Resumo

Actualmente a utilização de linguagens de *markup* é recorrente no mundo da tecnologia, sendo o HTML a linguagem mais utilizada graças à sua utilização no mundo da Web. Tendo isso em conta é necessário que existam ferramentas capazes de escrever documentos bem formados de forma eficaz. No entanto, a abordagem mais utilizada, *template engines*, tem cinco problemas principais:

- Não garante a geração de documentos HTML bem formados;
- Não garante que as regras da linguagem utilizada são verificadas;
- Não garante que a informação recebida corresponde à informação esperada;
- Problemas de desempenho associados à utilização de ficheiros de texto como ficheiros de template, assim como o desempenho do carregamento do ficheiro e o elevado número de operações sobre `String`;
- Introduzem uma complexidade elevada porque requerem a utilização de pelo menos três linguagens, a linguagem utilizada, por exemplo HTML, a sintaxe própria do *template engine* na definição do template e a linguagem Java.

De uma forma geral o problema deste tipo de soluções é a falta de segurança na definição de templates. Isto é problemático porque tende a atrasar a descoberta de eventuais problemas presentes quer nos templates, quer na informação recebida. Actualmente existem ferramentas que conseguem validar se um documento HTML é válido ou não. Isto, apesar de resolver o problema, prova também dois aspectos: 1) a abordagem dos *template engines* é deficitária uma vez que necessita de ferramentas externas para validar os documentos que gera e, 2) que as

soluções de *template engines* poderiam integrar essas ferramentas nas suas soluções para garantir uma maior segurança para os seus utilizadores, mas no entanto não o fazem.

Para desenvolver uma solução que resolva os problemas apresentados anteriormente existem dois objectivos principais:

- Remover a utilização de ficheiros textuais para definir templates;
- Introduzir segurança na utilização deste tipo de solução, quer por garantir que as regras da linguagem utilizada são verificadas, quer por garantir que a informação recebida de fontes externas tem a sua utilização validada, assim como garantir que os documentos gerados são bem formados.

Para resolver o primeiro problema propomos que um template HTML passe a ser definido como uma *first-class function*. Para tornar isto possível é necessário gerar uma *interface fluente* que represente uma linguagem específica de domínio. Esta *interface fluente* deverá ser então utilizada por funções para definir templates para a linguagem HTML. No entanto esta *interface fluente* vai ser mais do que apenas uma ferramenta que manipula a linguagem HTML. O objectivo desta *interface fluente* é, para além de manipular a linguagem, limitar a dita manipulação da linguagem de acordo com as regras da mesma. Isto faz com que estejam garantidas as regras da linguagem, uma vez que o utilizador apenas pode manipular a linguagem com as operações que esta lhe disponibiliza, de acordo com a sua definição sintática, definida em XSD.

O nosso objectivo principal é criar todas as ferramentas necessárias para gerar *intefaces fluentes* com base no contéudo explícito em ficheiros XSD que descrevam uma linguagem específica de domínio. Com a alteração da definição dos templates para a linguagem Java passa a ser possível remover os ficheiros textuais que definem templates, o que leva a que sejam minimizados os problemas de desempenho introduzidos pelo carregamento de ficheiros de texto e reduzem-se também o número de operações sobre `Strings`. Isto leva ainda a outra vantagem, todas as manipulações relacionadas com a geração de um documento HTML passam a estar restritas à linguagem Java, removendo a complexidade da utilização de outras linguagens. O facto de terem sido criadas *interfaces fluentes* deve também ser destacado, uma vez que a utilização que uma *interface fluente* disponibiliza é intuitiva e de fácil adaptação para utilizadores que estejam habituados a utilizar *template engines* ou simplesmente a utilizar a linguagem em questão.

Nesta dissertação vai ser apresentada a minha proposta, chamada `xmlet`, que inclui diversas ferramentas que possibilitam:

- A análise, extração e estruturação da informação presente num ficheiro XSD que irá ser utilizada no processo de geração de uma linguagem de domínio;
- A geração de classes e métodos que juntos definem uma *interface fluente* que deverá refletir o maior número de regras presentes no ficheiro XSD que contém a especificação da linguagem específica de domínio que a *interface fluente* irá representar;
- A abstração da utilização da *interface fluente* recorrendo à utilização do padrão Visitor. Isto faz com que a mesma *interface fluente* possa ser utilizada para realizar tarefas distintas, sempre garantindo que as regras da respectiva linguagem são validadas, mas deixando a cargo do utilizador final a definição do comportamento que a respetiva linguagem tem no contexto do seu problema.

Para validar os diferentes componentes desta solução criaram-se linguagens de domínio não só para a linguagem HTML como também para outras linguagens. A segunda linguagem a ser utilizada para testar a utilização do `xmlet` foi a linguagem que permite definir os layouts visuais para o sistema operativo Android, usando um ficheiro XSD já existente que define todas as regras sintáticas desta linguagem. A terceira linguagem a ser utilizada foi um pouco diferente das duas apresentadas anteriormente sendo que quer a linguagem HTML quer a linguagem que define os layouts para Android são linguagens XML. Para testar a solução com uma linguagem completamente diferente foi utilizada a linguagem que permite exprimir expressões regulares. Esta linguagem é completamente diferente de uma linguagem que se exprima em XML, no entanto a sua sintaxe pode também ser expressa num ficheiro XSD. O ficheiro XSD para a linguagem das expressões regulares foi criado manualmente, disponibilizando todas as operações de expressões regulares que o Java suporta.

Após implementar todos os diferentes componentes da solução que será apresentada nesta dissertação tornou-se necessário avaliar o desempenho da mesma e comparar o desempenho com outras soluções semelhates. Nesta comparação incluímos soluções com abordagens diversas, no que toca a *template engines* com abordagens tradicionais utilizamos *templates engines* como Mustache, Handlebars, Pebble ou Trimou. Fizemos também a comparação com soluções mais recentes que introduzem detalhes interessantes nas suas soluções, como por exemplo,

mover a definição do template para dentro da linguagem Java, removendo a necessidade da existência de ficheiros textuais de template ou outras soluções que ainda definem os templates em ficheiros de texto mas que usam esses ficheiros de texto para gerar uma classe Java em tempo de compilação, classe essa que irá representar aquele template em tempo de execução, removendo a necessidade do carregamento de ficheiro de texto em tempo de execução, o que leva a resultados muito positivos em termos de desempenho.

Para realizar uma avaliação independente do desempenho das soluções a comparar foram utilizados projectos de avaliação de desempenho disponíveis no Github. Os projectos escolhidos são projectos que tem um largo número de seguidores pelo que se pressupõe que se tratam de projectos que realizam avaliações imparciais das soluções propostas. Os resultados apresentados nesta dissertação tentam ser o mais justos possível para todas as soluções envolvidas na comparação, efectuando várias medições para evitar possíveis erros.

Em termos de resultados da comparação dos diferentes *template engines* a resposta foi bastante positiva. A solução apresentada nesta dissertação é a solução mais eficiente entre todas as soluções que testamos. Este resultado torna-se ainda mais importante se pensarmos que esta solução para além de ser a mais eficiente é a que introduz mais segurança na sua utilização assim como uma menor complexidade, devido a se tratar de uma solução que utiliza apenas uma linguagem, Java.

Palavras-chave: XML, eXtensive Markup Language, XSD, eXtensive Markup Language Schema Definition, Geração Automática de Código, Interface Fluente, Linguage Específica de Domínio.