Alternativas ao XML: YAML e JSON

Rúben Fonseca Alberto Simões
{rubenfonseca,ambs}@di.uminho.pt

Departamento de Informática
Universidade do Minho

Resumo O XML tem sido eleito como a linguagem de anotação por excelência, possuindo ao mesmo tempo boas capacidades para serialização de estruturas computacionais e transporte de dados independente da plataforma. Recentemente porém, novos formatos de dados têm surgido. Alguns deles têm tido uma boa aceitação porque resolvem alguns problemas ou limitações do XML, sendo em algumas situações um bom complemento ou substituto do mesmo. Neste artigo iremos apresentar dois desses formatos de dados - o YAML e o JSON - fazendo uma abordagem geral dos mesmos e analisando algumas métricas que nos poderão ajudar a decidir se e quando usar estas alternativas.
Palavras-chave: XML, YAML, JSON, serialização, anotação

1 Introdução

Nos últimos anos, o XML tem sido adoptado em áreas extremamente diversas: da justiça às finanças, dos hospitais às telecomunicações, da agricultura ao jornalismo. O XML tem sido a sintaxe de escolha para novos formatos de documentos na maior parte das aplicações de computador. É usado nas *mainframes* na bolsa de valores de *Wall Street* para trocar acções. Crianças em casa guardam o estado dos seus jogos de PC em XML. Fãs do futebol recebem resultados em directo no seu telemóvel usando XML. O XML é simplesmente a sintaxe de documentos mais robusta, fiável e flexível alguma vez inventada.

Significa isto que não necessitamos de mais nenhum formato de dados no mundo? Será que a investigação e o trabalho nesta área já atingiu a maturidade e pouco ou nada se poderá inventar? Será que o XML é mesmo a melhor solução para *todos* os problemas nesta área?

Dezenas de alternativas ao XML existem no mercado[Dum06]. Algumas delas não têm a massa crítica necessária para se tornarem conhecidas. Podiam ser estudados formatos como as *S-expressions* ou o *Data::Dumper*. No entanto, de entre as várias alternativas, duas têm-se destacado nos anos mais recentes: o YAML e o JSON. Ao longo deste artigo faremos uma análise comparativa de ambas estas linguagens, tentando analisar de várias perspectivas se elas se apresentam como uma alternativa ao XML, ou se apenas têm um universo mais reduzido de aplicação.

Antes de começar, analisaremos alguns problemas ou limitações bem conhecidos do XML.

2 O XML não é perfeito

O XML é um standard definido e recomendado pela W3C para anotação de documentos. São inegáveis as vantagens que o XML trouxe a todo o mundo da representação e transferência de dados ou documentos. A sua flexibilidade, robustez e relativa simplicidade permitiu uma rápida adopção por parte de todo o tipo de entidades. O seu predecessor - o SGML - já estava em uso desde 1986, o que fez com que já existisse extensiva experiência técnica e aplicações disponíveis.

No entanto, como tudo no mundo, o XML também tem os seus pontos fracos. De seguida mencionaremos apenas os mais comuns apontados.

- A sintaxe é simultaneamente palavrosa e redundante.
  Isto pode afectar a leitura humana, a eficiência das aplicações e obriga normalmente a custos de armazenamento mais elevados. Este característica torna o XML difícil de aplicar em casos em que a largura de banda é limitada, embora a compressão possa reduzir o problema em alguns casos[LS00]. Este problema é particularmente grave para aplicações multimédia em telemóveis e PDAs que usem XML para descrever imagens e vídeo.

- A construção dos parsers não é tão trivial como parece.
  Embora teoricamente o XML permite a construção de parsers simples e consistentes, um bom parser de XML têm de ser desenhado para processar dados arbitrariamente aninhados e efectuar testes adicionais para detectar dados mal formatados ou erros de sintaxe (devido, em parte, ao problema descrito no item anterior). Isto pode causar carga adicional significativa para os usos mais básicos do XML, particularmente nos sistemas embebidos[vE04]. Além disso, quando os recursos são limitados, problemas de segurança podem surgir se o XML provem de fontes não confiáveis, podendo levar à exaustão de recursos ou a *stack overflows* (e.g. XML composto por um número muitíssimo elevado de `<elemento>` aninhado).

- Os requisitos básicos de processamento não suportam um grande conjunto de tipos de dados.
  Por esta razão, a interpretação semântica envolve trabalho adicional para inferir os tipos de dados num documento. Por exemplo, não existe nenhum mecanismo em XML, para que `3.14159` seja um número de vírgula flutuante em vez de uma palavra de sete caracteres. As linguagens de *XML Schema* adicionam esta funcionalidade[CEM03].

- A sintaxe do XML é difícil de usar para humanos.
  A sequência de teclas que se usam para escrever expressões XML num teclado convencional de computador são particularmente difíceis de usar, dificultando a criação rápida de documentos anotados com XML. O uso de geradores ou de editores especializados para o efeito poderão diminuir o peso deste problema.

Quer se concorde com estes pontos fracos do XML ou não, será bom manter uma mente aberta e ver se existem alternativas que resolvam estes problemas ou que apenas complementem o XML em contextos mais específicos. Neste caso específico, até que ponto o YAML ou o JSON são bons candidatos a resolver estes problemas? Será que conseguem manter o mesmo nível de robustez e flexibilidade do XML? Como é o desempenho relativo dos parsers existentes? Valerá a pena tentar?

3 YAML

Talvez a primeira questão que surja é "porquê o nome YAML?". Existem um bom número de ferramentas que adoptaram acrónimos na forma `YA*`, significando "Yet Another XXX". Por isso dois nomes - aparentemente contraditórios - têm sido atribuídos para o acrónimo YAML: "Yet Another Markup Language" e o mais recursivo "YAML Ain't a Markup Language". No entanto, existe alguma lógica para isto: o YAML oferece grande parte do que as linguagens de anotação oferecem, mas com uma notação muito leve.

O YAML é um formato de serialização de dados legível por humanos, que se inspirou em conceitos de linguagens como o XML, C, Python, Perl e também o formato do correio electrónico especificado no RFC 2822. Foi proposto por Clark Evans em 2001 e desenhado em conjunto com Ingy döt Net e Oren Ben-Kiki[OBK05].

Embora não seja menos genérico que o XML, o YAML é em grande parte mais simples de ler, editar, modificar e produzir que o XML. Ou seja, quase tudo o que é possível de representar em XML pode ser representado em YAML, e ao mesmo tempo, de uma forma mais compacta.

O YAML foi definido para suportar apenas caracteres no sistema UTF8 ou UTF16, sendo o comportamento dos parsers indefinido quando a stream YAML está em qualquer outra codificação.

Os espaços de nomes são um exemplo de algo que a especificação corrente do YAML não suporta mas que podem ser mais ou menos adaptados em cima do mesmo.

3.1 Exemplo

O YAML foi criado com a crença de que todos os dados podem ser adequadamente representados por combinações de listas, mapas e registos. Uma lista é uma colecção ordenada de zero ou mais valores; um mapa é uma coleção de zero ou mais pares nome/valor; um registo é um tipo primário como uma string.

A sintaxe é relativamente simples e foi desenhada tendo em mente a legibilidade pelos humanos, mas ao mesmo tempo que permitia que os dados sejam facilmente transformados nas estruturas mais comuns das linguagens de alto nível.

Desta maneira, o YAML incide mais na representação limpa e compacta das estruturas de dados que se encontram em linguagens como Perl, Python, Ruby,

e com menos relevância, o Java. Existem bibliotecas YAML para quase todas as linguagens de programação.

Vejamos um exemplo concreto. Imagine-se o seguinte documento XML com a descrição de um clube de xadrez.

```
<?xml version="1.0"?>
<club>
  <players>
    <player id="kramnik"
            name="Vladimir Kramnik"
            rating="2700"
            status="GM" />
    <player id="fritz"
            name="Deep Fritz"
            rating="2700"
            status="Computer" />
    <player id="mertz"
            name="David Mertz"
            rating="1400"
            status="Amateur" />
  </players>
  <matches>
    <match>
        <Date>2002-10-04</Date>
        <White refid="fritz" />
        <Black refid="kramnik" />
        <Result>Draw</Result>
    </match>
    <match>
        <Date>2002-10-06</Date>
        <White refid="kramnik" />
        <Black refid="fritz" />
        <Result>White</Result>
    </match>
  </matches>
</club>
```

A representação acima é bastante clara. No entanto, sofre de alguns dos problemas mencionados na secção anterior quanto à verbosidade, fraca legibilidade e dificuldade na elaboração deste documento. Compare-se agora estes mesmos dados com a versão em YAML.

```
players:
  Vladimir Kramnik: &kramnik
    rating: 2700
    status: GM
  Deep Fritz: &fritz
    rating: 2700
    status: Computer
```

```
  David Mertz: &mertz
    rating: 1400
    status: Amateur

matches:
  -
    Date: 2002-10-04
    White: *fritz
    Black: *kramnik
    Result: Draw
  -
    Date: 2002-10-06
    White: *kramnik
    Black: *fritz
    Result: White
```

Há algumas coisas interessantes acerca deste formato. O site do YAML[OBK05] oferece a especificação exacta, mas este simples exemplo fornece uma ideia dos elementos básicos. A especificação também inclui maneiras intuitivas de incluir strings contendo parágrafos múltiplos. A necessidade de aspas é mínima, e os tipos de dados são inferidos a partir de padrões (por exemplo, se os dados se parecem com uma data são tratados como tal, a não ser que sejam protegidos com aspas explicitamente).

Pode-se usar referências para todos os nomes (notar o operador `&` e `*`). Na realidade, todas as entidades podem ser referenciadas em qualquer parte do texto. Com isto podemos obter e usar estruturas partilhadas para serializar a informação de maneira mais lógica e eficiente do que no XML. O YAML mantém a distinção entre colecções ordenadas e associadas. Como bónus final, qualquer um pode editar YAML num editor de texto convencional.

Uma das grandes vantagens do uso de YAML não está relacionado com o seu poder sintáctico. As interfaces uniformes em todas as linguagens suportadas são um grande trunfo para o YAML. Por exemplo, para ler e poder manipular os dados do exemplo anterior em Perl seria tão simples como:

```
use YAML ();
my $club = YAML::LoadFile('club.yml');
my $club_yamlstr = YAML::Dump($club);
```

Mas como se comportará o YAML para realizar tarefas de anotação de documentos?

3.2 Anotação de documentos

O YAML não foi construído com a anotação de documentos em mente. Embora devido à sua estrutura simples e flexível seja possível realizar algum trabalho nesta área, o YAML não se aproxima nem de perto ao poder que o XML

tem para anotar documentos. Talvez um exemplo ajude a entender este ponto. Considere-se o seguinte excerto XML:

```
<paragraph>
    O artigo <artigo id="1253">XML é fixe</artigo> é bom.
</paragraph>
```

A mistura de elementos com o texto é algo que dificilmente é conseguida no YAML. Este facto dificulta imenso o uso do YAML nesta área - na realidade nunca foi o seu objectivo. No entanto, o YAML trás uma vantagem: por definição não é possível escrever um documento mal formado por misturar *overlapping tags* (e.g. `<b><p></b></p>`).

Pelas razões apresentadas, o YAML não é um substituto, nem sequer um concorrente com o XML para anotação de documentos.

3.3 Serialização de dados

É nesta área que o YAML mais brilha. As suas características permitem facilmente representar os tipos de dados mais comuns das grandes linguagens de programação. A própria especificação YAML defende que todos os tipos de dados se podem representar à custa de combinações de listas, mapas e registos[OBK05].

Ao possuir uma sintaxe limpa, torna a serialização de dados mais legível à inspeção humana. No entanto, sem perder a generalidade, o YAML apresenta maneiras alternativas de expressar o mesmo tipo de dados. Por exemplo:

```
--- # Filmes favoritos, formato bloco
- Casablanca
- Spellbound
- Notorious
--- # Lista de compras, formato inline
[milk, bread, eggs]
```

```
--- # Bloco
name: John Smith
age: 33
--- # Inline
{name: John Smith, age: 33}
```

Um outro trunfo do YAML é que, as representações suportadas (listas, mapas e registos) correspondem a tipos nativos em todas as linguagens dinâmicas. Isto permite não gastar muito tempo a transformar uma stream YAML numa estrutura de dados a ser consumida pela aplicação. Os programadores destas linguagens sentem-se imediatamente confortáveis tanto na importação de dados em YAML, como na sua exportação ou criação.

Talvez por este facto, o YAML foi escolhido como formato de serialização por omissão na linguagem de scripting Ruby. É também utilizado cada vez mais para

escrever ficheiros de configuração, já que permite, com uma sintaxe limpa, simples e intuitiva, escrever estruturas de configuração arbitrariamente complexas que podem ser carregadas para qualquer linguagem de uma forma completamente uniforme.

3.4 Análise de desempenho do parser

Para ter uma ideia de como o YAML se comporta face ao XML efectuou-se um teste comparativo de desempenho. As condições foram as seguintes:

- Ficheiros de gerados automáticamente, com n elementos de texto aleatório
- Parser de XML: libxml2[1] (2.6.26) + XML::LibXML[2] (1.62)
- Parser de YAML: libsyck[3] (0.55) + YAML::Syck[4] (0.72)
- Computador pessoal Intel Core Duo T2300 1.66Ghz, 1GB RAM
- Linux 2.6.18.3

Nenhum dos parsers usados tira partido do multi processamento disponível na plataforma.

Os elementos aleatórios têm a seguinte forma.

```
<contacts>
    <person>
        <number>855904</number>
        <name>sdfasdfasdfasdfasdffasdfa</name>
        <place>aw da dk dfw  awfhkda hlhwhdfah jadfkw hw</place>
        <date>51 53 9 30 10 106 4 333 0</date>
    </person>
    (...)
</contacts>
```

Depois de gerar ficheiros de vários tamanhos, estes foram submetidos aos respectivos parsers (escritos em Perl, usando as bibliotecas C), e os tempos de parsing foram anotados. Os resultados obtidos estão nas figuras 1 e 2.

Como se pode observar, o YAML é ligeiramente mais eficiente quando o input é mais pequeno. No entanto não escala convenientemente com o tamanho do input, ou pelo menos não tão eficientemente como o parser de XML escala. Podemos concluir que o YAML é mais apropriado para pequenas e médias serializações. Isto explica porque é que o YAML é usado mais frequentemente para serialização de objectos e formato de ficheiros de configuração, onde o tamanho do ficheiro não é significativo para a perda de desempenho.

O parser de YAML é mais eficiente ao nível da memória, necessitando de menos memória para processar a mesma entrada. O ponto de comutação acontece

[1] http://xmlsoft.org/

[2] http://search.cpan.org/~pajas/XML-LibXML-1.62001/

[3] http://yaml.kwiki.org/index.cgi?LibSyck

[4] http://search.cpan.org/~audreyt/YAML-Syck-0.72/

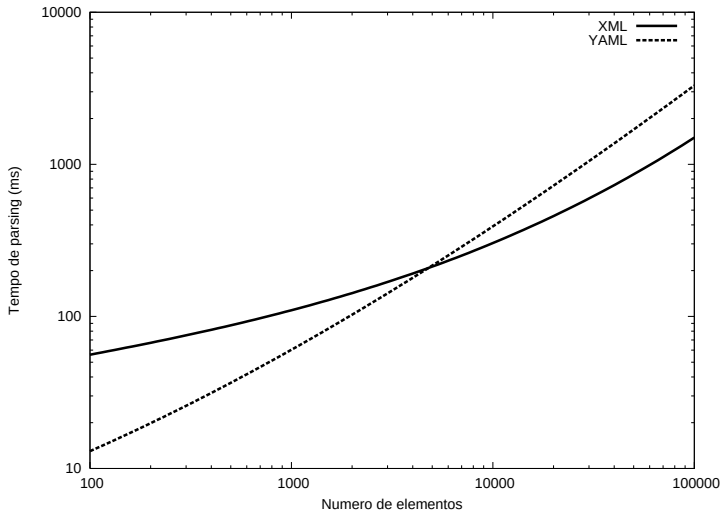

Figura 1. Tempo de execução em função do número de elementos

neste caso quando a entrada atinge o 1Mb. A partir deste ponto, o parser de XML libxml2 começa a ser mais eficiente no tempo de parsing, gastando no entando sempre mais memória do que o parser de YAML syck.

No entanto, estes resultados deverão ter em conta que o parser de XML libxml2 é um parser resultante de muitos anos de trabalho de uma comunidade numerosa, e por isso produz resultados supreendentemente eficientes quando comparados com outros parsers conhecidos, tanto ao nível de tempo de parsing como no consumo de memória[Chi04]. Por outro lado, o parser Syck é novo e trabalho de poucas pessoas, com muita margem para evoluir em termos de funcionalidades e desempenho.

4 JSON

O JSON (pronunciado como a palavra inglês *Jason*) é um acrónimo para "JavaScript Object Notation", e é um formato de texto para a serialização de dados estruturados. É derivado dos *object literals* do JavaScript, conforme definido no standard ECMAScript[Int99].

O JSON pode representar quatro tipos primários (strings, números, booleanos e nulos) e dois tipos estruturados (objectos e vectores). Um objecto é uma colecção não ordenada de zero ou mais pares nome/valor, onde o nome é uma string e o valor é uma string, número, booleano, nulo, objecto ou vector. Um vector é uma sequência ordenada de zero ou mais valores[Cro06].

Assim o JSON foi desenhado com o objectivo de ser simples, portável, textual, e um subconjunto do JavaScript. Iremos ver mais à frente o que isto pode significar no contexto da aplicação em serialização de dados na Web.

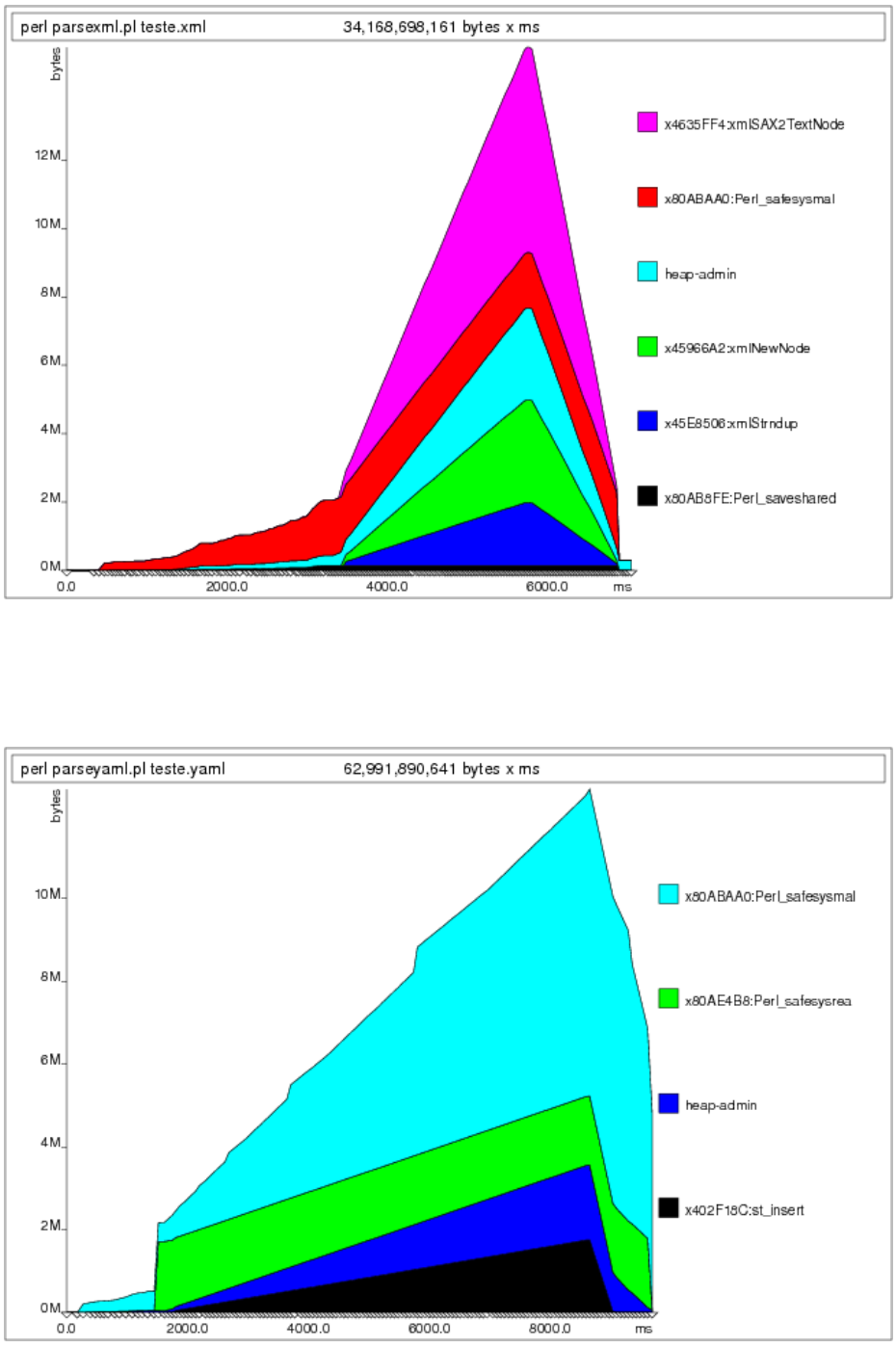

Figura 2. Consumo de memória do parser de XML e YAML para entradas de 10000 elementos

Assim como o YAML, o JSON apenas suporta texto em codificação UTF8 ou UTF16.

4.1 Exemplo

As similaridades de sintaxe entre o YAML e o JSON são mais do que à partida se pode imaginar[wtls05]. Atente-se ao seguinte exemplo:

```
{
    "type": "menu",
    "value": "File",
    "items": [
        {"value": "New", "action": "CreateNewDoc"},
        {"value": "Open", "action": "OpenDoc"},
        {"value": "Close", "action": "CloseDoc"}
    ]
}
```

Apesar de esta representação aparentar isso, a indentação e o uso de espaços em branco é totalmente opcional no JSON. O mesmo exemplo poderia ser escrito apenas num linha sem que isso resultasse em erro no compilador.

Um olhar atento poderá concluir correctamente que o JSON é praticamente um subconjunto funcional do YAML. Na realidade a maior parte dos parsers de YAML conseguem lidar com grande parte do JSON, já que este último partilha grande parte das características do YAML em modo *inline*. Este facto é mera coincidência e não foi decisão aquando do desenho do formato. O YAML e o JSON foram concebidos isoladamente e por equipas diferentes.

Devido à sua simplicidade, o JSON não suporta referências para elementos, e obriga ao uso de aspas para se manter compatível com a sintaxe do JavaScript.

4.2 Anotação de documentos

Sendo um subconjunto funcional do YAML, o JSON apresenta praticamente o mesmo poder expressivo e capacidade de anotação (quase nula) do YAML. Também este formato não foi desenhado com a anotação de documentos em mente, e por isso, normalmente não substitui o XML para a realização deste tipo de tarefas.

4.3 Transporte de dados

Como subconjunto do YAML, também aqui o JSON brilha como formato de serialização. Além de possuir a maior parte das vantagens do YAML nesta área, o JSON possui um trunfo fortíssimo que fez com que fosse aceite rapidamente pela comunidade e usado extensivamente: o JSON é um subconjunto do "JavaScript Object Notation". O que isto significa? Considere-se o seguinte código JavaScript no contexto de uma aplicação web que acaba de fazer uma invocação remota.

```
1 the_object = eval("(" + http_request.responseText + ")");
```

No contexto de aplicações que usem invocações remotas na Web (e já que o AJAX[Gar05] se considera indispensável para a construção de um novo site bem sucedido) o JSON é um recurso extremamente útil, visto que permite transportar um objecto serializado arbitrariamente complexo, e transformá-lo num objecto JavaScript com um simples *eval*. Temos assim um parser poderosíssimo de JSON incluido em todos os grandes browsers de internet existentes do mercado.

Este facto faz com que o JSON esteja cada vez mais a ser usado para transporte de dados serializados no contexto da internet, em detrimento do XML. Embora todos os browsers mais usados suportem correctamente o o processamento de XML, ainda nem todos possuem um bom e completo processador e interrogador de XML, incorporado numa linguagem dinâmica embutida, o que faz com que o seu uso aumente a complexidade das aplicações Web que dele fazem uso. O JSON é assim, neste contexto, um óptimo substituto do XML.

Mas como se comporta o JSON em termos de desempenho?

4.4 Análise de desempenho

Realizamos também um teste de desempenho ao JSON, exactamente nas mesmas condições que o teste da secção 3.4. Para tal foi usado:

- Parser de JSON: libsyck (0.55) + JSON::Syck[5] (0.72)

O leitor atento já percebeu que o parser de JSON é o mesmo que foi usado o parsing de YAML. Isto deverá reforçar a ideia passada na secção 4.1 onde concluímos que o JSON e o YAML partilham (por coincidência) grande parte da sintaxe. Logo o parser Syck consegue atacar ambas as linguagens de uma maneira uniforme. Os resultados no entanto são interessantes, conforme pode ser observado na figuras 3 e 4.

Como se pode verificar, também o JSON não escala tão bem como o XML quando o tamanho do input aumenta significativamente. No entanto, o JSON escala melhor do que o parser de YAML. A explicação poderá vir do facto da especificação do JSON ser menos complexa, sendo assim um formato menos flexível, mas mais simples e fácil de processar. O consumo de memória durante o parsing é idêntico ao do YAML (e melhor que o de XML), talvez resultado do uso da mesma biblioteca para o parser. No contexto de aplicações Web (onde o JSON é mais usado), podemos concluir que o JSON é uma boa aposta pois além das suas vantagens funcionais, é ligeiramente mais rápido que o XML para inputs de pequena ou média dimensão (no nosso teste, até 1Mb de entrada).

[5] `http://search.cpan.org/~audreyt/YAML-Syck-0.72/`

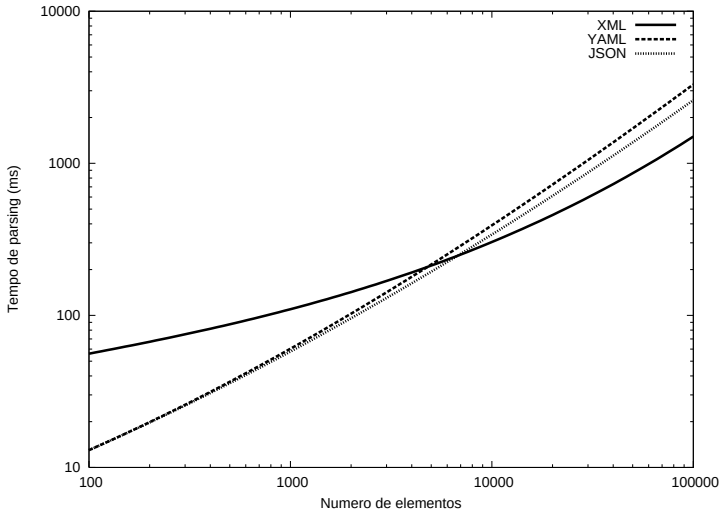

Figura 3. Tempo de execução em função do número de elementos

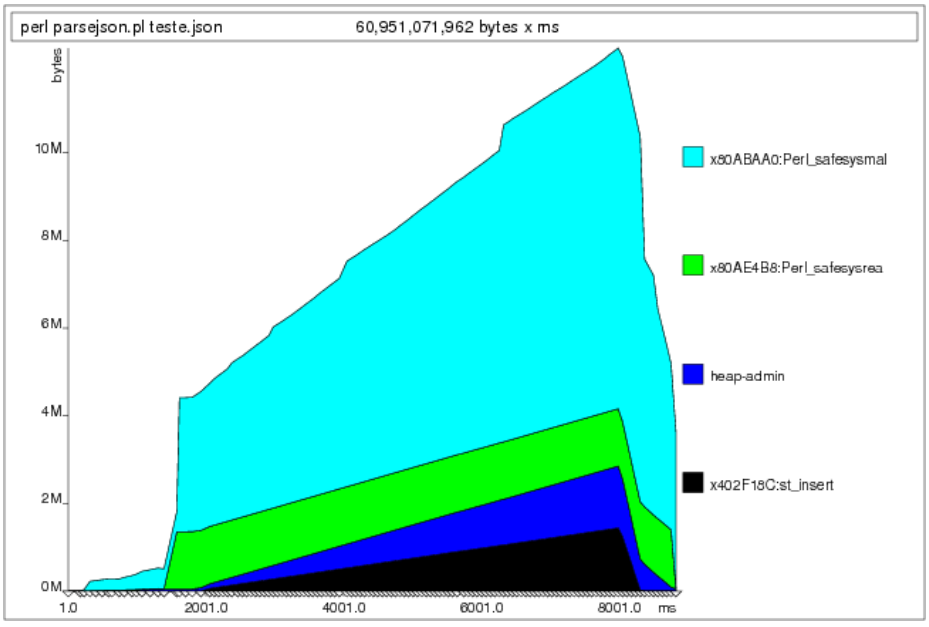

Figura 4. Memória consumida pelo parser de JSON para input de 10000 elementos

5 Conclusão

Embora o XML conte com uma enorme comunidade de utilizadores, não é perfeito nem a única alternativa para todos os problemas de anotação e serialização. Alguns desses problemas foram estudados e atacados por outros formatos como o YAML e o JSON.

Apesar de não competirem com o XML como ferramenta para anotação de documentos, tanto o YAML como o JSON demonstram boas capacidades no que toca à serialização de dados. O YAML apresenta uma sintaxe limpa, fácil de ler e editar, e muito bem estruturada. O JSON, sem grande perda de generalidade, coloca um pouco mais de açúcar na sua sintaxe de modo a torná-lo verdadeiramente um subconjunto do JavaScript.

Devido às suas características, tanto o YAML como o JSON permitem usar parsers simples, e serem transformados transparentemente para os tipos de dados nativos das grandes linguagens de scripting. Ao mesmo tempo são uniformes (i.e. usam-se sempre da mesma maneira nas diferentes linguagens suportadas). O JSON possui adicionalmente um forte trunfo: existem bons parsers (e fáceis de invocar), completamente funcionais em todos os grandes browsers em utilização actual, o que permite o uso eficiente de serializações para na construção de aplicações Web interactivas.

A especificação do YAML é mais complexa do que apresentado. O YAML suporta ainda vários documentos no mesmo ficheiro, atalhos, tipos de dados forçados, blocos e aliases. O YAML Cookbook é uma boa fonte para aprender mais detalhes sobre estes[OBK06].

É certo que não são substitutos, mas algumas das tecnologias que assentam sobre XML podem ser mais facilmente implementadas com YAML ou JSON. É necessário XML para RPC? Será necessário XML para RDF? É necessário usar XML para tudo? Não. É o YAML ou o JSON um substituto perfeito? Não, mas em algumas circunstâncias poderá ser mais útil usá-los em detrimento do XML. Todos eles são excelentes standards, e todos têm o seu lugar.

Referências

[CEM03] Charles E. Campbell, Andrew Eisenberg, and Jim Melton. Xml schema. *SIGMOD Rec.*, 32(2):96–101, 2003.

[Chi04] Suren A. Chilingaryan. Xml benchmark. `http://xmlbench.sourceforge.net/`, 2004. Provides benchmarking toolset for all available multiplatform C/C++ (and some Java) XML parsers.

[Cro06] D. Crockford. Json specification. `http://www.ietf.org/rfc/rfc4627.txt`, 2006.

[Dum06] Edd Dumbill. Exploring alternative syntaxes for xml. `http://www-128.ibm.com/developerworks/xml/library/x-syntax.html`, 2006. Artigo sobre alternativas ao XML.

[Gar05] Jesse James Garrett. Ajax: A new approach to web applications, 2005. `http://adativepath.com/publications/essays/archives/000385.php`.

[Int99] Ecma International. Standard ecma-262. `http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf`, 1999. ECMAScript Language Specification.

[LS00] Hartmut Liefke and Dan Suciu. Xmill: an efficient compressor for xml data. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 153–164, New York, NY, USA, 2000. ACM Press.

[OBK05] Brian Ingerson Oren Ben-Kiki, Clark Evans. Yaml specification. `http://yaml.org/spec/`, 2005.

[OBK06] Brian Ingerson Oren Ben-Kiki, Clark Evans. Yaml cookbook. `http://yaml4r.sourceforge.net/cookbook/`, 2006. Conjunto de exemplos do uso extensivo de YAML.

[vE04] Robert van Engelen. Code generation techniques for developing light-weight xml web services for embedded devices. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 854–861, New York, NY, USA, 2004. ACM Press.

[wtls05] why the lucky stiff. Yaml is json. `http://redhanded.hobix.com/inspect/yamlIsJson.html`, 2005. Artigo que compara as semelhanças entre YAML e JSON.