



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Automatic generation of Java API based on XML Schema

LUÍS CARLOS DA SILVA DUARTE

Licenciado em Engenharia Informática e de Computadores

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Doutor Fernando Miguel Gamboa de Carvalho

Júri:

Presidente: [Grau e Nome do presidente do júri]

Vogais: [Grau e Nome do primeiro vogal]
[Grau e Nome do segundo vogal]
[Grau e Nome do terceiro vogal]
[Grau e Nome do quarto vogal]

Junho, 2018



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Automatic generation of Java API based on XML Schema

LUÍS CARLOS DA SILVA DUARTE

Licenciado em Engenharia Informática e de Computadores

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Doutor Fernando Miguel Gamboa de Carvalho

Júri:

Presidente: [Grau e Nome do presidente do júri]

Vogais: [Grau e Nome do primeiro vogal]
[Grau e Nome do segundo vogal]
[Grau e Nome do terceiro vogal]
[Grau e Nome do quarto vogal]

Junho, 2018

Aos meus pais.

Agradecimentos

Ao meu orientador, por todo o apoio que me deu ao longo da realização desta dissertação. A todos os meus amigos que me acompanharam, ajudaram e animaram nesta jornada. E em particular, um grande agradecimento aos meus pais, sem eles nada disto seria possível.

Resumo

Independentemente da língua em que está escrita a dissertação, é necessário um resumo na língua do texto principal e um resumo noutra língua. Assume-se que as duas línguas em questão serão sempre o Português e o Inglês.

O *template* colocará automaticamente em primeiro lugar o resumo na língua do texto principal e depois o resumo na outra língua. Por exemplo, se a dissertação está escrita em Português, primeiro aparecerá o resumo em Português, depois em Inglês, seguido do texto principal em Português.

Resumo é a versão precisa, sintética e selectiva do texto do documento, destacando os elementos de maior importância. O resumo possibilita a maior divulgação da tese e sua indexação em bases de dados.

A redação deve ser feita com frases curtas e objectivas, organizadas de acordo com a estrutura do trabalho, dando destaque a cada uma das partes abordadas, assim apresentadas: Introdução - Informar, em poucas palavras, o contexto em que o trabalho se insere, sintetizando a problemática estudada. Objectivo - Deve ser explicitado claramente. Métodos - Destacar os procedimentos metodológicos adoptados. Resultados - Destacar os mais relevantes para os objectivos pretendidos. Os trabalhos de natureza quantitativa devem apresentar resultados numéricos, assim como seu significado estatístico. Conclusões - Destacar as conclusões mais relevantes, os estudos adicionais recomendados e os pontos positivos e negativos que poderão influir no conhecimento.

O resumo não deve conter citações bibliográficas, tabelas, quadros, esquemas. Dar preferência ao uso dos verbos na 3ª pessoa do singular. Tempo e verbo não devem dissociar-se dentro do resumo. Deve evitar o uso de abreviaturas e siglas - quando absolutamente necessário, citá-las entre parênteses e precedidas da explicação de seu significado, na primeira vez em que aparecem.

E, deve-se evitar o uso de expressões como "O presente trabalho trata ...", "Nesta tese são discutidos....", "O documento conclui que....", "aparentemente é...."etc.

Existe um limite de palavras, 300 palavras é o limite.

Para indexação da tese nas bases de dados e catálogos de bibliotecas devem ser apontados pelo autor as palavras-chave que identifiquem os assuntos nela tratados. Estes permitirão a recuperação da tese quando da busca da literatura publicada.

Palavras-chave: Palavras-chave (em português) ...

Abstract

The dissertation must contain two versions of the abstract, one in the same language as the main text, another in a different language. The package assumes the two languages under consideration are always Portuguese and English.

The package will sort the abstracts in the proper order. This means the first abstract will be in the same language as the main text, followed by the abstract in the other language, and then followed by the main text.

The abstract is critical because many researchers will read only that part. Your abstract should provide an accurate and sufficiently detailed summary of your work so that readers will understand what you did, why you did it, what your findings are, and why your findings are useful and important. The abstract must be able to stand alone as an overview of your study that can be understood without reading the entire text. However, your abstract should not be overly detailed. For example, it does not need to include a detailed methods section.

Even though the abstract is one of the first parts of the document, it should be written last. You should write it soon after finishing the other chapters, while the rest of the manuscript is fresh in your mind.

The abstract should not contain bibliography citations, tables, charts or diagrams. Give preference to the use of the verbs in the third person singular. Time and word must not dissociate yourself within the abstract. Abbreviations should be limited. Abbreviations that are defined in the abstract will need to be defined again at first use in the main text.

Finally, you must avoid the use of expressions such as "The present work deals with ... ", "In this thesis are discussed ", "The document concludes that ", "apparently and " etc.

The word limit should be observed, 300 words is the limit.

Abstracts are usually followed by a list of keywords selected by the author. Choosing appropriate keywords is important, because these are used for indexing purposes. Well-chosen keywords enable your manuscript to be more easily identified and cited.

Keywords: Keywords (in English) ...

Índice

Lista de Figuras	xv
Lista de Listagens	xvii
1 Introduction	1
1.1 Motivation	1
1.2 Use case	3
1.3 Document Organization	4
2 Existent Tools	5
2.1 J2html	5
2.2 Apache Velocity	5
3 Solution	7
3.1 Organization	7
3.2 Parsing	7
3.2.1 Document Object Model	8
3.3 Code Generation	8
3.3.1 Supporting Infrastructure	9
3.3.2 ASM	10
3.4 Client	10
3.5 HtmlFlow	10

4	Deployment	11
4.1	Github Organization	11
4.2	Maven	11
4.3	Sonarcloud	11
4.4	Testing metrics	12
5	Conclusion	13
5.1	Future work	13
	Referências	15

Lista de Figuras

1.1	Error validation in compile time	2
3.1	Supporting Infrastructure	9

Lista de Listagens

1.1	Failed HTML rule validation	2
1.2	Error validation in runtime	2



Introduction

The work described in this dissertation is concerned with the implementation of a Java solution, named `xmllet`, that allows the automatic generation of a fluent API based on a XML schema. The generated classes are very similar most of the time and such solution may save time to the user, eliminating repetitive tasks and human error.

1.1 Motivation

Text has evolved with the advance of technology resulting in the creation of markup languages [1]. This markup languages add annotations to text, also known as tags, that allow to add additional information to the text. Each markup language has its own tags and each of those tags add a different meaning to the text encapsulated within them. In order to use markup languages the users can write the text and add all the tags manually, either by fully writing them or by using some kind of text helpers such as intellisense which can help diminish the errors caused by manually writing the tags. But even with text helpers the resulting document can violate the restrictions of the respective markup language. In the following HTML example there is a violation of HTML rules, a `<html>` tag containing a `<div>` tag, which isn't allowed.

```
1 <html>
2   <div>
3     (...)
4   </div>
5 </html>
```

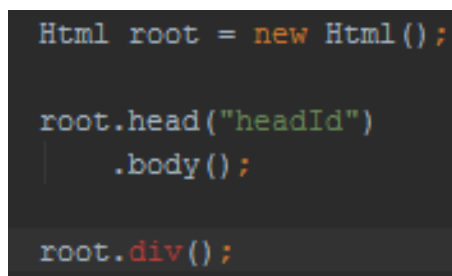
Listagem 1.1: Failed HTML rule validation

The solution to having documents that respect the markup language rules is changing the way the document writing works. If the user has control over the writing process errors can be induced in the document but if the writing control is given to an entity which can enforce the markup languages restrictions then it is guaranteed that the user can't produce a document with errors. In the following code sample the user is allowed to add any child to the Html element, resulting in a violation of the language restrictions, which is only detected during execution.

```
1 Html root = new Html();
2 root.add(new Div());
```

Listagem 1.2: Error validation in runtime

In order to create this entity all the restrictions of the given markup language could be recreated in a API and therefore allow the user to generate the document. In the 1.1, the API enforces the language restrictions in compile time, this way any document generated by this API will be valid.

A screenshot of a code editor with a dark background. The code is written in Java and shows the creation of an Html object, followed by a call to head() and body() methods, and finally a call to div(). The code is as follows:

```
Html root = new Html();

root.head("headId")
    .body();

root.div();
```

Figura 1.1: Error validation in compile time

The only problem with this solution is recreating all the rules of a given markup language, which can be a very long process since most markup languages have a vast number of elements and restrictions.

The solution for this problem is automation. An automated process that converts the definition of elements and restrictions of markups languages in classes that represent those elements and methods that enforce those restrictions. With this

automated process the application can generate a fluent API that allow the users to write their texts in a fluent way without errors and respecting the markup language semantics. This is the main objective of this work, creating an infrastructure that reads a markup language definition, in XML Schema Definition, and generates an API that allows the users to write well formed documents.

1.2 Use case

The use case that will be used to test and evaluate the solution will be the HTML5 XML schema. In this case we have multiple elements that share behavior and/or attributes that can be generated automatically. The generated classes allow to create a tree of elements that represent a XML document. This approach is very similar to DOM, the difference being that in this case each API will be specific to a given XSD file, in this case the HTML5 specification. The resulting classes can then be processed in different ways since the Visitor pattern is used. This way each different Visitor implementation can use the generated classes to write HTML documents to a stream, a socket or a file.

The generated HTML5 elements API will then be used in the HtmlFlow API, which is also a fluent Java API that is used to write well formed HTML files. With the Visitor pattern the HtmlFlow will only need to implement its own Visitor to achieve its goal. At the moment the HtmlFlow library only supports a set of the HTML elements which were created manually and the rest of the library interacts with those elements in order to write the HTML files. With the help of the solution which will be developed in this work the HtmlFlow will support the whole HTML syntax.

1.3 Document Organization

This document will be separated in five distinct chapters. The first chapter, this one, introduces the problem that was presented. The second chapter presents existent technology that was used. The third chapter explains in detail the different components of the suggested solution. The fourth chapter approaches the deployment and testing of the suggested solution. The fifth and last chapter of this document contains some final remarks and description of future work.

2

Existent Tools

A little introduction here

2.1 J2html

What is it? What are the similarities? What are the improvements introduced by this dissertation that make it better or worse than j2html?

2.2 Apache Velocity

What is it? What are the similarities? What are the improvements introduced by this dissertation that make it better or worse than apache velocity?



Solution

Here we gonna talk about the solution, its organization and components. Every component will have a detailed explanation.

3.1 Organization

The process of generating an API based on a xsd file contents is complex, so in order to simplify it it was divided into two smaller projects, each one with a given goal and responsibilities. The two main responsibilities in this process are parsing the information from the xsd file and generating the API based on that same parsed information.

3.2 Parsing

The parser will have the responsibility to read the XML schema element tree and extract the needed information in order to generate the respective classes. The result of the execution of this component should be a list of elements, each element representing a class that should be created and containing all the information needed for the creation.

3.2.1 Document Object Model

What is DOM? Why does its usage applies in this project? If DOM exists why does this project implements a XsdParser? Are there any alternatives to DOM, and if so, why was DOM chosen?

3.3 Code Generation

The class generator will have the responsibility of generating classes based on the information received from the Parser. The class generator should request the parsing of a XML schema file and based on the Parser result, create the classes accordingly. Apart from that the generator should also create an infrastructure that will help the usage of the the resulting API.

To achieve the generation of the classes a tool named ASM will be used. This tool allows the manipulation of byte codes, allowing the generation of classes, methods and fields.

3.3.1 Supporting Infrastructure

The generated code will be supported by an infrastructure that mimics the syntax of XML schema files. The supporting infrastructure is shown in 3.1.

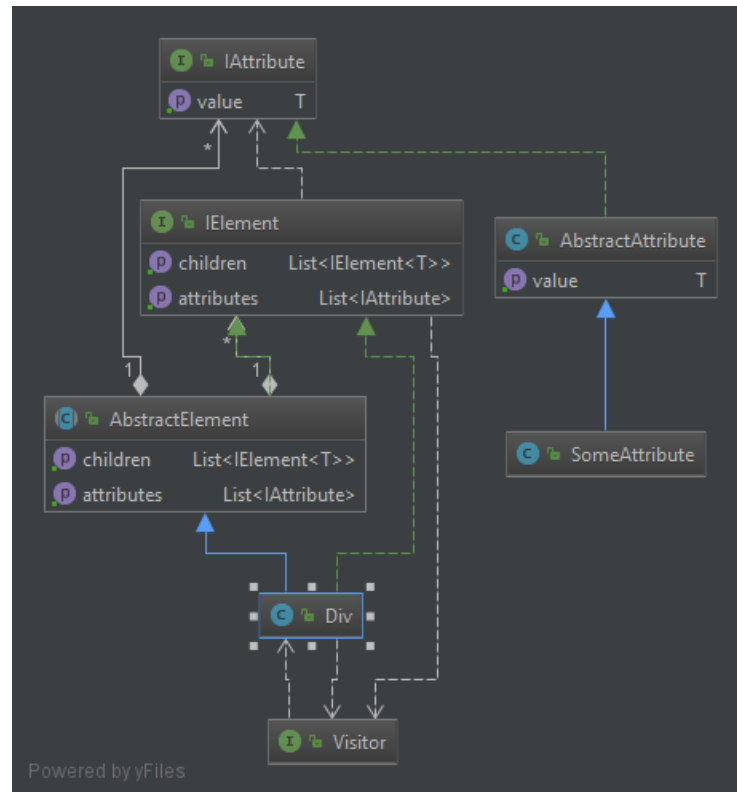


Figure 3.1: Supporting Infrastructure

All the generated APIs will have this classes which are independent of the contents of the parsed file. This infrastructure will then be extended by different type of classes, divided in four groups.

Elements

The elements are a group of classes that will be generated based in the existing xsd:elements. All these classes will extend AbstractElement. Each element will also contain the specific element code, which can include addition of elements, attributes or implementing group and element interfaces.

Attributes

The attributes are a group of classes that will be generated based in the existing xsd:attributes. All these classes will extend AbstractAttribute. Each attribute will have a type, that indicates the type of the value of the said attribute. Attributes

will also enforce restrictions to their value if there are any explicitly described in the XML schema file.

Group Interface

The group interfaces are an addition that represent the `xsd:attributegroups`. In the XML schema these attributegroups indicate that a given element is allowed to have the said attributes, in the generated code the respective interfaces allow the addition of all the attributes present in the said group to the element attributes.

Element Interface

The element interfaces are similar to the group interfaces, the difference being that element interfaces allow the addition of other elements as children of the current element.

Visitors

In order to the generated API allow manipulation by the client all the generated elements implement the Visitor pattern, therefore the client of the API can implement its own Visitor class and specify the behavior of the visit methods.

3.3.2 ASM

What is ASM? Why is it important in this project? Are there any alternatives to ASM? If so why was ASM chosen?

3.4 Client

Some stuff here.

3.5 HtmlFlow

More stuff here.

4

Deployment

4.1 Github Organization

This project and all its components belong to a github organization called xmlet . The aim of that organization is to contain all the related projects to this dissertation. All the generated APIs are also created as if they belong to this organization.

4.2 Maven

In order to manage the developed projects a tool for project organization and deployment was used, named Maven. Maven has the goal of organizing a project in many different ways, such as creating a standard of project building, managing project dependencies. Maven was also used to generate documentation and deploying the projects to a central code repository, Maven Central Repository .

4.3 Sonarcloud

Code quality and its various implications such as security, low performance and bugs should always be an important issue to a programmer. With that in mind all the projects present in this dissertation were evaluated in various metrics and

the results made public for consultation. This way, either future users of those projects or just for the developers, this metrics can be used to serve as another way of validating the quality of the produced code. The tool to perform this evaluation was Sonarcloud , which provides free of charge evaluations and stores the results which are available for everyone.

4.4 Testing metrics

Perform efficiency tests comparing the HtmlApi, j2html and Apache Velocity. The test should be based on a html page with multiple elements and attributes, probably the test should be performed with different number of html elements, like 10, 50, 100, 1000.



Conclusion

Here be conclusion.

5.1 Future work

Talk about adding support for `xsd:import` tag, moving the visitor calls to the insertion of elements and attributes in order to improve the efficiency of the resulting API.

Referências

- [1] Per. Christensson. Markup language definition., June 2011. URL https://techterms.com/definition/markup_language. (p. 1)

