

Instance-based XML Data Binding for Mobile Devices

Alain Tamayo
Institute of New Imaging
Technologies
Universitat Jaume I, Spain
Ave Vicent Sos Baynat, SN,
12071, Castellón de la Plana
atamayo@uji.es

Carlos Granell
Institute for Environment
and Sustainability
European Commission
Joint Research Centre
Ispra, Italy
carlos.granell@jrc.ec.europa.eu

Joaquín Huerta
Institute of New Imaging
Technologies
Universitat Jaume I, Spain
Ave Vicent Sos Baynat, SN,
12071, Castellón de la Plana
huerta@uji.es

ABSTRACT

XML and XML Schema are widely used in different domains for the definition of standards that enhance the interoperability between parts exchanging information through the Internet. The size and complexity of some standards, and their associated schemas, have been growing with time as new use case scenarios and data models are added to them. The common approach to deal with the complexity of producing XML processing code based on these schemas is the use of XML data binding generators. Unfortunately, these tools do not always produce code that fits the limitations of resource-constrained devices, such as mobile phones, in the presence of large schemas. In this paper we present *Instance-based XML data binding*, an approach to produce compact application-specific XML processing code for mobile devices. The approach utilises information extracted from a set of XML documents about how the application make use of the schemas.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation—*Languages and System, Standards*

General Terms

Performance, Design, Experimentation, Standardization, Languages

Keywords

XML Processing, XML Schema, Mobile applications, XML Data Binding

1. INTRODUCTION

eXtensible Markup Language (XML) has reached a great success in the Internet era. XML documents are similar to HTML documents, but do not restrict users to a single vocabulary, which offers a great deal of flexibility to represent

information. To define the structure of documents within a certain vocabulary, schema languages such as *Document Type Definition* (DTD) or *XML Schema* are used.

XML has been adopted as the most common form of encoding information exchanged by Web services [14, 35, 36]. [14] attribute this success to two reasons. The first one is that the XML specification is accessible to everyone and it is reasonably simple to read and understand. The second one is that several tools for processing XML are readily available. We add to these reasons that as XML is *vocabulary-agnostic*, it can be used to represent data in basically any domain. For example, we can find the *Universal Business Language* (UBL)¹ in the business domain, or the standards defined by the *Open Geospatial Consortium* (OGC) in the geospatial domain. UBL defines a standard way to represent business documents such as electronic invoices or electronic purchase orders. OGC standards define *web service interfaces* and *data encodings* to exchange geospatial information. All of these standards (UBL and OGC's) have two things in common. The first one is that they use XML Schema to define the structure of XML documents. The second one is that the size and complexity of the standards is very high, making very difficult its manipulation or implementation in certain scenarios [20, 23].

The use of such large schemas can be a problem when XML processing code based on the schemas is produced for a resource-constrained device, such as a mobile phone. This code can be produced using a manual approach, which will require the low-level manipulation of XML data, often producing code that is hard to modify and maintain. Another option is to use an XML data binding code generator that maps XML data into application-specific concepts. This way developers can focus on the semantics of the data they are manipulating [34]. The problem with generators is that they usually make a straightforward mapping of schema components to programming languages constructs that may result in a binary code with a very large size that cannot be easily accommodated in a mobile device [23].

Although schemas in a certain domain can be very large this does not imply that all of the information contained on them is necessary for all of the applications in the domain. For example, in [26] a study of the use of XML in a group of 56 servers implementing the *OGC's Sensor Observation Service (SOS) specification*² revealed that only 29.2% of the

¹<http://docs.oasis-open.org/ubl/cs-UBL-2.0/UBL-2.0.html>

²SOS is a standard web service interface to exchange information between sensor data producers and consumers[19].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

M-MPAC'2011, December 12th, 2011, Lisbon, Portugal.

Copyright 2011 ACM 978-1-4503-1065-9/11/12 ...\$10.00.

SOS schemas were used in a large collection of XML documents gathered from those servers. Based on this information we proposed in [25] an algorithm to simplify large XML schema sets in an application-specific manner by using a set of XML documents conforming to these schemas. The algorithm allowed a 90% reduction of the size of the schemas for a real case study. This reduction was translated in a reduction of binary code ranging between 37 to 84% when using code generators such as JAXB³, XMLBeans⁴ and XBinder⁵.

In this paper we extend the schema simplification algorithm presented in [25] to a more complete *Instance-based XML Data Binding* approach. This approach allows to produce very compact application-specific XML processing code for mobile devices. In order to make the code as small as possible the approach will use, similarly to [25], a set of XML documents conforming to the application schemas. From these documents, in addition to extract the subset of the schemas that is needed, we extract other relevant information about the use of schemas that can be utilised to reduce the size of the final code. A prototype implementation targeted to Android⁶ and the Java programming language has been developed.

The remainder of this paper is structured as follows. Section 2 presents an introduction to XML Schema and XML data binding. In Section 3, related work is presented. The *Instance-based data binding approach* is presented in Section 4. Section 5 overviews some implementation details and limitations found during the development of the prototype. Section 6 presents experiments to measure size and execution times of the code generated by the tool in a real scenario. Last, conclusions and future work are presented.

2. BACKGROUND

In this section we present a brief introduction to the topics of XML Schema and XML data binding. XML Schema files are used to assess the validity of well-formed element and attribute information items contained in XML instance files [31][32]. The term XML data binding refers to the idea of taking the information in an XML document and convert it to instances of application objects [17].

2.1 XML Schema

An XML Schema document contains components in the form of complex and simple type definitions, element declarations, attribute declarations, group definitions, and attribute group definitions. This language allows users to define their own types, in addition to a set of predefined types defined by the language. Elements are used to define the content of types and when global, to define which of them are valid as top-level element of an XML document.

XML Schema provides a derivation mechanism to express subtyping relationships. This mechanism allows types to be defined as subtypes of existing types, either by extending or restricting the content model of base types. Apart from type derivation, a second subtyping mechanism is provided through substitution groups. This feature allows global elements to be substituted by other elements in instance files. A global element *E*, referred to as *head element*, can be sub-

stituted by any other global element that is defined to belong to the *E*'s substitution group.

2.2 XML Data Binding

With *XML Data Binding*, an abstraction layer is added over the raw XML processing code, where XML information is mapped to data structures in an application data model. XML data binding code is often produced by using code generators that use a description of the structure of XML documents using some schema language. The use of generators potentially gives benefits such as increased productivity, consistent quality throughout all the generated code, higher levels of abstraction as we usually work with an abstract model of the system; and the potential to support different programming languages, frameworks and platforms [10].

Although most of the generators available nowadays are targeted to desktop or server applications, several tools have been developed for mobile devices such as XBinder and CodeSysynthesis XSD/e⁷, or for building complete web services communication end-points for resource constrained environments, such as gSOAP [29]. All of the tools mentioned before map XML Schema structures to programming languages construct in a straightforward way, which is not adequate when large schemas sets are used.

3. RELATED WORK

Problems related with having large and complex schemas have been presented in several articles [20, 21, 25, 30]. For example, [20] deal with problems of large schemas in schema matching in the business domain. In the context of schema and ontology mapping, [21] states that current match systems still struggle to deal with large-scale match tasks to achieve both good effectiveness and good efficiency. [25], the work extended here, expose the problems related to using XML data binding tools to generate XML processing code for mobile geospatial applications. Last, [30] present an algorithm to extract fragments of large conceptual schemas arguing that the largeness of these schemas makes difficult the process of getting the knowledge of interest to users.

When considering XML processing in the context of mobile devices, literature is focused in two main competing requirements: *compactness* (of information) and *processing efficiency* [12]. To achieve compactness compression techniques are used to reduce the size of XML-encoded information [11, 13, 33]. About processing efficiency, not much work has been done in the mobile devices field. A prominent exception in this topic is the work presented in [12], [13] and [16]. These articles are all related to the implementation of a middleware platform for mobile devices: the *Fuego mobility middleware* [28], where XML processing has a large impact. The proposed *XML stack* provides a general-purpose XML processing API called *XAS* [12], an XML binary format called *Xebu* [13], and others APIs such as *Trees-with-references* (RefTrees) and *Random Access XML Store* (RAXS)[16].

Regarding the use of instance files to drive the manipulation of schemas, [21] presents a review of different methods that use instance files for ontology matching. In the field of schema inference, instance files are used as well to generate adequate schema files that can be used to assess their validity (e.g. [2, 9, 18]).

³<https://jaxb.dev.java.net>

⁴<http://xmlbeans.apache.org>

⁵<http://www.obj-sys.com/xbinder.shtml>

⁶<http://www.android.com>

⁷<http://codesynthesis.com/products/xsde>

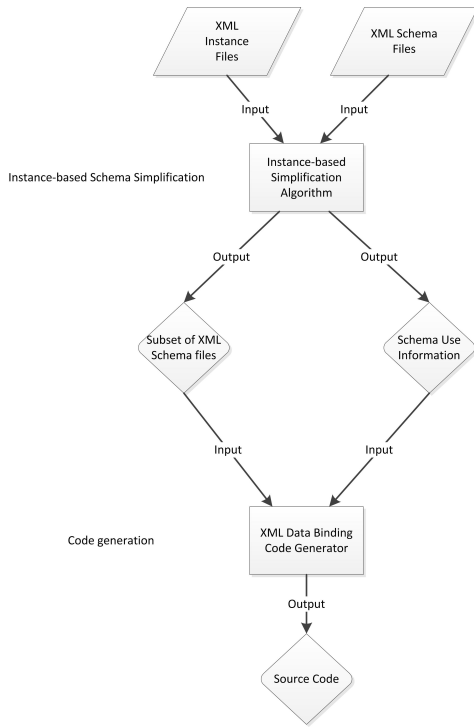


Figure 1: Instance-based XML data binding code generation process

4. INSTANCE-BASED DATA BINDING

Instance-based XML data binding, is a two-step process. The first step, *Instance-based schema simplification*, extracts the information about how schema components are used by a specific application, based on the assumption that a representative subset of XML documents that must be manipulated by the application is available. The second step, *Code generation*, consists of using all of the information extracted in the previous step to generate XML processing code as optimised as possible for a target platform.

The whole process is shown in Figure 1, the inputs to the first step are a set of schemas and a set of XML documents conforming to them. The outputs will be the subset of the schemas used by the XML documents and other information about the use of certain features of the schemas that can be used to optimise the code in the following step. The outputs of the first step are the inputs of the code generation step. The two steps of the process are detailed in the following subsections.

4.1 Instance-based Schema Simplification

The *Instance-based schema simplification* step extracts the subset of the schemas used on a set of XML documents. The algorithm used to perform this simplification was first presented in [25] and has been extended here to extract other information that can be used to produce more compact XML processing code.

The idea behind this algorithm, is depicted graphically in Figure 2. The figure shows to the left the graph of relationships between schemas components. The different planes represent different namespaces. Links between schema components represent dependencies between them. To the right

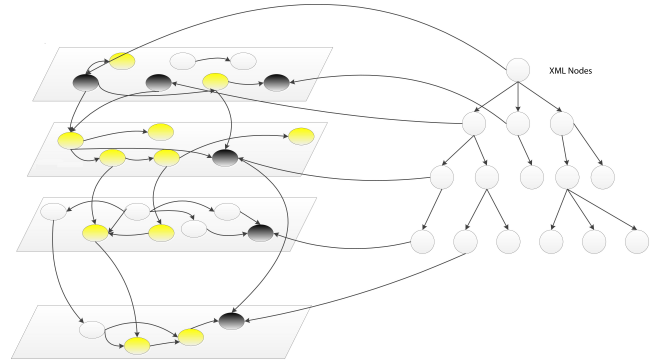


Figure 2: Relations between XML instance information items (right) and schemas components defining its structure (left)

we have the tree of information items (XML nodes) contained in XML documents. For the sake of simplicity we show in the figure only the tree of nodes corresponding to a single document. An edge between an XML node and a schema component represents that the component describes the structure of the node. To simplify the figure we have shown only a few edges, although an edge for every XML node must exist. Starting from a set of XML documents and the schema files defining their structure, it is possible to calculate which schema components are used and which are not. In doing so, the following information is also recorded:

- *Types that are instanced in XML documents*: For each XML node exists a schema type describing its structure. While XML documents are processed the type of each XML node is recorded. This way we can know which types are instanced and which are not.
- *Types and elements substitutions*: The subtyping mechanisms mentioned in Section 2.1 allow the *real* or *dynamic type* of an element to be different from its *declared type*. Elements declared as having type A, may have any type derived from A in an XML document. In this case the real type must be specified with the attribute *xsi:type*. Something similar happens with substitution groups, although in this case the attribute *xsi:type* is not necessary. The information about XML nodes whose dynamic type is different from its declared type is recorded.
- *Wildcards substitutions*: The elements used to substitute wildcards are recorded.
- *Elements occurrence constraints information*: For all of the elements it is checked that if they allow multiple occurrences there is at least one document where several occurrences of the element are present.
- *Elements with a single child*: All of the elements that contain a single child are also recorded.

4.2 Code generation process

A more detailed view of the code generation process is shown in Figure 3. The outputs of the schema simplification step are used as inputs to the *schema processor*, the

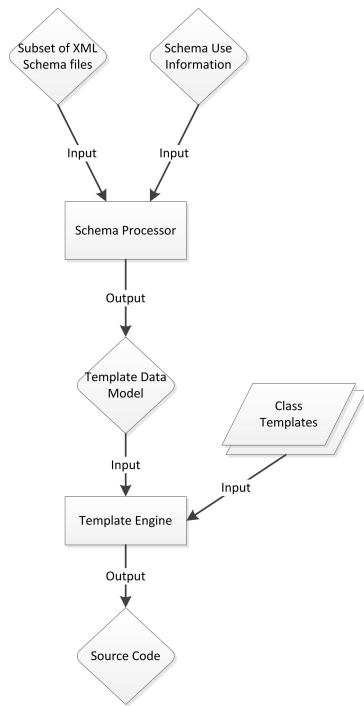


Figure 3: Flow diagram for the code generation process

component of the generator in charge of creating the data model that will be used later by the *template engine*. The *template engine* combines pre-existing *class templates* with the data model to generate the final source code. The use of a template engine allows the generation of code for other platforms and programming languages by just defining new class templates.

A summary of the features of the code generation process that contribute to the generation of optimised code is listed next:

- *Use of information extracted from XML documents:* The use of information about schema use allows to apply the following optimisations:
 - *Remove unused schema components:* The schema components that are not used are not considered for code generation. By removing the unused components we can substantially reduce the size of the generated code. The amount of the reduction will depend on how specific applications make use of the original schemas.
 - *Efficient handling of subtyping and wildcards:* The number of possible substitutions of a type by its subtypes, and a head element of a substitution group by the members of the group can be bounded with the information gathered from the instances files. In the general case, where no instance-based information is available, generic code to face any possible type or element substitution must be written. Limiting the number of possible substitutions to only a few allows the production of simpler and faster code. The same reasoning is applied to wildcards.

- *Inheritance flattening:* By flattening subtyping hierarchies for a given type, i.e., including explicitly in its type definition all of the fields inherited from base types and eliminating the subtype relationship with its parents, we can reduce the number of classes in the generated code. The application of this technique will not necessarily result in smaller generated code, as the fields defined in base types must be replicated in all of their child types, but it will have a positive impact in the work of the class loader because a lower number of classes have to be loaded while the application is executed. Let us consider the case of the geospatial schemas introduced in Section 1. These schemas typically present deep subtyping hierarchies with six or more levels, as a consequence when an XML node of a type in the lowest levels of the hierarchy must be processed, all of its parent types must be loaded first. The technique of inheritance flattening has been widely explored and used in different computer science and engineering fields as is proven by the abundant literature found in the topic [3, 4, 5, 6, 7, 15].

- *Adjust occurrence constraints:* If an element is declared to have multiple occurrences it must be mapped to a data structure in the target programming language that allows the storage of the multiple instances of the elements, e.g. an array or a linked list. In practice if the element has at most one occurrence in the XML documents that must be processed by the application it can be mapped to a single object instance. Using this optimisation the final code will make a better use of memory because instead of creating a collection (array, linked list, etc.) that will only contain a single object, it creates a single object instance.

- *Collapse elements containing single child elements:* Information items that will always contain single elements can be replaced directly by its content. By applying this optimisation we can reduce the number of classes in the generated code, which will have a positive impact in the size of the final code, the amount of work that has to be done by the class loader, and the use of memory during execution. This optimization is used by mainstream XML data binding tools such as JiBX⁸ and the XML Schema Definition Tool⁹.
- *Disabling parsing/serialization operations as needed:* Some code generators always includes code for parsing and serialization even when only one of these functions is needed. For example, in the context of geospatial web services, most of the time spent in XML processing by client-side applications is dedicated to parsing, as messages received from the servers are potentially large. On the other hand, most of these services allows request to be sent to the server encoded in an HTTP GET request, therefore XML serialisation is not needed at all.
- *Ignoring sections of XML documents:* Frequently, we are not interested in all of the information contained in

⁸<http://jibx.sourceforge.net/>

⁹<http://msdn.microsoft.com/en-us/library/x6c1kb0s>

XML files, ignoring the unneeded portions of the file will improve the speed of the parsing process and it may have a significant impact in the amount of memory used by the application.

In addition, the following features not related directly with code optimisation are also supported:

- *Source code based on simple code patterns*: The generated source code is straightforward to understand and modify in case it is necessary.
- *Tolerate common validation errors*: Occasionally, XML documents that are not valid against their respective schemas must be processed by our applications. In many cases, the validation errors can be ignored following simple coding rules.

A detailed explanation of each of the features presented in this section can be found in [22].

As mentioned before, the approach presented in this paper is based on the assumption that a representative set of XML documents exists. By *representative* we mean that these documents contain instances of all of the possible XML Schema elements and types that will be processed by the application in the future. Nevertheless, this subset might not always be available. In this case, we can still take advantage of the approach by building *synthetic* XML documents containing relevant information. Whether XML processing code is produced manually or automatically developers typically have some knowledge of the structure of the documents that must be processed by the applications. Therefore, we can use this knowledge to build sample XML documents that can be used as input to the algorithm. In case it were necessary, the final code can be manually modified later, or the sample files changed and used to regenerate the code.

If we were using synthetic documents instead of actual documents some of the optimisations related to the information extracted from them should not be applied. The reason for this is that we do not have enough information about how the related schema features are used. For example, we cannot apply optimisations such as the efficient handling of subtyping and wildcards, as we might not know all of the possible type substitutions. Something similar happens with the adjustment of occurrence constraints. Nevertheless, other optimisations such as inheritance flattening or removing unused schema components can be still safely applied.

5. IMPLEMENTATION

DBMobileGen (DBMG for short) is the current implementation of the *Instance-based XML data binding* approach [22]. It includes components implementing both the simplification algorithm and code generation process. It is implemented in Java and relies on existing libraries such as *Eclipse XSD*¹⁰ for processing XML schemas, *Freemarker*¹¹ as template engine library, and as well as the generated code, *kXML*¹² for low-level XML processing. This tool produces code targeted to Android mobile devices and the Java programming language.

¹⁰<http://www.eclipse.org/modeling/mdt/?project=xsd#xsd>

¹¹<http://freemarker.sourceforge.net>

¹²<http://kxml.sourceforge.net/kxml2/>

The current implementation has some limitations. Because of the complexity of the XML Schema language itself, support for certain features and operations have been only included if it is considered necessary for the case study or applications where the tool has been used [1, 27]. Some of these limitations are listed next:

- *Serialization is not supported yet*: The role of parsing for our sample applications and case studies is far more important than serialization. This is mainly because we have preferred to use HTTP GET to issue server requests wherever possible.
- *Dynamic typing using xsi:type not fully supported*: The mechanism of dynamic type substitution by using the *xsi:type* attribute has not been fully implemented yet, as the XML documents processed in the applications developed so far do not use this feature.

6. EXPERIMENTS

In this section we present two experiments. The first one tries to test how much the size of the generated code can be reduced by using DBMG. The second one measures the execution times of generated code in a mobile phone.

6.1 Measuring code size

In this experiment we borrow the test case presented in [25] that implements the communication layer for an SOS client. SOS is a standard web service interface defined to enhance interoperability between sensor data producers and consumers [19]. The SOS schemas are among the most complex geospatial web service schemas as they are comprised of more than 80 files and they contain more than 700 complex types and global elements [23].

The client must process data retrieved from a server that contains information about air quality for the Valencian Community. This information is gathered by 57 control stations located in that area. The stations measure the level of different contaminants in the atmosphere.

A set of 2492 XML documents was gathered from the server to be used as input, along with the SOS schemas, to the Instance-based data binding process. The source code generated by DBMG is compiled to the *compressed jar* format and compared with the final code generated by other generators: *XBinder*, *JAXB* and *XMLBeans*. The last two are not targeted to mobile devices but are used here as reference to compare the size of similar code for other types of applications.

Table 1 shows the size of the code produced with the different generators from the full SOS schemas (Full) and from the subset of the schemas used in the input instance files (Reduced). The reduced schemas are calculated applying the schema simplification algorithm to the full SOS schemas. The last row of the table (Libs) includes the size of the supporting libraries needed to execute the generated code in each case.

Figure 4 shows the total size of XML processing code when using the full and reduced schemas. In both cases, we can see the enormous difference that exists between the code generated by DBMG and the code generated by other tools.

The size for DBMG is the same in both cases because it implicitly performs the simplification of the schemas before generating source code. It must be noted that serialisation is not still implemented in DBMG. We roughly estimate

Table 1: Comparing size of code (KBs) for original and simplified schema sets

	XBinder	JAXB	XMLBeans	DBMG
Full	3,626	754	2,822	88
Reduced	567	90	972	88
Libs	100	1,056	2,684	30

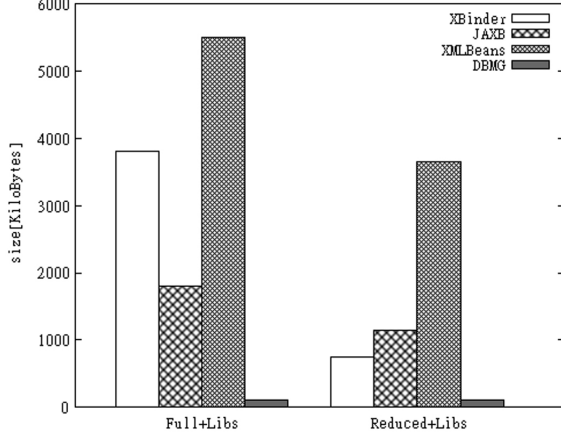


Figure 4: Size of generated code for full schemas

that including serialisation code will increase the final size in about 30%. In any case, the code generated by this tool is about 6 times smaller than the code generated by XBinder from the reduced schemas and about 30 times smaller than the code generated from the full schemas. One of the reasons for this difference in size is the lack of serialisation support in DBMG. Another reason is that XBinder generates code to ensure all of the restrictions related to user-defined simple types. This is an advantage if we parse data obtained from a non-trusted source and the application requires the data to be carefully validated, but it is a disadvantage in the opposite case, as unneeded verification increase processor usage and memory footprint. In the case of DBMG, as it aggressively tries to lower final code size, these simple type restrictions are ignored and these types do not even have a counterpart in the generated code.

When compared to JAXB, using the reduced schemas, the main difference in size is in the supporting libraries, as the code generated by JAXB is very simple. Still, the code generated by DBMG is slightly smaller because the step of removing elements with single child elements and inheritance flattening eliminates a large number of classes. In all of the cases, XMLBeans has the largest size. This tool is mostly optimised for speed at the expense of generating a more sophisticated and complex code and the use of bigger supporting libraries.

6.2 Measuring execution times

To test the performance of the generated code we will parse a set of 38 capabilities files¹³ obtained from different SOS servers. The code needed to parse these files is gen-

¹³Capabilities files contain metadata about the information contained in a server instance that implement any of the OGC's web service interfaces.

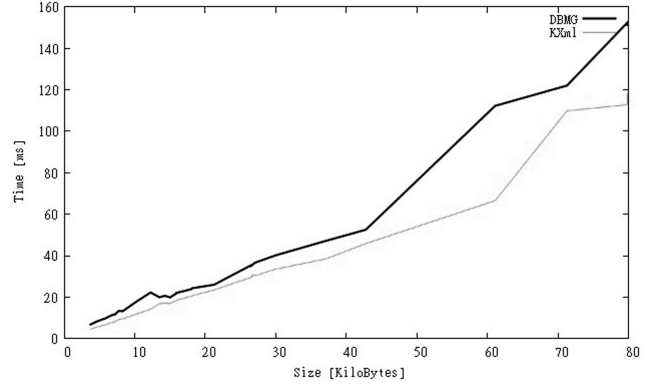


Figure 5: Execution times for small files

erated and deployed to a HTC Desire Android smartphone with a 1 GHz Qualcomm QuadDragon CPU and 576 MB of RAM. The 38 files have sizes ranging from less than 4 KB to 3.5 MB, with a mean size of 315 KB and a standard deviation of 26.7 KB. As the size range is large and with the purpose of simplifying presentation we divide the files in two groups, those with a size below 100 KB, CAPS-S (30 files), and those with size equal to or higher than 100 KB, CAPS-L (8 files).

To obtain accurate measures of the execution time for the code we selected the methodology presented in [8]. This methodology provides a statistically rigorous approach to deal with all of the non-deterministic factors that may affect the measurement process (multi-threading, background processes, etc.).

As our goal is only to measure the execution times of XML processing code, we stored the files to be parsed locally to avoid interferences related to network delays. Besides, to minimise the interference of data transfer delays from the storage medium all of the files below 500 KB were read into memory before being parsed. It was impossible to do the same for files with sizes above 500 KB because of the device memory restrictions.

Figures 5 and 6 shows the execution times of code generated by DBMG. The figures also include the execution times needed by *kXML*, the underlying parser used by DBMG, to process the same group of files. The execution times for *kXML* were calculated by creating a simple test case where files are processed using this parser, but no action is taken when receiving the events generated by it.

When files below 100 KB are processed it can be observed that the overhead added by the generated code is not high (Figure 5). Nevertheless, we can see in Figure 6 that when file size is above 1 MB, the overhead starts to be important (>1s). This happens because the large amount of memory that is required to store the information that is being processed forces the execution of the garbage collector with a high frequency. We have to keep in mind that code produced manually can have similar problems if it were necessary to retain most of the information read from the XML files in memory.

The experiment described above was extended in [24] to compare the code generated by DBMG with other data binding tools and to measure also the performance of this code

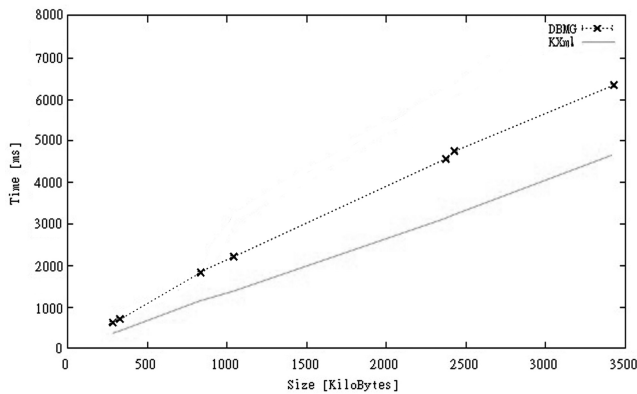


Figure 6: Execution times for large files

when executed in a Windows PC. The experiments showed that the execution times for the mobile devices were around 30 to 90 times slower than those for the personal computer. The experiments also showed that the code generated by DBMG was as fast as code generated by other data binding tools for the Android platform.

7. CONCLUSIONS

In this paper we have presented an approach to generate compact XML processing code based on large schemas for mobile devices. It utilises information about how XML documents make use of its associated schemas to reduce the size of the generated code as much as possible. The solution proposed here is based on the observation that applications that makes use of XML data based on large schemas do not use all of the information included in these schemas.

A code generator implementing the approach that produces code targeted to Android mobile devices and the Java programming language has been developed. This tool has been tested in a real case study showing a large reduction in the size of the final XML processing code when compared with other similar tools generating code for mobile, desktop and server environments. Nevertheless, this result must be looked at with caution as the magnitude of the reduction will depend directly from the use that specific applications make of their schemas.

8. ACKNOWLEDGEMENTS

This work has been partially supported by the “España Virtual” project (ref. CENIT 2008-1030) through the Instituto Geográfico Nacional (IGN); and project GEOCLOUD, Spanish Ministry of Science and Innovation IPT-430000-2010-11.

9. REFERENCES

- [1] P. Aragó, A. Tamayo, P. Viciano, J. Huerta, and L. Díaz. Forest Fire Survey and Processing Tool for Android-Based Mobile Devices. In *INSPIRE Conference 2011*, 2011.
- [2] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *Proceedings of 33rd International Conference on Very Large Databases (VLDB '07)*, pages 998–1009. VLDB Endowment, 2007.
- [3] D. Beyer, C. Lewerentz, and F. Simon. Impact of inheritance on metrics for size, coupling, and cohesion in object-oriented systems. In *Proceedings of the 10th International Workshop on New Approaches in Software Measurement, IWSM '00*, pages 1–17, London, UK, 2000. Springer-Verlag.
- [4] C. bogdan Chirila, M. Ruzsilla, P. Crescenzo, D. Pescaru, and E. Tundrea. Towards a reengineering tool for java based on reverse inheritance. In *Proceedings of the 3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence (SACI 2006)*, pages 963–7154, 2006.
- [5] H.-J. Bungartz, W. Eckhardt, M. Mehl, and T. Weinzierl. DaStGen - A Data Structure Generator for Parallel C++ HPC Software. In *Proceedings of the 8th International Conference on Computational Science, Part III, ICCS '08*, pages 213–222, Berlin, Heidelberg, 2008. Springer-Verlag.
- [6] H. J. Bungartz, W. Eckhardt, T. Weinzierl, and C. Zenger. A precompiler to reduce the memory footprint of multiscale PDE solvers in C++. *Future Generation Computer Systems*, 26:175–182, January 2010.
- [7] A. Cicchetti, D. D. Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 222–231, Washington, DC, USA, 2008. IEEE Computer Society.
- [8] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous java performance evaluation. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications, OOPSLA '07*, pages 57–76, New York, NY, USA, 2007. ACM.
- [9] J. Hegewald, F. Naumann, and M. Weis. XStruct: Efficient Schema Extraction from Multiple and Large XML Documents. In *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW06 2006)*, page 81, 2006.
- [10] J. Herrington. *Code Generation in Action*. Manning Publications Co., Greenwich, CT, USA, 2003.
- [11] S. Kabisch, D. Peintner, J. Heuer, and H. Kosch. Efficient and Flexible XML-Based Data-Exchange in Microcontroller-Based Sensor Actor Networks. In *Proceedings of the 24th International Conference on Advanced Information Networking and Applications Workshops, WAINA '10*, volume 0, pages 508–513, Los Alamitos, CA, USA, 2010. IEEE Computer Society.
- [12] J. Kangasharju, T. Lindholm, and S. Tarkoma. XML Messaging for Mobile Devices: From Requirements to Implementation. *Computer Networks*, 51:4634–4654, November 2007.
- [13] J. Kangasharju, S. Tarkoma, and T. Lindholm. Xebu: A binary format with schema-based optimizations for xml data. In A. Ngu, M. Kitsuregawa, E. Neuhold, J.-Y. Chung, and Q. Sheng, editors, *Web Information Systems Engineering - WISE 2005*, volume 3806 of *Lecture Notes in Computer Science*, pages 528–535.

Springer Berlin / Heidelberg, 2005.

- [14] M. H. Kay. XML Five Years On: A Review of the Achievements So Far and the Challenges Ahead. In *Proceedings of the 2003 ACM symposium on Document Engineering*, DocEng '03, pages 29–31, New York, NY, USA, 2003. ACM.
- [15] G. Lagorio, M. Servetto, and E. Zucca. Flattening versus direct semantics for featherweight jigsaw. In *FOOL'09, International Workshop on Foundations of Object Oriented Languages*. ACM Press, 2009.
- [16] T. Lindholm and J. Kangasharju. How to edit gigabyte XML files on a mobile phone with XAS, RefTrees, and RAXS. In *Proceedings of Mobiquitous '08*, pages 50:1–50:10, 2008.
- [17] B. McLaughlin. *Java and XML Data Binding*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.
- [18] J. K. Min and C. W. Ahn, J. Y. Chung. Efficient extraction of schemas for XML documents. *Information Processing Letters*, 85, 2003.
- [19] OGC. Sensor Observation Service 1.0.0. *OGC Document*, (06-009r6), 2007.
- [20] C. Pichler, M. Strommer, and C. Huemer. Size Matters!? Measuring the Complexity of XML Schema Mapping Models. *IEEE Congress on Services*, 0:497–502, 2010.
- [21] E. Rahm. Towards large-scale schema and ontology matching. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, Data-Centric Systems and Applications, pages 3–27. Springer Berlin Heidelberg, 2011.
- [22] A. Tamayo. *XML Data Binding for Geospatial Mobile Applications*. PhD thesis, University Jaume I, 2011.
- [23] A. Tamayo, C. Granell, and J. Huerta. Analysing complexity of XML schemas in geospatial web services. In *Proceedings of the 2nd International Conference and Exhibition on Computing for Geospatial Research and Application (COM.Geo 2011)*, pages 17:1–17:9, New York, NY, USA, 2011. ACM.
- [24] A. Tamayo, C. Granell, and J. Huerta. Analysing Performance of XML Data Binding Solutions for SOS Applications. In *Proceedings of Workshop on Sensor Web Enablement 2011 (SWE 2011)*, 2011.
- [25] A. Tamayo, C. Granell, and J. Huerta. Dealing with large schema sets in mobile SOS-based applications. In *Proceedings of the 2nd International Conference and Exhibition on Computing for Geospatial Research and Application (COM.Geo 2011)*, pages 16:1–16:9, New York, NY, USA, 2011. ACM.
- [26] A. Tamayo, P. Viciano, C. Granell, and J. Huerta. Empirical Study of Sensor Observation Services Server Instances. In S. Geertman, W. Reinhardt, and F. Toppen, editors, *Advancing Geoinformation Science for a Changing World*, volume 1 of *Lecture Notes in Geoinformation and Cartography*, pages 185–209. Springer Berlin Heidelberg, 2011.
- [27] A. Tamayo, P. Viciano, C. Granell, and J. Huerta. Sensor Observation Service Client for Android Mobile Phones. In *Proceedings of Workshop on Sensor Web Enablement 2011 (SWE 2011)*, 2011.
- [28] S. Tarkoma, J. Kangasharju, T. Lindholm, and K. Raatikainen. Fuego: Experiences with Mobile Data Communication and Synchronization. In *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pages 1–5, sept. 2006.
- [29] R. A. Van Engelen and K. A. Gallivan. The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '02*, pages 128–, Washington, DC, USA, 2002. IEEE Computer Society.
- [30] A. Villegas and A. Olivé. A method for filtering large conceptual schemas. In *Proceedings of the 29th international conference on Conceptual modeling*, ER'10, pages 247–260, Berlin, Heidelberg, 2010. Springer-Verlag.
- [31] W3C. XML Schema Part 1: Structures Second Ed., 2004. Available from: <http://www.w3.org/TR/xmlschema-1>.
- [32] W3C. XML Schema Part 2: Datatypes Second Ed., 2004. Available from: <http://www.w3.org/TR/xmlschema-2>.
- [33] W3C. Efficient XML Interchange (EXI) Format 1.0, 2011. Available from: <http://www.w3.org/TR/exi>.
- [34] J. White, B. Kolpackov, B. Natarajan, and D. C. Schmidt. Reducing application code complexity with vocabulary-specific XML language bindings. In *Proceedings of the 43rd annual Southeast regional conference - Volume 2*, ACM-SE 43, pages 281–287, New York, NY, USA, 2005. ACM.
- [35] E. Wilde. XML Technologies Dissected. *IEEE Internet Computing*, 7:74–78, September 2003.
- [36] E. Wilde and R. J. Glushko. XML fever. *Communications of the ACM*, 51:40–46, July 2008.