A Model-Driven Solution for Financial Data Representation Expressed in FIXML

Vahdat Abdelzad Hamoud Aljamaan Opeyemi Adesina
Miguel A. Garzon Timothy C. Lethbridge
University of Ottawa
School of Electrical Engineering and Computer Science,
Ottawa, Canada
{v.abdelzad,hjamaan,oades013,mgarzon}@uottawa.ca, tcl@eecs.uottawa.ca

In this paper, we propose a solution based upon Umple for data transformation of Financial Information eXchange protocol (FIXML). The proposed solution includes syntactic and semantic analysis and automatic code generation. We discuss our solution based on development effort, modularity, complexity, accuracy, fault tolerance, and execution time factors. We have applied our technique to a set of FIXML test cases and evaluated the results in terms of error detection and execution time. Results reveal that Umple is suitable for the transformation of FIXML data to object-oriented languages.

1 Introduction

Accuracy of information elicited via financial data processing is crucial to decision makers and portfolio managers in financial domains [12]. Achieving this goal for huge volume of data might be difficult or impossible without automated, dependable, flexible, and scalable implementation solutions. Model-based design and automated code generation methods [7, 11], thereby provide inter-connected partial solutions to developing these systems with minimum effort and defects. Proponents of these methods [6, 9, 7] argued that they tend to deliver better quality artifacts because of their promises of higher productivity, reduced turn-around times, increased portability, and elimination of manual coding errors.

Hence, this paper provides a transformation solution to financial transactions expressed in a FIXML format. Our transformation approach reverse engineers FIXML data into Umple model which is translated later into targeted object-oriented languages. In our transformation, Umple is seen as M1 level in which Umple classes representing the FIXML schema. Umple [1, 2] is an open-source model-oriented language we adopted for the FIXML transformation contest [10]. Proposed solution allows us to have a real-time graphical visualization of FIXML documents, which is done without code generation, in the form of a class diagram. Input FIXML documents can be processed in three environments including UmpleOnline [4], Umple Eclipse plugin, and Umple command-line tool [5]. The results obtained from the test cases show that the generated code is syntactically and semantically accurate and robust.

The rest of this paper is organized as follows. In Section 2, we present why Umple has been chosen for this transformation. Section 3 describes our solution based upon parsing, analysis, and code generation. We will focus on the evaluation of our work and results in Section 4. Finally, we will present the conclusions in Section 5.

Submitted to:
TTC 2014

2 Why Umple?

Umple [1, 2] is an open-source model-oriented language which we have adopted for FIXML transformation contest [10]. Our reasons for choosing Umple are as follows. Firstly, the lightweight capabilities of Umple allow modelers and programmers to seamlessly build applications by having a coding layer within the textual model, which is impossible with just modelling solutions [2]. Secondly, Umple has been developed with a focus on three key qualities named usability, completeness, and scalability. These are prerequisite to any successful tools for generating code from a plethora of data, which is usually generated, and often require processing from financial domains. Thirdly, the integration of FIXML to Umple only requires us to define a grammar to parse FIXML documents and create instances of its meta-model. The parser analyses the input text statically against the defined FIXML grammar. Upon successful static analysis, Umple constructs the internal model of the input as an instance of its own metamodel which is then used to generate the target languages. Fourthly, Umple has already supported code-generation for several object-oriented programming languages. Last but not least, it allows us to visualize the corresponding UML class diagram with attributes and associations between them. This diagram helps to visualize FIXML documents automatically. Umple's architecture is presented in Figure 1.

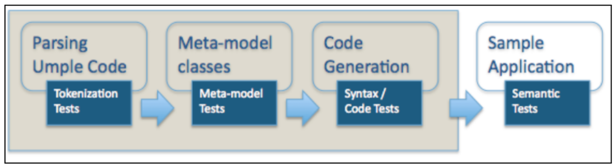


Figure 1: The components of the Umple System.

3 Our solution to the FIXML challenge

To address the challenge, we added an extension to Umple to parse FIXML documents and to process them such that they become instances of Umples own internal metamodel. We use Umples mixin capability to inject the algorithm for analysis of the FIXML input into Umple. The mixin capability helps us not to alter base Umple code but to create the FIXML extension as a separate concern. The Umple mixin mechanism automatically adds the algorithm to the core of Umple.

The first step in our process is to create a valid model from a FIXML document. To achieve this, we need to perform a syntactic and sematic validation of FIXML documents. We validated FIXML documents in two phases. In the first phase, our parser verifies that we have a syntactically valid FIXML document. Then, it produces an internal syntax tree but does not cover semantic checking yet. In the second phase, we do semantic checking for FIXML documents. This validates that we have the same opening and ending tag names. In the second step of having a valid model, Umple meta-model which adds semantic constraints guarantees that we have a valid model and also generates completely valid code for target programming languages.

For syntactic validation, we have defined a set of grammars to parse FIXML documents. The FIXML grammar can be accessed at [3] . Umple has its own EBNF syntax which has special features adapted to processing source that contains multiple languages.

In our solution, we consider tag attributes to be Umple attributes for the model. In the process of analysis, we detect the type of attributes (Integer, Double, and String) and use the correct Umple types for these attributes. On the other hand, whenever we are unable to detect correct types, we assigned a String type. With this we are able to have a correct and robust model and code generation. We also are able to detect the errors in the values of attributes. Moreover, we automatically create related set and get methods for those attributes. We defined attributes with private visibility and generated automatically related set and get methods so as to support data encapsulation. For example, Listing 1 shows a FIXML document in which there is a tag with three attributes. According to the values of attributes, we have two integer attributes and a float attribute. The generated code for the FIXML document in Listing 1 is represented in Listing 2. We removed here set and get methods and other codes (such as constructors, delete, toString etc.) due to space limitation. All generated code can be obtained online through UmpleOnline [5].

```
<FIXML> <Order ClOrdID="123456" Side="2" Px="93.25"> </Order></FIXML>
```

Listing 1: A sample FIXML document

```
class Order{
   private int ClOrdID,  Side;
   private double Px;
   //The rest of code }
```

Listing 2: Java code with proper attribute types

In [10], Lano et al. used an instance variable in generated code for every nested tag in FIXML documents. This approach is also applied to the nested tags with the same name (which results in the same objects). Listing 3, for example, shows three nested tags with the same name called Pty. The generated code for Java according to the solution proposed in [10] is shown in Listing 4. In Listing 4, we can see that there are three instance variables and a constructor with three parameters. This approach is not correct for large FIXML documents and also it does not have a good code implementation for associations in model-driven development. In fact, when we have a large FIXML document with a tag which has more than 255 nested tags, this approach will not work. According to the solution in [10], we should add all of those object instances as parameters to the related class constructors. However, it is impossible because there is a limitation on the number of parameters in programming languages (e.g. limitation of 255 words for method parameters in Java).

```
<PosRpt>
   <Pty ID="OCC" R="21"/> <Pty ID="99999" R="4"/> <Pty ID="C" R="38"/>
</PosRpt>
```

Listing 3: A sample FIXML document

```
class PosRpt {
   Pty Pty_object_1 = new Pty("OCC","21");
   Pty Pty_object_2 = new Pty("99999","4");
   Pty Pty_object_2 = new Pty("C","38");
   PosRpt (Pty Pty_1, Pty Pty_2, Pty Pty_3){
        this.Pty_object_1 = Pty_1;
        this.Pty_object_2 = Pty_2;
        this.Pty_object_3 = Pty_3;
   }
   PosRpt (){ } }
```

Listing 4: Java code generated by the solution in [10]

We have addressed this with the concept of association in the model and arrays as inputs for those same objects in the implementation. Listing 5 shows our generated code in which we have just an instance variable and a constructor with a parameter. With this, we resolved the limitation related to the number of parameters in programming languages. In the same vein, we have just an instance variable which helps us not to lose the model-driven meaning of associations even in the code level. It means that we have an instance variable for each association without worries about multiplicity.

```java
class PosRpt{
   private List<Pty> Pty_Object;
   public PosRpt(Pty... allPty_Object)
   {
      Pty_Object = new ArrayList<Pty>();
      boolean didAddPty_Object = setPty_Object(allPty_Object);
   }
   public PosRpt()
   {
      Pty_Object.add(new Pty("OCC", 21));
      Pty_Object.add(new Pty("99999", 4));
      Pty_Object.add(new Pty("C", 38));
   } //the rest of code
}
```

Listing 5: Java Code generated using our approach

4 Results and Evaluation

In this section, we present the results and evaluation of our implementation. The code generated from any given FIXML documents conforms to their native syntax and semantics. We achieved syntactic conformance by invoking static analyzer embedded in Umple compiler. With this approach, we were able to uncover errors and modify our implementation to ensure syntactic correctness of the generated code. In the same vein, we have adopted the concept of associations in order to preserve semantics as expected. With Umple, creation of links by associations ensures that unique names are created for every instance variables of the same class and preserves the underlying semantics.

We raised the level of abstraction, and minimized development time as well as complexity for future changes. We achieved this with the aid of Umple, which is a level higher than general purpose programming languages, for developing our solution. We performed model-driven development and automatic code generation for the solution. This has been achieved with the minimum effort and belief that future extension or modification will require minimum effort too. The approximate development effort for implementation, testing and debugging is 5 man-hour.

The solution is robust and detected malformed FIXML documents provided as test cases [10]. The solution parses test cases #1, #2, #5, and #6 but the remaining set of test cases are considered as malformed documents. The parser specifies exactly the tag which includes a sub-tag with errors but it is unable to show the exact address of the sub-tag. We have been working on a new parsing engine to solve this issue. Since, Umple has been developed in a modular way, this modification will not have any side-effect in the functionality of our solution. Our solution also provides a real-time graphical visualization for FIXML documents. As shown in Figure 2, it can be visualized as a UML class diagram with attributes and associations between objects (right pane). This is done without code generation so it is independent of target object-oriented languages.

We have instrumented our compiler with a Timer to measure the time taken to process an input file and produce the target code. Specifically, the Timer measures the time in ms (System.currentTimeMillis()) taken to 1) parses an input file 2) to analyze and build an instance of the Umple metamodel 3) to generate source codes. Table 1 summarizes the executions times in milliseconds, for each of the eight FIXML test cases. It shows that our technique gives good performance results even for larger inputs, as is the case for the test #8. The tests were executed on a machine exhibiting the following characteristics: Intel Core i5-2400 CPU @ 3.10GHz, RAM: 8.00 GB, Win 8 - 64 bits, JRE 7.

5 Conclusions

In this paper, we proposed and implemented a solution for automatic object-oriented code generation for financial data representation expressed in FIXML. In order to achieve this, we utilized Umple which includes mechanisms for parsing, analysis, and automatic code generation. Extending Umple grammar to support FIXML satisfied the requirement for accurate syntactic and semantic processing of FIXML documents and provision of a flexible way for ongoing modification. Furthermore, the solution provides a real-time visualization for FIXML documents without code generation.

References

[1] Omar Badreddin (2010): *Umple: a model-oriented programming language*. 2010 ACM/IEEE 32nd International Conference on Software Engineering 2, pp. 337–338, doi:10.1145/1810295.1810381.

[2] Omar Badreddin, Andrew Forward & Timothy C Lethbridge (2012): *Model oriented programming: an empirical study of comprehension*. In: Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '12, IBM Corp., pp. 73–86. Available at `http://dl.acm.org/citation.cfm?id=2399776.2399784`.

[3] CRuiSE: *FIXML Grammar in Umple*. Available at `https://code.google.com/p/umple/source/browse/trunk/cruise.umple/src/umple_fixml.grammar`.

[4] CRuiSE: *Umple Online*. Available at `http://try.umple.org`.

[5] CRuiSE: *Umple tools*. Available at `http://cruise.eecs.uottawa.ca/umple/UmpleTools.html`.

[6] Krysztof Czarnecki & Ulrich Eisenecker (2000): *Generative Programming: Methods, Tools, and Application*. Addison-Wesley.

[7] Ewen Denney & Bernd Fischer (2009): *Generating Code Review Documentation for Auto-Generated Mission-Critical Software*. In: Third IEEE International Conference on Space Mission Challenges for Information Technology, IEEE, pp. 394–401, doi:10.1109/SMC-IT.2009.54. Available at `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5226807`.

[8] Robert Grossman & Yunhong Gu (2008): *Data mining using high performance data clouds*. In: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08, ACM Press, New York, New York, USA, p. 920. Available at `http://dl.acm.org/citation.cfm?id=1401890.1402000`.

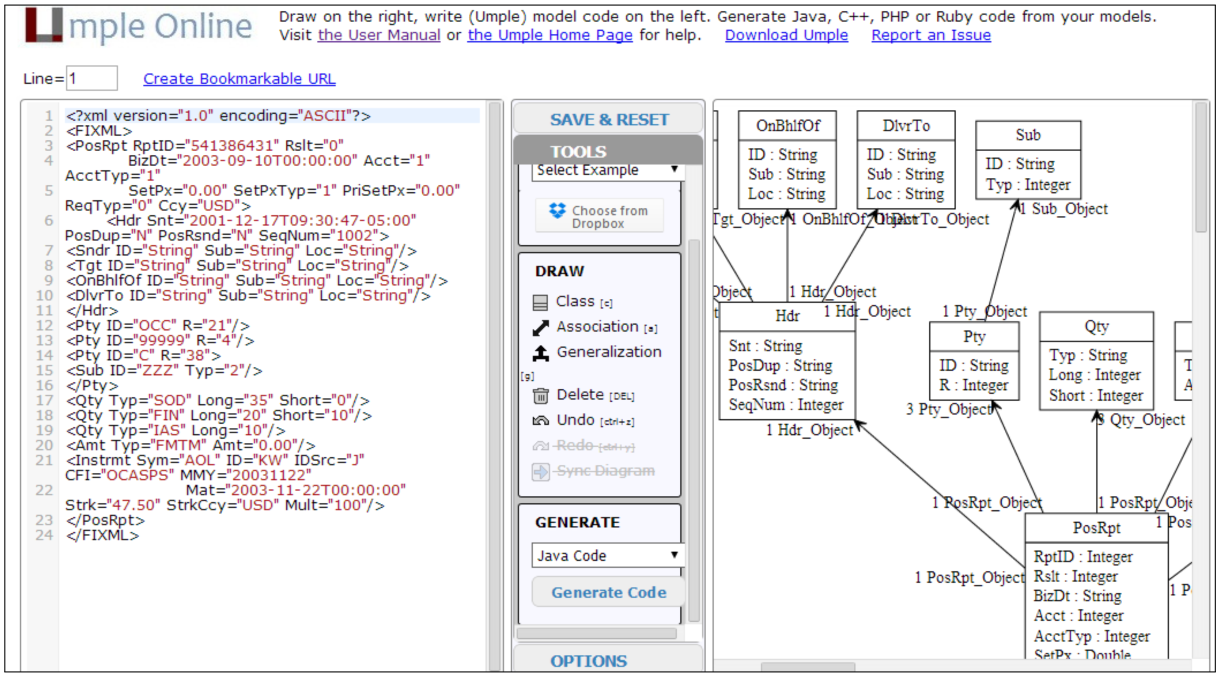
[9] Anneke Kleppe, Jos Warmer & Wim Bast (2003): *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley.

[10] K. Lano, S. Yassipour-Tehrani & K. Maroukian: *Case study: FIXML to Java, C# and C++*. In: Transformation Tool Contest - TTC2014. Available at `https://github.com/TransformationToolContest/ttc2014-fixml/blob/master/case_description.pdf`.

[11] M Boström Nakićenović: *An Agile Driven Architecture Modernization to a Model-Driven Development Solution – An Industrial Experience Report*. International Journal On Advances in Software 5(3–4), pp. 308–322. Available at `http://www.thinkmind.org/index.php?view=article&articleid=soft_v5_n34_2012_13`.

[12] J.W. O'Brien (1970): *How market theory can help investors set goals, select investment managers and appraise investment performance*. Financial Analysts Journal 26(4), pp. 91–103. Available at `http://www.jstor.org/stable/4470707`.

A APPENDIX



Draw on the right, write (Umple) model code on the left. Generate Java, C++, PHP or Ruby code from your models. Visit the User Manual or the Umple Home Page for help. Download Umple Report an Issue

```
<?xml version="1.0" encoding="ASCII"?>
<FIXML>
<PosRpt RptID="541386431" Rslt="0"
        BizDt="2003-09-10T00:00:00" Acct="1"
AcctTyp="1"
        SetPx="0.00" SetPxTyp="1" PriSetPx="0.00"
ReqTyp="0" Ccy="USD">
     <Hdr Snt="2001-12-17T09:30:47-05:00"
PosDup="N" PosRsnd="N" SeqNum="1002">
<Sndr ID="String" Sub="String" Loc="String"/>
<Tgt ID="String" Sub="String" Loc="String"/>
<OnBhlfOf ID="String" Sub="String" Loc="String"/>
<DlvrTo ID="String" Sub="String" Loc="String"/>
</Hdr>
<Pty ID="OCC" R="21"/>
<Pty ID="99999" R="4"/>
<Pty ID="C" R="38">
<Sub ID="ZZZ" Typ="2"/>
</Pty>
<Qty Typ="SOD" Long="35" Short="0"/>
<Qty Typ="FIN" Long="20" Short="10"/>
<Qty Typ="IAS" Long="10"/>
<Amt Typ="FMTM" Amt="0.00"/>
<Instrmt Sym="AOL" ID="KW" IDSrc="J"
CFI="OCASPS" MMY="20031122"
                Mat="2003-11-22T00:00:00"
Strk="47.50" StrkCcy="USD" Mult="100"/>
</PosRpt>
</FIXML>
```

Figure 2: Test case #2 loaded in UmpleOnline

Table 1: Execution time for the eight FIXML test cases

	Execution Time (in ms)							
Component	Case #1	Case #2	Case #3	Case #4	Case #5	Case #6	Case #7	Case #8
Parsing	314	333	324	331	396	607	322	329
Analyzing	17	20	18	20	27	41	17	18
Generating Java Code	198	430	265	294	1543	3572	221	214
Total Time:	529	783	607	645	1966	4220	560	561