiramente adotou uma ferramenta conhecida como Git, utilizada em grandes projetos de código aberto, como o próprio núcleo Linux.

Esta ferramenta, aliada a serviços de repositórios públicos com o Github, criou uma dinamicidade ímpar na história do desenvolvimento de *softwares*, dando acesso irrestrito à atividade de colaboração de código para equipes voluntárias, geograficamente

distribuídas, sem quaisquer burocracias. Na velocidade das redes sociais de *web* 2.0, o uso massivo de Git fez com que um repositório de bibliotecas antes quase desconhecido, como o RubyGems.org, se modernizasse a ponto de ultrapassar repositórios tradicionais, como o CPAN do mundo Perl, atingindo mais de 30 mil *gems* publicadas, mais de 15 GB de espaço ocupado por essas *gems* e mais de 200 milhões de *downloads* em pouco mais de dois anos.

O ecossistema Ruby não só correu atrás do prejuízo, como também ultrapassou a maioria das outras comunidades em diversas áreas de tecnologia.

relativamente fácil

Dominar qualquer linguagem de programação é algo que requer anos de experiência prática. Como qualquer linguagem dinâmica moderna, a Ruby tem aparência simples para, porém a comunidade Ruby preza pela qualidade do conhecimento do desenvolvedor, e não existem versões "simplificadas". Não basta saber "digitar" código; é necessário saber colaborar com código de alta qualidade.

A forma como o ecossistema se formou, numa era pós-Ágil, em que a regra passou a ser escrever código de teste antes do código real, em que não se aceita refatorar código para conseguir melhor *design* ou mesmo desempenho, não pode existir colaboração em código aberto sem que se siga uma série de restrições de qualidade. Isso garante que a maioria das novas tecnologias sendo desenvolvidas em Ruby siga padrões elevados de boas práticas de engenharia, muito maior do que a média de qualquer outra plataforma, havendo ao mesmo tempo um equilíbrio para que não se construa uma comunidade orientada a "comissões" de burocratas.

A comunidade Ruby conseguiu evoluir unindo velocidade e qualidade, fatores considerados incompatíveis para a maioria dos desenvolvedores de *software* tradicional. E são justamente as duas características que projetos abertos deveriam almejar para conquistar crescimento sustentado de longo prazo, criando um ambiente adaptativo.

comunidade vibrante e empreendedora

Como deve ter ficado claro, a comunidade Ruby e o ecossistema de tecnologias e práticas geradas ao seu redor são os pontos fortes. Mais do que isso, a evolução rápida é aliada à vontade empreendedora particular desta comunidade.

O objetivo de qualquer indivíduo é produzir e tirar proveito dos resultados de sua produção. Ninguém trabalha de graça, nem deveria. E em uma comunidade cuja meta é construir novos negócios e colaborar com a tecnologia para se atingir esses objetivos, o resultado é um ecossistema robusto e inovador.

Plataformas para dados abertos visam oferecer informações que seus consumidores possam aproveitar para gerar mais valor. A comunidade Ruby tem experiência e capacidade para auxiliar em projetos com foco de mercado.

por que lua?

LUA é uma linguagem brasileira de programação interpretada, imperativa, de *script*, procedural, pequena, reflexiva e leve. Foi projetada para expandir aplicações em geral – por ser uma linguagem extensível (que une partes de um programa feitas em mais de uma linguagem) –, para prototipagem e para ser embarcada em *softwares* complexos, como jogos. Assemelha-se a Python, Ruby e Icon, entre outras.

A LUA foi criada por um time de desenvolvedores do Tecgraf, da Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ), a princípio para ser usada em um projeto da Petrobras. Devido a sua eficiência, clareza e facilidade de aprendizado, passou a ser aproveitada em diversos ramos da programação, como no desenvolvimento de jogos (a LucasArts, por exemplo, usou-a no jogo *Escape from Monkey Island*), controle de robôs, processamento de texto, etc. Também é frequentemente utilizada como uma linguagem de propósito geral.

A LUA combina programação procedural com poderosas construções para descrição de dados, baseadas em tabelas associativas e semântica extensível. É tipada dinamicamente, interpretada a partir de *bytecodes* e tem gerenciamento automático de memória com coleta de lixo. Essas características fazem da LUA uma linguagem ideal para configuração, automação (*scripting*) e prototipagem rápida.

história

A Lua foi criada em 1993 por Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes, membros da Computer Graphics Technology Group, da PUC-RJ. Versões anteriores à 5.0 foram liberadas sob uma licença similar à BSD. A partir da versão 5.0, LUA recebeu a licença MIT.

Alguns de seus parentes mais próximos são o Icon, por sua concepção, e a Python, por sua facilidade de utilização por não-programadores. Em artigo publicado no *Dr. Dobb's Journal*, os criadores da linguagem afirmaram que Lisp e Scheme também foram uma grande influência na decisão de desenvolver o quadro como a principal estrutura de dados de LUA.

LUA tem sido usada em várias aplicações, tanto comerciais como não comerciais.

Sendo uma linguagem extensível, foram geradas bibliotecas que podem tratar dados abertos de forma simples, como será demonstrado a seguir.

lua activerdf (http://activerdf.luaforge.net/)

Trata-se de uma biblioteca para acessar dados de arquivos RDF a partir de LUA, De fato, Lua ActiveRDF é uma versão de LUA para o ActiveRDF, construído para Ruby, que permite a criação rápida de semânticas para aplicações *web*.

Exemplo retirado do site http://activerdf.luaforge.net/ para criar e editar entradas de pessoas:

```
-- carrega-se activerdf
rdf = require 'activerdf'

-- adiciona-se um banco de dados SPARQL existente como uma base de dados
url = 'http://tecweb08.tecweb.inf.puc-rio.br:8890/sparql'
rdf.ConnectionPool.add_data_source { type = 'sparql', engine = 'virtuoso', url = url }

-- registra-se uma notação short-hand para o nome de campo usado nesses dados de teste
rdf.Namespace.register ( 'test', 'http://activerdf.luaforge.net/test/' )

-- agora pode-se acessar todas as propriedades RDF de uma pessoa como atributos Ruby:
john = rdf.RDFS.Resource.new 'http://activerdf.luaforge.net/test/john'
print ( john.test.age )
print ( john.test.eye )
table.foreach ( john.rdf.type, print )

-- agora é possível construir classes para as classes RDFs já existentes
rdf.ObjectManager.construct_classes()

-- e pode-se usar essas classes
bob = rdf.TEST.Person.new 'http://activerdf.luaforge.net/test/bob'

-- não se pode alterar nada, já que os resultados SPARQL têm acesso apenas para leitura
```

json4lua e jsonrpc4lua (http://json.luaforge.net/)

Essas bibliotecas implementam a codificação e decodificação do JSON (JavaScript Object Notation) e um cliente JSON-RPC-over-http para LUA.

JSON é uma simples codificação de objetos Javascript-like, que é ideal para uma transmissão leve de dados fracamente tipados. Um subpacote do JSON4Lua é o JSONRPC4Lua, um simples cliente e servidor JSON-RPC-over-http (em um ambiente CGILua) para LUA.

Exemplo de decodificação de uma estrutura codificada JSON, retornando um objeto em Lua com os dados apropriados:

```
json = require("json")
testString = [[ { "one":1 , "two":2, "primes":[2,3,5,7] } ]]
o = json.decode(testString)
table.foreach(o,print)
print ("Primes are:")
table.foreach(o.primes,print)
```

Resultado:

```
one             1
two             2
primes          table: 0032B928
Primes are:
1               2
2               3
3               5
4               7
```

processando xml com lua (http://www.keplerproject.org/luaexpat/)

O Projeto Kepler oferece LuaExpat, um módulo de interface com Expat, um processador XML.

Na verdade, o módulo oferece diretamente apenas uma função :lxp.new(), que retorna um objeto processado de XML (*parser*). Esta função recebe como parâmetro uma tabela especial de *callbacks*, que são as funções responsáveis por tratar os elementos XML.

Se for passado como parâmetro uma tabela vazia, o *parser* apenas verificará a integridade do código XML. Portanto, para usar LuaExpat para tratar XML, é preciso conhecer duas coisas: os *callbacks* e o *parser*.

callbacks

Na tabela de *callbacks*, as chaves devem possuir nomes específicos para indicar em que caso cada *callback* será usado. Os valores são funções: os *callbacks*.

Há uma variedade de *callbacks* (no momento em que este artigo foi escrito, havia quinze). Mas neste documento apenas os três principais serão vistos: StartElement, EndElement e CharacterData.

StartElement é chamado quando se encontra a abertura de um elemento (*tag*) (por exemplo, <xhtml:div id="main">). A função possui três argumentos: *parser*, *elementName* (nome do elemento) e *attributes* (atributos).

O primeiro argumento recebe o próprio *parser*. O segundo, *elementName,* recebe o nome do elemento (no exemplo, xhtml:div).

Attributes recebe uma tabela com os atributos, tanto de forma indexada quanto associativa. Assim, no exemplo (<xhtml:div id="main">):

```
{
[1] = "main";
id = "main"
}
```

EndElement é chamado quando se encontra o fechamento de um elemento (por exemplo, </xhtml:div>). A função possui dois argumentos: *parser* e *elementName* (nome do elemento).

Quando é encontrado um elemento simples (<elem>, </elem> ou <elem/>), são chamados o *callback* StartElement e, imediatamente, EndElement.

CharacterData é chamado quando se encontra uma *string* CDATA (conteúdo de um elemento). A função recebe dois argumentos: *parser* e *string* (o texto).

parser

O *parser* é criado pela função lxp.new(), que recebe a tabela de *callbacks* como parâmetro.

O *parser* possui diversos métodos, sendo os principais *parse()* (que processa uma *string* como parte do documento XML) e *close()* (método de chamada obrigatória que fecha o *parser*).

O método *parse()* deve ser chamado sem parâmetros para indicar o fim do documento.

A cada chamada de *parse()*, são retornados cinco valores:

1. *true*, se correu bem, ou *nil*, se ocorreu algum erro;
2. *nil*, se correu bem, ou uma mensagem de erro no caso de erro;
3. número da linha ou *nil;*
4. número da coluna ou *nil;*
5. posição absoluta ou *nil.*

A última chamada (sem parâmetros) retornará *true* se a estrutura XML estiver bem formada, ou *nil* e uma mensagem de erro se tiver ocorrido erro em algum momento.

Exemplo:

```
require "lxp"
local fd, l, st, erro
local contador = 0
local p = lxp.new {
        StartElement = function (self, nome, atributos)
        io.write("+ ", (" "):rep(contador), nome, "\n")
        contador = contador + 1
        end,
 EndElement = function (self, nome)
        contador = contador - 1
        io.write("- ", (" "):rep(contador), nome, "\n")
end,
CharacterData = function (self, texto)
        io.write("* ", (" "):rep(contador + 1), texto, "\n")
```

```
 end,
}
fd = io.input(argv[1])
for l in fd:lines() do
        p:parse(l .. "\n")
end
fd:close()
st, erro = p:parse()
p:close()
if not st then
        print("Ocorreu o seguinte erro:", erro)
end
```

Salve este código no arquivo "xmlparser.lua". Pegue um arquivo XML qualquer e execute:

```
bash$ lua xmlparser.lua nome-do-arquivo.xml
```

Se não houver nenhum disponível, use este:

```
<?xml version="1.0"?>
<elem1>
        texto
        <elem2/>
        mais texto
</elem1>
```

Drupal 7 e a *web* semântica

introdução

A gestão de conteúdo é um imenso desafio desde a época em que nossos antepassados transmitiam – de forma oral, por meio de histórias, contos e fábulas – a seus descendentes o que consideravam importante, passando depois à uniformização dos símbolos, que permitiram a memória escrita, e chegando até os dias de hoje, em que uma infinidade de informações (à qual nenhum ser humano conseguirá ter acesso durante seu curto tempo de vida) é gerada por uma infinidade de fontes. As técnicas de gestão de conteúdo, de fato, sempre correram atrás da geração de conteúdo. As bibliotecas são posteriores aos livros, e os mecanismos de busca são posteriores às páginas na *web*.

Em *A web semântica*, Tim Berners-Lee, que em 1989 havia proposto o que viria a ser a *web* da forma como a conhecemos hoje, alertava para a importância de contextualizar, para os vários dispositivos que fazem parte da Internet, os significados das informações, permitindo que ações fossem tomadas a partir desses significados. Ao menos dois conceitos seminais estavam expostos no artigo: interoperabilidade e significado. Tudo o que existe na *web* (ou na nuvem) tem a possibilidade de ser exposto de muitas formas, por intermédio de muitos dispositivos. Tudo o que existe na *web* deve ter um significado para algo ou alguém. Começaram-se a discutir, então, formas e padrões de organização que

estivessem intimamente ligados à informação na *web* desde seu nascimento (ou ao menos desde sua publicação na Internet).

Uma forma de organização é a RDF (*Resource Description Framework*, ou Estrutura para a descrição de ecursos). De forma simples, a RDF trata toda e qualquer peça de informação na *web* como um recurso, permitindo que se indique em uma determinada página, por exemplo, o que é o nome de uma pessoa, qual é a foto dessa pessoa, qual é o seu endereço de e-mail, etc. Essas informações podem ser até intuitivas para nós, humanos, quando navegamos por uma página pessoal de alguém, mas, para que programas (como mecanismos de busca) consigam identificá-las bem, elas devem estar descritas de maneira explícita. Além disso, a RDF permite estabelecer relações entre recursos que podem estar em locais distintos na *web*. Uma empresa pode ser um recurso associado a pessoas (também recursos) que nela trabalham.

Assim, usando a RDF, pode-se associar, por exemplo, o recurso "The Semantic Web" (um documento) aos recursos que o criaram (seus autores, pessoas) e aos recursos que financiaram a criação do documento (empresas ou instituições de ensino). Cada um desses recursos, descrito individualmente, pode estar em computadores e bases de dados distintas. De fato, a descrição dos recursos na *web* permite que ela seja tratada como uma enorme e única base de dados na qual se podem executar buscas de forma similar ao que seria feito em qualquer outra base de dados.

Ao se criar um portal *web*, expõe-se conteúdo ao mundo. A grande probabilidade é que esse conteúdo sequer seja encontrado a partir de um portal, mas, sim, por meio de mecanismos genéricos de busca. Há uma crescente probabilidade, também, que esse conteúdo não seja exibido da forma como é imaginado, mas que ela seja "importado" por visualizadores dos mais variados tipos, e não necessariamente por um navegador convencional. Assim,

se a vontade for que o conteúdo seja visualizado de todas as formas possíveis, é preciso estar atento a padrões de interoperabilidade e semântica. Felizmente, o sistema de gestão de conteúdo Drupal já expõe, no padrão RDF, a informação nele armazenada.

por que drupal?

O Drupal, projeto iniciado em 2000 por Dries Buytaert modestamente como um sistema de avisos para seus colegas da Universidade da Antuérpia, na Bélgica, evoluiu rapidamente para um sistema completo de gestão de conteúdo, que conta, atualmente, com cerca de 10 mil desenvolvedores que fazem mais de 2 mil contribuições semanais nas diversas partes do núcleo do sistema e módulos adicionais. Além disso, o Drupal sempre possibilitou a exposição do conteúdo nele gerado por meio de padrões, desde a simples exportação e importação de informações pelo uso de *feeds* RSS ou Atom até a adoção do padrão RDF de forma pioneira entre todos os sistemas abertos de gestão de conteúdo para a *web*.

A resposta à pergunta "Por que Drupal?" está intimamente conectada à resposta de "Por que PHP?". Por ter sido desenvolvido em PHP, o Drupal beneficia-se de tudo o que já foi desenvolvido nessa linguagem no que diz respeito a *web* semântica, como a interface programável RAP (Rdf Api for PHP) e outras que facilitam a criação de módulos para o Drupal por se beneficiarem da *web* semântica.

A instalação padrão do Drupal, a partir de sua versão 7, já ativa o módulo do núcleo do sistema que é específico para tratar a informação em um portal, expondo-a no formato RDF. Ou seja, sequer é preciso conhecer o RDF para beneficiar-se dele. Para

mais informações sobre a instalação do Drupal 7, visite o portal *Dicas-L* (ver referência ao final deste texto). Feito isso, serão vistas, no código-fonte das páginas geradas pelo Drupal 7, informações como essa:

```
<h1 property="dc:title" datatype="">Título da Página</h1>
<div class="submitted">
    <span property="dc:date dc:created" content="2011-07-21T20:37:
    29-03:00"
    datatype="xsd:dateTime" rel="sioc:has_creator">Enviado por
        <span class="username" xml:lang="" about="/user/1"
        typeof="sioc:UserAccount"
        property="foaf:name">Fulano de Tal
        </span>
      em qui, 07/21/2011 – 20:37
      </span>
</div>
```

Note que, misturados ao código HTML, há uma série de descritores para a informação contida na página.

FOAF

FOAF são descritores de pessoas (por exemplo: name, homepage, mbox, account, based_near). No exemplo, vê-se o criador do conteúdo:

```
<span class="username" xml:lang="" about="/user/1"
typeof="sioc:UserAccount"
property="foaf:name">Fulano de Tal
</span>
```

dublin core (dc)

DC são os descritores de publicação (por exemplo: abstract, created, dateCopyrighted, publisher). Reparem no exemplo:

```
<h1 property="dc:title" datatype="">Título da Página</h1>
<div class="submitted">
    <span property="dc:date dc:created" content="2011-07-21T20:37:
    29-03:00"
```

semantically-interlinked online communities (sioc)

SIOC são descritores relativos a redes sociais e seus usuários (por exemplo: follows, has_reply, last_reply_date, moderator_of, subscriber_of).

```
datatype="xsd:dateTime" rel="sioc:has_creator">Enviado por
<span class="username" xml:lang="" about="/user/1"
typeof="sioc:UserAccount"
property="foaf:name">Fulano de Tal
</span>
```

Com esses descritores, os mecanismos de busca poderão indexar melhor um portal *web*. Isso pode ser testado com a ferramenta Rich Snippets Testing Tool, do Google, no endereço http://www.google.com/webmasters/tools/richsnippets. Basta fornecer o endereço do portal com o Drupal 7 para observar os resultados[26].

ferramentas auxiliares

Conhecendo-se a linguagem SQL para a busca de informações em bases de dados, não se terá dificuldade com a linguagem SPARQL. Há vários aplicativos que implementam a busca com a linguagem SPARQL, mas um bastante fácil de instalar é o sparql--browser (http://code.google.com/p/sparql-browser/). Para executá-lo, é preciso antes ter o Adobe Air instalado. Os usuários das distribuições Linux baseadas no Debian (como o Ubuntu e outras) podem instalá-lo com o comando:

```
sudo apt-get install adobeair
```

[26] No momento em que este texto foi escrito, a ferramenta estava em estágio de testes, e, mesmo sendo útil para mostrar que o Drupal 7 expõe as informações no formato RDF, alguns erros eram exibidos.

Em seguida, basta baixar e executar o sparql-browser.

Outra ferramenta promissora é o sparql-query https://github.com/tialaramex/sparql-query), que permite que se façam consultas à web semântica de forma similar às consultas em uma base de dados SQL, por meio da linha de comando.

importando dados da *web* semântica para o drupal

O módulo SPARQL Views do Drupal (http://drupal.org/project/sparql_views) permite que se "importem" dados de outros locais na *web* que expõem suas informações no formato RDF. Um bom exemplo está em "The Semantic Web, Linked Data and Drupal, Part 2: Combine linked datasets with Drupal 7 and SPARQL Views".

conclusão

Mesmo que não se queira aprofundar-se muito em todos os aspectos técnicos que fazem parte da *web* semântica, ao instalar o Drupal 7 o usuário já passa a se beneficiar dela ao expor seus dados para pesquisa por mecanismos de buscas e outros sistemas que podem buscar as informações disponibilizadas.

referências bibliográficas

BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. The Semantic Web. Scientific American, maio de 2001. Disponível em http://kill.devc.at/system/files/scientific-american_0.pdf.

"The Semantic Web and Drupal", parte 1. Disponível em http://www.youtube.com/watch?v=xcPf4PeF57Y.

"The Semantic Web and Drupal", parte 2. Disponível em http://www.youtube.com/watch?v=VvEVoY0sl6o.

"Semana Drupal 7 no Dicas-L". Disponível em http://miud.in/QWk.

"The Semantic Web, Linked Data and Drupal, Part 1: Expose your data using RDF". Disponível em http://www.ibm.com/developerworks/library/wa-rdf/.

"The Semantic Web, Linked Data and Drupal, Part 2: Combine linked datasets with Drupal 7 and SPARQL Views". Disponível em http://www.ibm.com/developerworks/library/wa-datasets/.

"Potential RDF use cases for Drupal". Disponível em http://groups.drupal.org/node/9010.

Referências bibliográficas

Fontes de material para referências utilizadas no texto e nos artigos da comunidade PERL:

AGILE. Synergy Map, How to Map Out Your Current Strategy. Disponível em http://www.agileweboperations.com/synergy-map-how-to-map-out-your-current-strategy-part-1-of-2.

BARBUT, Marc *et al. Sémiologie Graphique: Les diagrammes, les réseaux, les cartes*. Paris: Gauthier-Villars.

BERNERS-LEE, Tim. *Design Issues: Linked Data*. Disponível em http://www.w3.org/DesignIssues/LinkedData.html.

FARRIS, Paul W.; BENDLE, Neil T.; PFEIFER, Phillip E. Métricas de marketing. São Paulo: Bookman, 2007.

PANOFSKY, Erwin. *Estudos de iconologia*. Lisboa: Estampa, 1995.

ROLF, Edward. *The Visual Display of Quantitative Information (pictures of numbers)*. Nova York: Tufte.

SEGARAN, Toby; HAMMERBACHER, Jeff. *Beautiful Data: The Stories Behind Elegant Data Solutions*. Nova York: O'Reilly Media, 2009.

W3.ORG. *Publishing Open Government Data*. Disponível em http://www.w3.org/TR/gov-dataa/.

_____. *Improving Access to Government through Better Use of the Web*. Disponível em http://www.w3.org/TR/egov-improving/.

_____. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. Disponível em http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

_____. *RDF Semantics*. Disponível em http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

_____. *RDF Primer*. Disponível em http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

_____. *Metadata Architecture*. Disponível em http://www.w3.org/DesignIssues/Metadata.

WORKING GROUP WIKI. *SPARQL*. Disponível em http://www.w3.org/2009/sparql/wiki/Main_Page.

Apêndice

Projeto OpenData-BR

Este movimento surgiu dentro de uma comunidade de *software* livre, a São Paulo Perl Mongers, que é um grupo de pessoas interessadas na linguagem Perl. Depois de um tempo discutindo sobre como manipular dados públicos, foi criado um grupo de desenvolvedores com interesse em trabalhar com este tipo de dado e desenvolver ferramentas para facilitar o seu uso.

O interesse é explorar os dados abertos para o bem da sociedade, seja no governo, instituições privadas ou universidades. Onde existirem informações, a ideia é trabalhar para torná-las acessíveis a favor de todos.

O objetivo é desenvolver ferramentas técnicas, documentação e auxiliar com o desenvolvimento de aplicativos para todos que queiram explorar os benefícios que a informação possa trazer.

Para contatos, há uma lista de discussão do grupo em http://groups.google.com/group/opendata-br. E também na IRC, na rede irc.perl.org, no canal #opendata-br

Na Internet:

Comunidade São Paulo Perl Mongers:
http://sao-paulo.pm.org

Projeto OpenData-BR:
http://opendatabr.org

Projetos:

Para onde foi o meu dinheiro:
http://opendatabr.org/index.php?title=Projetos:pofomd

Latitude e longitude de cidades brasileiras:
http://opendatabr.org/index.php?title=Projetos:llcb

Dataflow:
http://opendatabr.org/index.php?title=Projetos:Dataflow