Marshaling and Un-marshaling CityGML Using Various XML Bindings Techniques

Peter Follo, Robert Forsgren and Gustav Tolt
Sensor Informatics unit
FOI (Swedish Defence Research Agency)
Linköping, Sweden
peter.follo@foi.se, robert.forsgren@foi.se, gustav.tolt@foi.se

***Abstract*— This paper presents the methods and initial results of a study at the Swedish Defence Research Agency in 2012, aiming at investigating various techniques for binding a complex XML schema, such as the OGC's CityGML schema, to a database. This form of binding technique performs a mapping between XML objects and database objects thereby enabling the use of relational databases for update and input/output of customized CityGML models. The results so far indicate that this might be possible with today's framework but not without an undesirable work effort. Also, the lack of documentation makes this work even harder. Follow-on work is planned for 2013 and will provide a basis for an assessment of the frameworks. In this paper, we propose a method to evaluate such frameworks.**

Keywords-CityGML; XML binding; marshaling; un-marshaling

I. INTRODUCTION

The ever-increasing amount of geospatial data and the demand to exchange data within and between authorities and companies drive the demand of standards and regulated ways to fulfil this. One initiative to achieve this is the EU directive INSPIRE [1] stating that all geospatial data that affect environmental domains should be accessible to all member states within the EU. By 2018, these data should even be accessible online. On the national level, in Sweden, the answer to this directive is the implementation of a geodata portal lead by the Swedish mapping, cadastral and land registration authority [2]. Furthermore, many commercial initiatives are possible only if governmental geospatial data are freely available online [3]. This leads to more standards with increasing complexity being developed. One such example is the CityGML XML schema [4] from the Open Geospatial Consortium (OGC) [5].

The question, then, arises how to handle vast amounts of complex data – how to store and maintain it, how to keep up with emerging standards and how to prevent vendor lock-in? Digital geographical data, especially vector layers, are commonly stored in relational databases. It is not to stretch a thought to its limit to believe that also much of these emerging, more complex data structures will be, or at least are preferred to be, stored in relational databases, at least from the point of view of municipal authorities. One example of this is the 3DCityDB [6] developed at Technische Universität Berlin, which stores CityGML models in an Oracle Spatial database. While complex data can be represented in many ways, it is believed that one of the most common ways will be in XML documents with a supporting schema, which is why this paper focuses on such. Even though some relational databases, i.e., Oracle Spatial [7], handle XML directly, this is often done with a severe performance penalty and decreased functionality [8][9] and thus the need to handle complex data structures in traditional database schemes arises. Pure XML databases exist [10]; but, they often lack the GIS functionality present in other more traditional databases such as Oracle Spatial and PostgreSQL /PostGIS [11]. XML databases are also rare in the municipal community where often significant investments in time, money and education related to relational databases have already been made. Several frameworks for (semi-)automatically mapping XML schemes to data structures exist, but not many highlight how to deal with database storage.

This paper proposes a method for evaluating such frameworks and studies a selection of them for their potential ability to marshal and un-marshal CityGML models from and to a database. Since these frameworks are complex and often consist of frameworks-of-frameworks one needs a structured and efficient way to evaluate them. The paper also highlights some of the difficulties encountered with this semi-automatic approach. Our study focuses on the process of creating data structures to create and modify CityGML objects. This is done through automatic generation of source code and libraries from the CityGML XML schema. The goal is to achieve this with a reasonable work effort. CityGML references more than 40 other XML schemes making per schema adoption a cumbersome exercise. Some of the techniques studied in this paper may fail not because of technical impossibilities, but rather on impractical workload or lack of documentation.

Three XML schemas to data objects frameworks have been evaluated: Oracle's Java XML bindings (JAXB), Castor and Apache's Xmlbeans. Two data objects to Data Definition Language (DDL) frameworks, Castor and Hyperjaxb3 (the later based on JAXB and the Java Persistence Application Programming Interface (JPA)) were studied to generate DDL scripts. All these frameworks generate Java sources and the ones capable of generating DDL scripts are highly configurable in choice of database.

 ISBN: 978-1-61208-251-6

Other framework exists such as Apache's JaxMe, Extensible Stylesheet Language Transformations (XSLT) and others. Time, resources and in some cases technical difficulties prevented these and others to be evaluated in this study.

The remainder of this paper is organized as follows. The next section presents the method and the setup for the experiments conducted. Then, the Experiments section presents the conducted experiments and some intermediate findings. In Section IV, the results are summarized and figures of merits from the experiments are shown. The last chapter, Conclusions, presents the authors' beliefs regarding the feasibility to proceed further with the described method.

II. METHOD

The proposed method consists of three phases with increasing complexity.

The first phase, Object Model Generation, examines the framework's ability to generate compilable and useful source code. Within this phase we start with a minimal set of customizations and later on add more customizations when needed. When the source finally is generated the compilation stage starts. Some frameworks offer to do this automatically. If this fails we import the generated source code into some Integrated Development Environment (IDE) such as Netbeans and try to compile it that way. On success, to prove usability, we un-marshal and marshal one or more documents and compares the result with the original.

If the first phase is successful it is time to focus on the framework's DDL abilities which are done in the second phase, Data Model Generation. Again, with a similar approach as in phase 1, we try to generate DDL scripts with a minimal set of customizations. It is anticipated that this phase will generate very complex, maybe even impractical, data models, and these generated DDL scripts should not to be considered as the final product. If possible though; un-marshaling and marshaling should be done to prove no loss of data.

The third and last phase, Application Customization, focuses on the framework's abilities to customize the data model to the user's need and the database's capabilities, such as the ability to handle geometries and coordinate transformations. In addition, many times it is not necessary to store every part of the XML structure in a structured way. To reduce the complexity of the data model one might have to simplify the model. The third phase investigates if this is possible and what implications this has on reduced functionality and information loss.

III. EXPERIMENTS

In this section, we describe the work conducted for the first two experimental phases along with observations made during the work. The experiments are described in a chronological manner reflecting the difficulties that could arise along the way.

To evaluate the frameworks a LOD 3 sample data set from citygml.org covering a street in Frankfurt was downloaded and used. Each experiment was divided in three phases, as stipulated by the proposed method; however no framework reached the third phase due to technical difficulties.

The following (semi-)automatic XML bindings frameworks was used:

- JAXB version 2.2.4 (from the JDK)
- Hyperjaxb3 version 0.5.6
- Castor version 1.3
- Apache Xmlbeans version 2.4.0-7.

Among these frameworks, only Castor and Hyperjaxb3 have the ability to generate DDL scripts.

All experiments were conducted on a Dell XPS M1330 laptop with an Intel Dual Core 2.2 GHz Processor (T7500) and 4 GB of RAM. Operating system of choice was Fedora 16. Java version was Oracle's JDK version 1.7.0_04. Netbeans 7.0.1 was used as Java IDE.

A. Phase 1: Object Model Generation

1) Oracle JAXB 2.2.4

First try: generate and build a Java library using JAXB's *xjc* without any customization. This yielded name conflicts in the included XML schemes due to multiple definitions of elements with the same name. Some of them should really not be a conflict since they should be defined in different namespaces, i.e., the element *Role* defined both in cityObjectGroup.xsd and xlink.xsd. *xjc* is kind enough to output some hints to overcome the name clashes. Thus, the second try was to generate and build a Java library with a minimal set of customizations. Several conflicts arise due to XML names being converted to the same Java name, i.e., the XML name *_Solid* transforms to the Java name *Solid* which conflicts with a previous defined XML name *Solid* also transformed to the Java name *Solid*. Thirteen customizations were necessary due to conflicts in the XML schemas. Thereafter, the errors change focus to conflicts within the package to be built, conflicts sometimes already addressed in earlier customization, some of them really not conflicts at all. The third try was then to generate sources and build in a separate step. 594 Java source files were generated in about 10 seconds, including the time for internet access to the referenced schemes. The sources were imported into Netbeans and compiled without errors in 12 seconds. Marshaling and un-marshaling of the sample data set went without errors. Visual random inspection of the original data and the marshaled data indicated no errors or data loss.

2) Apache Xmlbeans 2.4.0-7

First try: generate and build Java library without any customizations. The *SchemaCompiler* worked out of the box generating 2526 classes in 6 seconds and building a Java library in 55 seconds with one ignorable warning about classpath settings. Marshaling and un-marshaling went without problems.

3) Castor 1.3

First try: generate and build Java libraries without any customizations. This resulted in a null pointer exception complaining on missing parent for "the built in parent type for: MeasureOrNullListType". No sources were generated. *MeasureOrNullListType* origins from the referenced GML 3.1.1 schema basicTypes.xsd.

 ISBN: 978-1-61208-251-6

Second try: generate sources for basicTypes.xsd. This yielded in more errors of the same type, this time pointing to *CountExtentType*. Some sources where generated though. No workaround has been found not implying much work or unacceptable loss of functionality why castor sadly has to leave the stage. A bug report (CASTOR-3223) has been filed.

B. Phase 2: Data Model Generation

1) Oracle JAXB 2.2.4/Hyperjaxb3 0.5.6

The first try was to examine the possibilities to add JPA annotations to the generated classes. The goal is to annotate every *complexType* as an Entity class. This was possible with the JAXB Annotate plugin [12], but only feasible if enabling the undocumented feature to allow multiple matches from the XPath expressions. Otherwise, one has to do a customization for each and every of the generated classes. Moreover, the JAXB Annotate plugin extensions were not allowed in the global binding scope, leading to a schema binding for each and every imported and included schema. However, Hyperjaxb3 targets this specific question, being a plugin to JAXB linking JAXB to JPA. Hyperjaxb3 also takes into account many other problems that arise when bridging the object model and data model [13]. Thus, the second try was to do the same experiment with Hyperjaxb3: Close to twenty customizations were made and given to Hyperjaxb3. 770 classes were generated, but the generated code did not compile due to invalid arguments to some methods. This occurs in four of the generated classes but it is manageable by manually editing the files. After editing, the process of generating DDL scripts starts but soon bails out again. According to the generated output, it is suggested that the hibernate *EntityManager* jars are missing. Further investigation could not conclude that this is the case; the needed jars seems to be included.

IV. RESULTS

Table 1 summarizes some figure of merits for the different frameworks together with some important notes. Oracle's JAXB 2.2.4 passes phase 1 but must be paired with some other technique to reach phase 3 of the experiments. This could be done with standalone JPA annotations but since hyperjaxb3 exists which do exactly this and more this was not further investigated in this study. Hyperjaxb3 was the most successful framework reaching the DDL generation phase but fails on a configuration error. No work-around was found. Hyperjaxb3 is a maven plugin, using several other artefacts. It is hard to follow exactly where or in what artefact it goes wrong. Also, it is hard to understand what is configurable or not from hyperjaxb3's point of view. Some configurations applied in the pom.xml file for the underlying artefacts did not follow through. Documentation to clarify this is desirable. Castor 1.3 fails with a null pointer exception not finishing phase 1. No documentation was found indicating what went wrong. To understand why one has to dig into the source code but this is out of scope of this study. Apache Xmlbeans 2.4.0-7 went without errors. No customizations was needed what so ever and marshalling and un-marshalling went without errors. Unfortunately, Xmlbeans is not able to generate DDL scripts.

TABLE I. FIGURE OF MERITS FOR THE DIFFERENT FRAMEWORKS.

Framework	Cust.	Classes	Note
Oracle's JAXB 2.2.4	3	594	Passes phase 1. Not applicable to phase 2 & 3 standalone.
Hyperjaxb3 0.5.6	16	770	Passes phase 1 but fails in phase 2 during DDL generation. Generated code does not compile without editing. Build must be done in a separate step.
Castor 1.3	7	-	Fails on phase 1.
Apache Xmlbeans 2.4.0-7	0	2526	Passes phase 1. No customizations needed. Not applicable to phase 2 & 3 standalone.

V. CONCLUSIONS

While promising, these frameworks still seem far from being capable to handle large complex XML schemas such as OGC's CityGML schema out-of-the-box. If not by technical means, so by the lack of documentation. The lack of documentation and the fact that these frameworks often are frameworks-of-frameworks makes debugging and understanding of the internal processes hard. Yes, the source code is there, but with productive aspects from e.g. a municipal agency's point of view it is not realistic to dig that deep into a problem. First and foremost, these frameworks must be better documented, to make it possible to know if you are trying to solve your problem the right way. This will give the developers the proper feedback to make the frameworks easier to use and in the long term more robust. Better documentation will also ease the burden of the developers and the community to answer newbie questions and it will also ease the burden of the users to adopt these techniques. In the (failed) experiments it is pointed out, several times, that no work-around was found, although this does not mean that such does not exist.

VI. FUTURE WORK

In this paper, we have investigated three frameworks for binding a complex XML schema to a database.

One of the most promising, due to its ease-of-use, frameworks is Apache Xmlbeans. However, it could not be investigated thoroughly enough up to this point due to time and resource constraints, but a deeper analysis is scheduled as an upcoming action. We also note that this framework most be combined with other techniques to reach the third phase of the experiments.

From what we have experienced so far, Hyperjaxb3 should also be further investigated to see whether it is possible to circumvent the configuration error described in Section III.B.1. It must be investigated how the hyperjaxb3 maven plugin interacts with other maven plugins, especially

 ISBN: 978-1-61208-251-6

the *hibernate* and *ant task* plugins. Lack of time prevented us from addressing this issue thoroughly in this study, but will instead be considered as a next step.

It is also desirable to dig deeper to find the exact reason for processing errors, and for this we anticipate that working closer together with the developers and the community is a necessity.

A better XML comparison technique must be developed; we must be able to compare large XML files in a more clever way than manual inspection to guarantee integrity of the object during marshaling and un-marshaling; taking into account the fact that the object can be represented in many valid ways in an XML structure.

ACKNOWLEDGMENTS

Thanks to the developers of these frameworks, working under high pressure and short of resources, sometimes sacrificing their spare time to answer questions and fix bugs. Without your efforts this would be a daunting task, maybe even impossible. Thanks also to the companies sponsoring them in their work working along with the open source community.

REFERENCES

[1] EU, "Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE)", European Commission, 2007

[2] Lantmäteriet - the Swedish mapping, cadastral and land registration authority, "Geodataportalen", Accessed Dec 2012: http://www.geodata.se/GeodataExplorer/start.jsp?loc=en

[3] G. Vickery, "Review of recent studies on PSI re-use and related market developments", European Commission, 2011

[4] Open Geospatial Consortium, "OpenGIS® City Geography Markup Language (CityGML) Encoding Standard", OGC, 2008

[5] Open Geospatial Consortium, "About OGC|OGC®", Accessed Jan 2013: http://www.opengeospatial.org/ogc

[6] A. Stadler, C. Nagel, G. König. and T.H. Kolbe, "Making interoperability persistent: A 3D geo database based on CityGML", Lee & Zlatanova (eds.), 3D Geo-Information Sciences, Selected papers from the 3rd International Workshop on 3D Geo-Information, Seoul, Korea. LNG&C series, Springer Verlag, pp. 175-192. 2008

[7] Oracle, "Oracle Database Online Documentation", Accessed Aug 2012: http://www.oracle.com/pls/db111/portal.portal_db?frame=&selected=7

[8] PostgreSQL, "PostgreSQL: Manuals", Accessed Dec 2012: http://www.postgresql.org/docs/manuals/

[9] MySQL, "MySQL :: MySQL 5.5 Reference manual", Accessed Dec 2012, http://dev.mysql.com/doc/refman/5.5/en/index.html

[10] "XML database", Wikipedia, Accessed Dec 2012: http://en.wikipedia.org/wiki/XML_database

[11] PostGIS, "PostGIS: Home", Accessed Dec 2012: http://www.postgis.org/

[12] A. Valikov, "Annotate Plugin - Confluence", Accessed Dec 2012: http://confluence.highsource.org/display/J2B/Annotate+Plugin

[13] A. Valikov, "JAXB vs. JPA - Confluence", Accessed Dec 2012: http://confluence.highsource.org/display/HJ3/JAXB+vs.+JPA

 ISBN: 978-1-61208-251-6