Solving the TTC FIXML Case with FunnyQT

Tassilo Horn
horn@uni-koblenz.de

April 15, 2014

Abstract

FunnyQT is a model querying and model transformation library for the functional Lisp-dialect Clojure providing a rich and efficient querying and transformation API. This paper describes the FunnyQT solution to the TTC 2014 FIXML transformation case. It solves the core task of generating Java, C#, and C++ code for a given FIXML message. It also solves the extension tasks of determining reasonable types for the fields of classes, and it generates non-object-oriented C code as well.

1 Introduction

This paper describes a solution of the TTC 2014 FIXML Case [LYTM14] which solves the core task of generating Java, C#, and C++ code for a given FIXML messages. It also solves the extension task of heuristically determining appropriate types for the fields of the generated classes. Furthermore, the extension task to generate non-object-oriented C code has been solved as well. The solution project is available on Github[1], and it is set up for easy reproduction on the SHARE[2] image `Ubuntu12LTS_TTC14_64bit_FunnyQT4.vdi`.

The solution is implemented using FunnyQT [Hor13] which is a model querying and transformation library for the functional Lisp dialect Clojure[3]. Queries and transformations are plain Clojure programs using the features provided by the FunnyQT API. This API is structured into several task-specific sub-APIs/namespaces, e.g., there is a namespace *funnyqt.polyfns* containing constructs for writing polymorphic functions dispatching on metamodel type, a namespace *funnyqt.model2model* containing constructs for model-to-model transformations, a namespace *funnyqt.bidi* containing constructs for bidirectional transformations, and so forth.

As a Lisp dialect, Clojure provides strong metaprogramming capabilities that are exploited by FunnyQT in order to define several *embedded domain-specific languages* (DSL, [Fow10]) for different tasks. For example, the pattern matching, in-place transformation, model-to-model transformation, and bidirectional model transformation constructs are provided in terms of a small, task-oriented DSL each.

FunnyQT currently supports querying and transforming EMF [SBPM08] and JGraLab[4] TGraph models, and support for other modeling frameworks can be added without touching FunnyQT's internals. For both EMF and JGraLab, there is one FunnyQT core namespace (i.e., *funnyqt.emf* and *funnyqt.tg*) providing functions for accessing and manipulation models of that

[1] `https://github.com/tsdh/ttc14-fixml`
[2] `http://is.ieis.tue.nl/staff/pvgorp/share/`
[3] `http://clojure.org`
[4] `http://jgralab.uni-koblenz.de`

kind using the framework's terminology and giving access to any feature provided by that framework. These core namespaces are complemented by a namespace *funnyqt.generic* which provides functions for functionality available on all frameworks in a generic, framework-agnostic manner.

2 Solution Overview

Before digging into implementation details, here is a brief overview of the features of the FunnyQT solution.

1. For every tag name occuring in the XML document, a data type of the same name is generated. For Java, C#, and C++ the data type is a class, for C the data type is a struct.
2. The generated classes/structs contain one field for each XML attribute declared for at least one of the corresponding XML elements. The field type is chosen heuristically based on the XML attribute values. Possible types are timestamps, 32 bit integers, 64 bit integers, floating point numbers, and strings, each represented by a matching concrete type available in the respective programming language. For timestamps which have no literal representation in all four languages, custom parse methods are generated to create a timestamp object from a string.
3. The generated classes/structs contain one field per XML tag of which child elements occur in at least one XML element. If a child of a given tag occurs at most once, the field's type is of the corresponding class/struct (or pointers to that in the case of C++ and C). If a child of a given tag occurs multiple times in at least one element, then the field's type is an array of the corresponding class/struct (or a pointer array in the case of C++ and C).
4. The generated classes/structs contain one field with name `Content` if at least one of the corresponding XML elements contains character contents. The type of the field is again chosen heuristically based on those text contents.
5. The generated classes contain one default constructor which initializes all fields with (one of) the corresponding XML attribute values. Fields that have another class as type are initalized by calling the default constructor of that class. Fields that are arrays of some other class are initialized with the maximum number of objects of that class that occur in one single XML element. Each object is initialized with the default constructor.
   Since there are no constructors for C structs, the generated C code contains a function `Str* make_default_Str()` for any struct `Str` instead.
6. The generated classes contain one constructor that has parameters for initializing all of its fields. In the C code, there's one function `Str* make_Str(...)` instead.
7. The generated C++ classes contain a *destructor* that deletes all pointer and array-pointer typed fields. Likewise, for each struct in the C code there is a function `void free_Str(Str* sp)` that recursively calls the freeing functions of nested struct pointers and arrays of struct pointers.
8. The generated code is pretty idiomatic in all four languages. That is,
   - for Java and C#, there is *one source code file per class*,
   - for C++ and C, there is *one header file* and *one implementation file* for each class/struct,
   - for C++ and C, *headers have proper include guards*,
   - *proper import/using statements* are generated for Java and C#, and similarly proper *proper include preprocessor instructions* are generated for C++ and C,
   - the Java classes are defined in a *package corresponding to the XML file name*,
   - the C# and C++ classes are placed in a *namespace corresponding to the XML file name*,
   - for Java and C++, every class has a *getter/setter pair* for each field,

- and for C# every field is declared with *public auto-implemented get and set properties*.
- for Java and C++ all *fields are private* while the *getters, setters, constructors, and destructors are public*,

9. To ensure *static correctness* (wrt. syntax and typing) of the generated code, it is *compiled*, and in the case of C#, C++, and C *linked together in a library* ready to be used.
10. The transformation allows to create a data model given a single FIXML message as requested by the case description, but it can also be *run on arbitrary many FIXML messages at once*. The idea is that with a reasonable large number of sample FIXML messages, the transformation is able to produce a much more accurate data model. By having more samples, optional attributes and child elements are more likely to be identified. Similarly, child elements which usually occur only once but may in fact occur multiple times are more likely to be identified and lead to the declaration of a corresponding array-valued field instead of just an object-valued field. And finally, the heuristical detection of an appropriate field type benefits from more sample data, too.
    Therefore, running the `generate_code_and_compile.sh` script of the solution project runs the transformation once for each message separately, and then again for all messages at once. In addition to the provided test cases, we have added several other FIXML documents available on the web.

Section A in the appendix on page 16 shows the Java, C#, and C++ classes as well as the C structs and functions that are generated for the FIXML position report message `test2.xml`. It might be a good idea to have a look at that first before digging into the solution description.

3 Solution Description

In this section, the complete transformation specifications for the core and extension tasks are going to be explained. In the listings given in the following, all function calls are shown in a namespace-qualified form to make it explicit in which namespace those functions are defined. Clojure allows to define short aliases for used namespaces in order to allow qualification while still being concise. Table 1 gives an overview of all namespaces used by the solution, and the aliases used for accessing them.

Alias	Namespace	Description
emf	funnyqt.emf	Core EMF API
gen	funnyqt.generic	Generic model access functions
io	clojure.java.io	File IO functions
m2m	funnyqt.model2model	Model-to-Model transformation API
oo	ttc14-fixml.oo	Generated OO model API
poly	funnyqt.polyfns	Polymorphic function API
s	stencil.core	The Stencil templating library
str	clojure.string	String utility functions
tg	funnyqt.emf	Core TGraph API
u	funnyqt.utils	Utility functions (e.g., error handling)
xml	ttc14-fixml.xml	Generated XML model API
xmltg	funnyqt.xmltg	Generic XML to XML-Graph transformation

Table 1: Used Clojure and FunnyQT namespaces with their aliases

All function calls that are not qualified with an alias are calls to functions in the *clojure.core* namespace which is available in any other namespace by default.

3.1 XML to Model

Since handling XML files is a common task, FunnyQT already ships with a namespace *funnyqt.xmltg* which contains a transformation from XML files to a DOM-like TGraph model conforming to the metamodel in Figure 1.

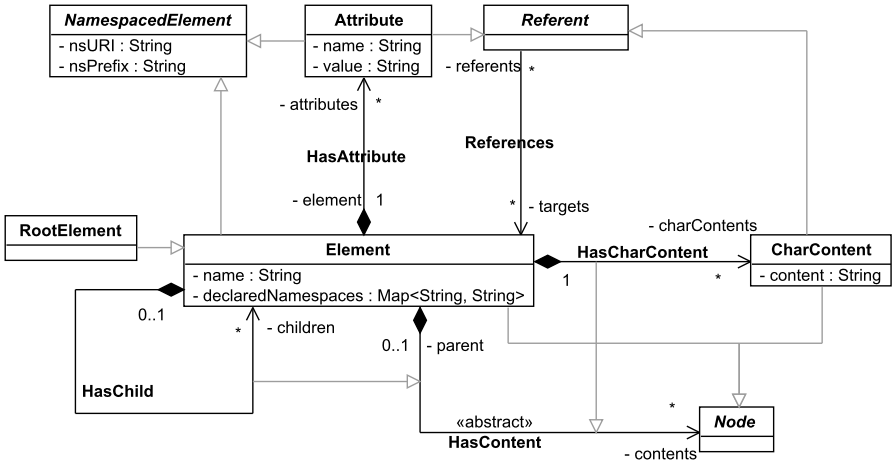


Figure 1: FunnyQT's XML Graph metamodel

It models XML elements, their attributes, text contents, and children. It also supports XML namespaces, and the transformation may be supplemented with additional XML schema information in order to resolve `IDREF` and `IDREFS` references automatically.

Namespaces and references are not important in this case, so a plain call `(xmltg/xml2xml-graph "test2.xml")` suffices to handle core task number 1, i.e., generate an XML model from an XML file. The graph corresponding to the position report message contained in the file `test2.xml` is visualized in the appendix Section A.1.

The `xml2xml-graph` function uses Java's *Stream API for XML* (*StAX*) under the hoods, so XML files that aren't well-formed lead to parsing errors. These are the errors you get for the invalid test files `test7.xml` and `test8.xml`:

```
(xmltg/xml2xml-graph "messages/test7.xml")
;NoSuchElementException ParseError at [row,col]:[14,7]
;Message: The element type "Sndr" must be terminated by the matching end-tag "</Sndr>".
(xmltg/xml2xml-graph "messages/test8.xml")
;NoSuchElementException ParseError at [row,col]:[19,9]
;Message: The element type "Hdr" must be terminated by the matching end-tag "</Hdr>".
```

3.2 XML Model to OO Model

Core task 2 deals with transforming the XML models generated by core task 1 into models conforming to a metamodel suited for object-oriented programming languages. The metamodel used by the FunnyQT solution is shown in Figure 2.

Classes have a name and arbitray many fields. Each field has a name, an initial value, and a type. Types may be classes, builtin types, or arrays. The different builtin types will be used for solving the extension task of determining a sensible type for a field, and instead of creating many separate fields if an XML element contains multiple child elements with a given tag, we create just one field which is an array of the class corresponding to the child element's tag name.

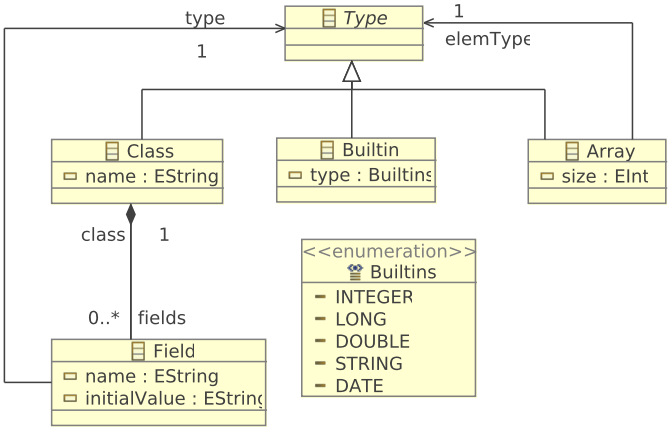


Figure 2: The OO metamodel

The metamodel in Figure 2 is an Ecore model, so the transformation from XML model to OO model actually transforms between two different modeling frameworks: the input is an XML TGraph model, and the output is an OO EMF model.

In the remainder of this section, the transformation will be discussed in details. Before the actual transformation definition, there's a little bit helper code. The following two calls generate metamodel-specific APIs for the XML and the OO metamodels in the namespaces `ttc14-fixml.xml` and `ttc14-fixml.oo` which are referred to by the aliases `xml` and `oo`.

```
1 (tg/generate-schema-functions       "xml-schema.tg"  ttc14-fixml.xml xml)
2 (emf/generate-ecore-model-functions "model/oo.ecore" ttc14-fixml.oo  oo)
```

The generated APIs contain a pair of getter and setter functions for each attribute (e.g., `(xml/name el)` and `(xms/set-name! el val)`), role name accessor functions (e.g., `(xml/->children el)` or `(oo/fields cls)`), and several more.

Next, we define some constants for the values of the `Builtins` enumeration of the OO metamodel.

```
3 (def DATE    (emf/eenum-literal 'Builtins.DATE))
4 (def INTEGER (emf/eenum-literal 'Builtins.INTEGER))
5 (def LONG    (emf/eenum-literal 'Builtins.LONG))
6 (def DOUBLE  (emf/eenum-literal 'Builtins.DOUBLE))
7 (def STRING  (emf/eenum-literal 'Builtins.STRING))
```

The EMF-specific function `emf/eenum-literal` returns the enumeration literal given by a symbol denoting its qualified name.

Then, the actual transformation definition from XML model to OO model follows. In FunnyQT, a model-to-model transformation is specified using the `m2m/deftransformation` macro. It receives the name of the transformation (`xml-graph2oo-model`), and a vector defining input models and output models plus additional parameters. In this case, there is only one single input model `xml`, one single output model `oo`, and no additional parameters.

```
8 (m2m/deftransformation xml-graph2oo-model [[xml] [oo]]
9   ;; rule & function definitions...
10  )
```

Inside such a transformation definition, arbitrary many rule and helper function definitions may occur. The first rule of the transformation is `element2class` shown in the next listing.

```
(^:top element2class
 :from [e '[:and Element !RootElement]]
 :to   [c (element-name2class (xml/name e))])
```

The `^:top` annotation defines this rule as a top-level rule. Such a rule is automatically applied to all matching elements. The `:from` clause restricts the elements `e` this rule is applicable for to those that are of metamodel type `Element` but not of type `RootElement` (which is a subclass of `Element` according to Figure 1). The reason is that we don't want to create a class for the `FIXML` element which is the root element of any FIXML message.

The `:to` clause defines which elements should be created for matching elements. Usually, it would be specified as `:to [x 'SomeClass]` in which case `x` would be a new element of type `SomeClass`. However, in the current case, there is no one-to-one mapping between XML elements and OO classes, because the XML model may contain multiple elements with the same tag name, and there should be exactly one OO class per unique tag name. Therefore, the `:to` clause delegates the creation of class `c` to another rule `element-name2class` providing `e`'s tag name as argument.

The semantics of a rule are as follows. When it is applied to an input element for which its `:from` clause matches, target elements are created according to its `:to` cause. The mapping from input to output elements is saved. When a rule gets applied to an element it has already been applied to, the elements that have been created by the first call are returned instead of creating new elements. That way, calling a rule serves both creation of target elements as well as resolution of input to output elements in terms of traceability.

The next rule is `element-name2class` which is shown below.

```
(element-name2class
 :from [tag-name]
 :to   [c 'Class {:name tag-name}]
 (doseq [[an at av] (all-attributes tag-name)]
   (attribute2field an at av c))
 (doseq [[tag max-child-no] (all-children tag-name)]
   (children-of-same-tag2field tag max-child-no c))
 (when-let [char-conts (seq (all-character-contents tag-name))]
   (character-contents2field char-conts c)))
```

This rule receives as input a plain string, the `tag-name` of an element, and it creates a `Class c` in the target model. The name of the class corresponds to the `tag-name`. According to the rule semantics sketched above and the fact that this rule gets called with the tag name of any element by `element2class`, there will be one target class for every unique tag name.

Following the `:from` and `:to` clauses comes the rule's body where arbitrary code may be written. Here, three other rules `attribute2field`, `children-of-same-tag2field`, and `character-contents2field` are called for all XML attributes, child elements, and character contents[5] of element `e`, respectively. These rules are shown below.

```
(attribute2field
 :from [an at av class]
 :to   [f 'Field {:name an :type at :initialValue av :class class}])
(children-of-same-tag2field
 :from [tag max-child-no class]
 :to   [f 'Field {:class class}]
 (if (= 1 max-child-no)
   (do (oo/set-name! f (str tag "_obj"))
       (oo/->set-type! f (element-name2class tag)))
   (let [array-type (oo/create-Array! oo :size max-child-no :elemType (element-name2class tag))]
     (oo/set-name! f (str tag "_objs"))
     (oo/->set-type! f array-type))))
```

[5]The case description doesn't specify if and how XML character content should be handled. However, without them transforming `test3.xml` and `test4.xml` would lead to classes with no fields at all which doesn't make much sense.

```
34  (character-contents2field
35   :from [char-conts class]
36   :to   [f 'Field {:name "Content" :initialValue (first char-conts)
37                    :type (guess-type char-conts) :class class}])
```

The `attribute2field` rule gets an attribute name `an`, its type `at`, its initial value `av`, and the `class` to which to add the new field `f`.

The `children-of-same-tag2field` rule gets a child tag name `tag`, the maximum number of children having that tag `max-child-no`, and again the `class` to which to add the new field `f`. If there is only one child of a tag, then the field's type is the class corresponding to the tag[6]. If there are multiple children of that tag, then the field's type is array of objects of the class corresponding to the tag.

Lastly, `character-contents2field` receives a collection `char-conts` of character contents and again the `class` to which to add the new field `f`. That new field gets the fixed name "Content", and its type is guessed by a function `guess-type` which is shown in the next listing.

```
38  (guess-type [vals]
39   (let [ts (set (map #(condp re-matches %
40                         #"\d\d\d\d-\d\d-\d\d.*" DATE
41                         #"[+-]?\d+\.\d+"        DOUBLE
42                         #"[+-]?\d+"             (int-type %)
43                         STRING) vals))]
44     (get-or-create-builtin-type
45      (cond
46       (= (count ts) 1)              (first ts)
47       (= ts #{DOUBLE INTEGER})      DOUBLE
48       (= ts #{DOUBLE LONG})         DOUBLE
49       (= ts #{DOUBLE LONG INTEGER}) DOUBLE
50       (= ts #{INTEGER LONG})        LONG
51       :else                         STRING))))
52  (int-type [val]
53   (try (and (Integer/parseInt val) INTEGER)
54        (catch NumberFormatException _
55          (try (and (Long/parseLong val) LONG)
56               (catch NumberFormatException _
57                 STRING)))))
58  (get-or-create-builtin-type
59   :from [t]
60   :to   [bit 'Builtin {:type t}])
```

The `guess-type` function receives a collection `vals` from which to guess an appropriate type. `vals` could either be all character contents of an XML element, or all attribute values of an attribute that occurs in many XML elements of the same tag.

Every given value is checked against a regular expression that determines its type being either a timestamp in ISO 8601 notation, a double value, or an integer value. If neither matches, then `STRING` is used as its type. In case of an integer value, the function `int-type` further determines if the value can be represented as a 32 bit integer, or if a 64 bit long is needed, or if it is so large that it can only be represented as a string.

Lines 45 to 51 pick the type that can be used to represent all values. If all values are guessed to be of the very same type (line 46), then this type is chosen. For multiple numeric types, the respective "largest" type is chosen where `INTEGER` < `LONG` < `DOUBLE`. Else, we fall back to `STRING`.

The picked type is then passed to the rule `get-or-create-builtin-type` which creates a `Builtin` whose `type` attribute is set to the picked type `t`. As a result, the OO model contains at most one `Builtin` element per `Builtins` enumeration literal.

The last part of the transformation are the functions that compute the information about attributes, child elements, and character contents that are passed to the respective rules by the `element-name2class` rule starting in line 12. All three functions have to cater for the fact that

[6]Note how the rule `element-name2class` is called here to resolve the class created for a tag name

elements of the same tag may occur multiple times in the XML document, and each occurence may use a different subset of the attributes and child elements it may have according to the FIXML specification. Therefore, a helper function `elements-of-tag` is defined that given a `tag-name` returns all elements having that tag.

```
61  (all-attributes [tag-name]
62   (->> (elements-of-tag tag-name) (mapcat xml/->attributes) (group-by xml/name)
63        (map (fn [[an as]]
64               [an (guess-type (map xml/value as)) (xml/value (first as))]))))
65  (elements-of-tag [tag-name]
66   (->> (xml/vseq-Element xml) (filter #(= tag-name (xml/name %)))))
67  (all-children [tag-name]
68   (let [child-tag-names (->> (elements-of-tag tag-name) (mapcat xml/->children)
69                              (map xml/name) (into #{}))]
70     (map (fn [ctn] [ctn (max-children-of-tag tag-name ctn)]) child-tag-names)))
71  (max-children-of-tag [tag-name child-tag-name]
72   (reduce (fn [old el]
73             (let [cot (count (filter #(= child-tag-name (xml/name %)) (xml/->children el)))]
74               (if (> cot old) cot old)))
75             0 (elements-of-tag tag-name)))
76  (all-character-contents [tag-name]
77   (->> (elements-of-tag tag-name) (mapcat xml/->charContents)
78        (map xml/content) (remove str/blank?))))
```

`all-attributes` gets a `tag-name`, computes all elements having this tag, computes the union of attributes, and groups them by their name. This results in a sequence of tuples of the form `[an as]` where `an` is the attribute name and `as` is a collection of all attributes with that name. All those tuples are then processed with a function that returns a tuple of the attribute name, the type guessed from all values, and the first attribute's value (which will be used as initial value for the corresponding field). That's exactly the information relevant for the `attribute2field` rule (see call in line 16 and definition starting in line 21).

`all-children` first computes the tag names of all elements that occur as child of an element with tag `tag-name`. For each of them, it computes a tuple of the form `[ctn max-child-no]` where `ctn` is the child tag name, and `max-child-no` is the largest number of children of that child tag that occur together in an element. Again, that's the relevant information for the `children-of-same-tag2field` rule (see call in line 18 and definition starting in line 24)

Lastly, `all-character-contents` returns a sequence containing all non-empty character contents of all elements of tag `tag-name`. Its result is passed to the `character-contents2field` rule (called in line 20 and defined in line 34).

The OO model generated for the the position report message contained in the file `test2.xml` is visualized in the appendix Section A.3.

3.3 OO Model to Code

The third and last step of the overall transformation is to generate code in different programming languages from the OO model created in the previous step. In addition to the core task languages, the FunnyQT solution also generates C code as an extension.

One crucial benefit of FunnyQT being a Clojure library is that we can simply use arbitrary other Clojure and Java libraries for our needs. So for this task, we use the excellent *Stencil*[7] library. Stencil is a Clojure templating library implementing the popular, lightweight *Mustache* specification[8]. The idea of Mustache is that one defines a template file containing placeholders which can be rendered to a concrete file by providing a map where the keys are the placeholder names and the values are the text that should be substituted. There are also placeholders for collections in which case the corresponding value of the map has to be a collection of maps.

[7] `https://github.com/davidsantiago/stencil`
[8] `http://mustache.github.io/`

We'll discuss the solution using the template for Java.

```
1  package {{{pkg-name}}};
2
3  {{#imports}}
4  import {{{imported-class}}};
5  {{/imports}}
6
7  class {{{class-name}}} {
8      {{#fields}}
9      private {{{field-type}}} {{{field-name}}};
10     {{/fields}}
11
12     public {{{class-name}}}() {
13         {{#fields}}
14         this.{{{field-name}}} = {{{field-value-exp}}};
15         {{/fields}}
16     }
17
18     public {{{class-name}}}({{#fields}}{{^first}}, {{/first}}{{{field-type}}} {{{field-name}}}{{/fields}}) {
19         {{#fields}}
20         this.{{{field-name}}} = {{{field-name}}};
21         {{/fields}}
22     }
23     {{#fields}}
24
25     public {{{field-type}}} get{{{field-name}}}() {
26         return {{{field-name}}};
27     }
28
29     public void set{{{field-name}}}({{{field-type}}} {{{field-name}}}) {
30         this.{{{field-name}}} = {{{field-name}}};
31     }
32     {{/fields}}
33 }
```

When we want to render a class named `C` in package `p` that uses the `java.util.Date` class and contains two fields `f1` and `f2` of type `String` and `Date` and initial values "Hello" and `2003-09-10T00:00:00`, we would need to provide the following map to the rendering function.

```
{:pkg-name "p"
 :imports [{:imported-class "java.util.Date"}]
 :class-name "C"
 :fields [{:field-type "String"
           :field-name "f1"
           :field-value-exp "\"Hello\""
           :first true}
          {:field-type "Date"
           :field-name "f2"
           :field-value-exp "Util.parseDate(\"2003-09-10T00:00:00\");"}]}
```

The `:first` key with value `true` needs to be added to the map corresponding to the first field of the class for rendering the parameter list of the non-default constructor. Every parameter which has no `:first` key gets prepended by a comma and a space[9].

The templates for the other languages use the same keys[10], so the essential job of the code generation task is to derive such a map for every class in our OO model that can then be passed to Stencil's rendering function.

This is done using a FunnyQT polymorphic function `to-mustache` whose definition is given below.

[9] See line 26 of the template which uses an inverted mustache section: `{{^first}}, {{/first}}`. Only if there is no `:first` key (or a `:first` key with value `false`) the comma followed by a space is emitted.

[10] The key names are a bit Java specific, but they have analogues in the other languages. For example, for C and C++ the `:imported-class` values in the `:imports` vector are in fact the header files to be included.

```
1 (poly/declare-polyfn to-mustache [el lang pkg])
2
3 (poly/defpolyfn to-mustache oo.Class [cls lang pkg]
4   {:pkg-name pkg
5    :imports (get-imports cls lang)
6    :class-name (oo/name cls)
7    :fields (mark-first-field (map #(to-mustache % lang pkg) (oo/->fields cls)))})
8
9 (poly/defpolyfn to-mustache oo.Field [f lang pkg]
10  {:field-type (field-type (oo/->type f) lang)
11   :field-name (oo/name f)
12   :field-value-exp (field-value-exp f lang)
13   :uscored-field-name (str "_" (oo/name f))
14   :plain-field-type (let [t (oo/->type f)]
15                       (gen/type-case t
16                         'Array (oo/name (oo/->elemType t))
17                         'Class (oo/name t)
18                         nil))
19   :pointer (gen/has-type? (oo/type f) 'Class)
20   :array (gen/has-type? (oo/type f) 'Array)})
```

A polymorphic function in FunnyQT is a function that dispatches between several implementations based on the metamodel type of its first argument. Thus you can view them as a kind of object-oriented method attached to metamodel classes that may be overridden by subclasses.

Line 1 declares the polymorphic function `to-mustache` and defines that it gets three parameters: an OO model element `el`, the target language `lang`, and the package/namespace name `pkg` in which the class/struct should be generated.

Lines 3 to 7 then define an implementation for elements of the metamodel class `Class`. It returns a literal map where `:pkg-name` is the package name provided as argument, `:imports` is computed by a function `get-imports`, the `:class-name` is the name of the `Class` element `cls`, and `:fields` is computed by applying the `to-mustache` function to all fields of the class (and marking the first as discussed above).

Lines 9 to 20 define the `to-mustache` implementation for elements of metamodel class `Field`. The language-dependent `:field-type` is computed by a function `field-type`, the `:field-name` is the name of the `Field` element, and the `:field-value-exp` used to initialize the field in the default constructor is computed by a function `field-value-exp`.

The additional key `:uscored-field-name` is the field name prefixed with an underscore. This variant is used for fields in C++ and C#, because in these languages, a field must not be named like its enclosing class but there are FIXML elements that have an attribute named like the element tag, e.g., `<Amt Typ="FMTM" Amt="0.00"/>`.

The three further keys `:plain-field-type`, `:pointer`, and `:array` are only needed for the C and C++ templates where they are used to generate a proper destructor or a proper function for freeing a struct pointer recursively.

The next listing shown the `field-type` function returning the appropriate data type name for a given `field` when generating code for language `lang`.

```
21 (def string-type {:java "String", :csharp "string", :cpp "std::string", :c "char*"})
22 (def timestamp-type {:java "Date", :csharp "DateTime", :cpp "std::tm", :c "struct tm"})
23
24 (defn object-type [class lang]
25   (str (oo/name class) (when (#{:c :cpp} lang) "*")))
26
27 (defn field-type [type lang]
28   (gen/type-case type
29     'Class   (object-type type lang)
30     'Builtin (condp = (oo/type type)
31                DATE    (timestamp-type lang)
32                DOUBLE  "double"
33                INTEGER (case lang (:c :cpp) "long"      "int")
34                LONG    (case lang (:c :cpp) "long long" "long")
35                STRING  (string-type lang))
```

```
36      'Array   (str (field-type (oo/->elemType type) lang)
37                   (if (#{:c :cpp} lang) "*" "[]"))))
```

In case the field's type is a class, that classes' name is used. In the case of C and C++, an asterisk is prepended, because there pointers to that class are used.

For the builtin types distinctions have to be made for integral numbers, dates and strings. While in Java and C# an integer is 32 bit and a long is 64 bit, the size is implementation-dependent for C and C++. It could also be 32 and 64 bit, but 16 and 32 bit are also allowed. Therefore, we use the type `long` for integers and `long long` for long integers which are guaranteed to be at least 32 and 64 bit wide by the C99 standard. For strings and timestamps, lines 21 and 22 simply define constant maps from language keys to appropriate type names. In Clojure, a map is a function of its keys, so `(string-type :cpp)` returns `"std::string"`.

In the case the field's type is an array, the array's element type name gets prepended with either an asterisk (for C and C++) or a pair of brackets (Java and C#).

The probably most complex function of the code generation is `field-value-exp` shown in the next listing. It generates the expression used to initialize the given `field` in the programming language `lang`.

```
38 (defn field-value-exp [field lang]
39   (let [type (oo/->type field)
40         val (oo/initialValue field)]
41     (gen/type-case type
42       'Class (if (= lang :c)
43                (str "make_default_" (oo/name (oo/->type field)) "()")
44                (str "new " (oo/name (oo/->type field)) "()"))
45       'Builtin (condp = (oo/type type)
46                  DATE   (case lang
47                           :cpp (str "Util::parseDate(\"" val "\")")
48                           :c   (str "parseDate(\"" val "\")")
49                           (str "Util.parseDate(\"" val "\")"))
50                  STRING (str "\"" val "\"")
51                  LONG   (str val "L")
52                  val)
53       'Array (if (= lang :c)
54                (format "(%s) make_pointer_array(%s, %s)"
55                        (field-type type lang)
56                        (oo/size type)
57                        (str/join ", " (for [i (range (oo/size type))]
58                                         (format "make_default_%s()"
59                                                 (oo/name (oo/->elemType type))))))
60                (format "new %s[%s] {%s}"
61                        (field-type (oo/->elemType type) lang)
62                        (if (= lang :cpp) (oo/size type) "")
63                        (str/join ", " (for [i (range (oo/size type))]
64                                         (format "new %s()"
65                                                 (oo/name (oo/->elemType type)))))))))))
```

Fields that have some other class as their type, the value expression is usually `new OtherClass()`, i.e., a call to that class' default constructor. In C, however, there is no `new` and no constructors, so the C template generates a `make_default_StructName()` function for which a call is returned here.

For fields of builtin types several distinctions have to be made. Dates are always parsed by an auxilliary `parseDate()` method contained by some `Util` class. Strings have to be enclosed in double quotes. Long literals need to be suffixed with `L`, else a long literal that's larger than the largest integer will lead to a parsing error. Lastly, integer and double numbers can be emitted as-is, i.e., as they occured in the XML document.

For array fields, the value expression is usually a literal array initializer form containing calls to the default constructors of the class corresponding to the array's element type. Again, since there's no `new` and no constructors in C, a call to a generic `make_pointer_array()` varargs function is generated. This function allocates memory for a pointer array of the given size n plus one, assigns the pointers given as arguments to the indices 0 to $n-1$, and index n is set to `NULL`. This

`NULL`-termination of the arrays allows us to generate proper freeing functions that recursively free the memory held by a struct of the data model. A possibly better approach was to use some third-party memory management library like *talloc*[11] instead which wouldn't require such "tricks", but we decided to only use the standard C library.

The next listing contains the `file-ending` function which has implementations for arity one and two. If given only a `lang` keyword, it returns the typical file name suffix for that language where for C and C++ it returns the suffix of a header. To get the file name ending for the implementation file instead, one needs to call `file-ending` with additional argument `impl` set to `true`.

```
(defn file-ending
  ([lang] ({:java "java", :csharp "cs", :cpp "hpp", :c "h"} lang))
  ([lang impl]
     (if impl
       (case lang :cpp "cpp" :c "c")
       (file-ending lang))))
```

The next listing shows the `get-imports` function which calculates the classes/files/namespaces of a class `cls` that need to be imported/included/used in the language `lang`.

```
(defn get-imports [cls lang]
  (letfn [(includes-from-type [t]
            (gen/type-case t
              'Class   (when (#{:c :cpp} lang)
                         [(format "\"%s.%s\"" (oo/name t) (file-ending lang))])
              'Builtin (condp = (oo/type t)
                         DATE   (case lang
                                      :java   ["java.util.Date"]
                                      :csharp ["System"]
                                      :cpp    ["<ctime>" "\"Util.hpp\""]
                                      :c      ["<time.h>" "\"Util.h\""])
                         STRING (when (= lang :cpp) ["<string>"])
                         nil)
              'Array   (concat (when (#{:c :cpp} lang)
                                 (includes-from-type (oo/->elemType t)))
                               (when (= lang :c)
                                 ["\"Util.h\""])))])
    (map (fn [import] {:imported-class import})
         (set (mapcat includes-from-type (gen/adjs cls :fields :type))))))
```

In the case of C and C++, every implementation file of a class needs to include the corresponding header. Classes that have fields of a date type need to import special classes/namespaces/headers in all four languages, and for C++, there needs to be an include even for being able to use the standard string type. For arrays, the C and C++ versions need to include the header corresponding to the element type, and for C also the `Util.h` header has to be included because of the call to the generic pointer-array creation function.

The main driver of the code generation is the function `oo2code` given below.

```
(defn oo2code [oo pkg lang]
  (let [dir (format "results/%s/%s" (name lang) pkg)]
    (.mkdirs (io/file dir))
    (spit (format "%s/Util.%s" dir (file-ending lang))
          (s/render-file (format "templates/Util.%s" (file-ending lang))
                         {:pkg-name pkg}))
    (when (#{:c :cpp} lang)
      (spit (format "%s/Util.%s" dir (file-ending lang true))
            (s/render-file (format "templates/Util.%s" (file-ending lang true))
                           {:pkg-name pkg})))
    (doseq [cls (oo/eall-Classes oo)]
      (spit (format "%s/%s.%s" dir (oo/name cls) (file-ending lang))
            (s/render-file (format "templates/class.%s" (file-ending lang))
                           (to-mustache cls lang pkg)))
```

[11] `http://talloc.samba.org/`

```
105       (when (#{:c :cpp} lang)
106         (spit (format "%s/%s.%s" dir (oo/name cls) (file-ending lang true))
107               (s/render-file (format "templates/class.%s" (file-ending lang true))
108                              (to-mustache cls lang pkg)))))))
```

It the receives the OO model `oo`, a package name in which to generate the code, and the target language `lang`. The files will be generated in a directory `results/<lang>/<pkg>/`.

Lines 94 to 100 generate the utility class file containing the date parsing methods (for C and C++, split in header and implementation) and the generic pointer-array creation function in the case of C.

Then, lines 101 to 108 generate a class file for every class contained in the OO model. Again, in the case of C and C++, actually one header and one implementation file is emitted per class.

4 Evaluation

Complexity. The complexity according to the evaluation criteria should be measured as the sum of number of operator occurrences and feature and entity type name references in the specification expressions.

The FunnyQT solution contains about 300 expressions (function calls, conditional expressions, etc.), 24 metamodel type references, and 18 property references resulting in a complexity of 342. So it is probably quite complex, but most of its complexity is a result of that it does much more than what was required, e.g., guess appropriate field types, generate C code, generate destructors and freeing functions for C++ and C, separate header and implementation files for C++ and C, generate getters and setters, etc.

Accuracy. Accuracy should measure the degree of syntactical correctness of the generated code, and the degree of how well the generated code matches the source FIXML messages.

The FunnyQT solution has a very high accuracy. The code is correct for all four languages and compiles without warnings. It also matches the source FIXML messages very well. Especially creating array fields for XML child elements that occur multiple times is much better than creating numbered fields. Also, guessing appropriate types for the fields instead of always using string improves the usefulness of the generated code. Finally, that the transformation can be run on an arbitrarily large sample of FIXML messages in one go improves the accuracy of the generated classes even more.

Development effort. Developing the solution has been a on-and-off effort. Prototyping an initial solution only for Java took about 4 person-hours. Extending that to C# was rather easy, however when C++ (and even worse C) with their pointers and split in headers and implementation files came into play, a major refactoring of the code generation code was needed. So all in all, the overall development time can be estimated with about 12 person-hours.

Fault tolerance. Since FunnyQT's generic `xml2xml-graph` transformation uses Java's StAX API internally, the fault tolerance is high. Documents that are not well-formed lead to errors like the ones printed in Section 3.1 for the broken test files `test7.xml` and `test8.xml`.

If the XML documents also declare or reference a DTD or an XML schema, then the transformation can also use a validating StAX parser that also throws an exception when the document while being well-formed does not conform to the document's schema.

Execution time. Table 2 shows the execution times for the different test files. The times have been gathered on SHARE.

The column **XML2OO** contains the times for parsing the XML document and generating an XML graph from it (Section 3.1), plus the time needed to transform that into an OO model (Section 3.2).

The columns **Java**, **C#**, **C++**, and **C** contain the time needed to generate code in the respective language from the OO model (Section 3.3).

The column **Overall** is the overall time needed for parsing the XML document to a graph, transforming that to an OO model, and generating code in all four languages.

Message	XML2OO	Java	C#	C++	C	Overall
`test1`	19.1	6.8	5.2	18.6	15.3	69.7
`test2`	47.8	17.5	25.6	49.9	35.7	179.2
`test3`	19.1	13.8	18.2	35.7	32.2	130.4
`test4`	33.1	22.3	26.9	51.1	42.6	178.7
`test5`	70.5	13.8	15.4	31.2	37.4	178.4
`test6`	88.0	14.9	17.0	30.0	31.1	184.2
`batch_message`	31.1	83.0	31.2	85.9	54.7	305.5
`new_order_single`	30.8	5.8	5.4	12.6	20.9	78.3
`trade_subm_allocs`	30.8	8.9	7.4	21.8	21.8	93.5
`trade_subm_allocs_ack`	32.1	12.7	15.5	30.4	31.5	128.2
`trade_subm_clearing`	22.2	9.0	8.0	24.0	22.6	88.7
All 11 at once	364.1	42.0	39.4	86.9	81.3	618.5

Table 2: Execution times for the test FIXML messages measured on SHARE (in milliseconds)

The FIXML messages `test1` to `test6` are the test files provided in the case project[12]. The other five messages are some additional ones that can be found on the web[13].

The last line shows the time needed for parsing all eleven FIXML messages into one single XML graph, transforming that to an OO model, and then emitting code in the four languages.

Modularity. The `xml-graph2oo-model` consists of $r = 6$ rules, one marked as a top-level rule. All other rules are called explicitly which gives $d = 5$ or $d = 6$ dependencies, depending on if the marking of a top-level rule counts as ordering dependency, too. Thus, its modularity according to the formula $Mod = 1 - \frac{d}{r}$ is between zero and $0.1\overline{6}$.

The code generation is implemented using 10 functions that call each other. Since `to-mustache` and `field-type` are recursive, and `field-type` is called from two different functions, we can count 12 call dependencies. Thus, the modularity $Mod = 1 - \frac{d}{r} = 1 - \frac{12}{10} = -0.2$.

Abstraction level. FunnyQT model-to-model transformations like `xml-graph2oo-model` are mostly declarative, but rules may also contain functional/imperative code, and the rules defined here do so.

The code generation is split into declarative templates that express the contents of the source code files including their formatting, and functions that derive a map of template placeholder keywords to the values that have to be filled in for each class. Those functions are all pure functional, i.e., they have no side-effects and are referentially transparent.

[12] `https://github.com/TransformationToolContest/ttc2014-fixml`

[13] `http://www.cmegroup.com/confluence/display/EPICSANDBOX/CME+ClearPort+API+-+FIXML+Message+Samples`

Thus, the abstraction level of the FunnyQT solution is about medium.

5 Conclusion

In this paper, the FunnyQT solution to the TTC 2014 FIXML case has been discussed. It solves the core task of generating Java, C#, and C++ code for a given FIXML message, and the transformation is structured as requested by the case description, i.e., there is a generic XML-to-XMLModel transformation (built into FunnyQT), there is an XMLModel-to-OOModel transformation, and there is an OOModel-to-Text transformation.

The solution also solves the extension tasks of determining appropriate field types from the attribute values occuring in the FIXML messages and of generating non-object-oriented C code.

The extension task of generating the data model from the FIXML schema instead from instance messages has not been tackled. Of course, this would be the only sane approach to generating a correct data model. But the FIXML specification is very large (more than 50 individual XML schemas) and complex, so the effort needed to write such a transformation, preferably as a generic XMLSchema-to-OOModel transformation, is much too high for a TTC case. However, the possibility to run the FunnyQT transformation on an arbitrary large set of sample FIXML messages at once offers a different possibility to generate a suitable data model. When being run on a set of samples containing messages of all possible types with representative attribute values including minima and maxima and children elements, then the data model generated by the FunnyQT solution should be very similar to what could be generated from the FIXML XSD.

Nevertheless, the FunnyQT solution adds several more features that haven't been requested like generation of getters and setters, separation of headers and implementation files for C and C++, and the generation of destructors for C++ and freeing functions for struct pointers in C.

The generated code in all four languages can be compiled and linked as-is without warnings or special compiler settings.

Given its amount of features, the solution is quite concise. The complete `xml-graph2oo-model` transformation consists of 78 lines of code, and the code generation is implemented in 108 lines of code plus additional 206 lines of Java/C#/C++/C code and Mustache markup in 12 templates.

References

[Fow10] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[Hor13] Tassilo Horn. Model querying with funnyqt - (extended abstract). In Keith Duddy and Gerti Kappel, editors, *ICMT*, volume 7909 of *Lecture Notes in Computer Science*, pages 56–57. Springer, 2013.

[LYTM14] K. Lano, S. Yassipour-Tehrani, and K. Maroukian. Case study: Fixml to java, c# and c++. In *Transformation Tool Contest 2014*, 2014.

[SBPM08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2 edition, 2008.

A Transformation of a Position Report Message

In this section, the stepwise outcomes of transforming a position report message (`test2.xml`) are illustrated.

The FIXML document itself is printed in Section A.1.

Section A.2 shows its representation as TGraph conforming to the XML metamodel shown in Figure 1. This part of the overall transformation has been discussed in Section 3.1.

Section A.3 shows the OO model conforming to the metamodel shown in Figure 2 which is generated by the `xml-graph2oo-model` transformation discussed in Section 3.2.

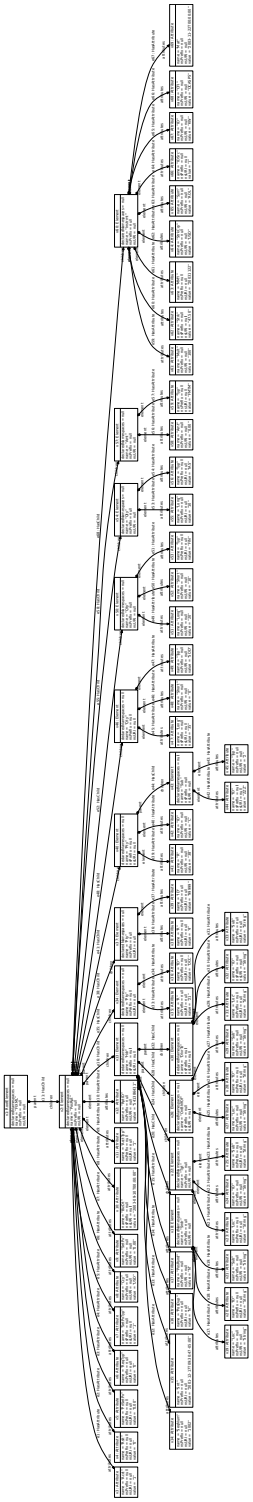
Finally, the sections A.4, A.5, A.6, and A.7 show the source code files for the `PosRpt` class and the `Util` class which contains helpers for the data model classes. For Java and C#, there is only one source code file for the `PosRpt` class whereas for C++ and C, the class/struct is declared in a header file and its definition is held in a separate implementation file. It should be noted that all source code files are printed here exactly as produced by the transformation. No additional formatting has been done, and they all compile without warnings using standard compilers for the languages, i.e., `javac` from the OpenJDK project[14] for Java, `mcs` from the Mono project[15] for C#, and `g++`/`gcc` from the GNU Compiler Collection[16] for C++ and C. However, the C++ code uses extended initializer lists which are new in the C++11 standard, so a `-std=c++0x` (or `-std=c++11`) has to be added to the `g++` call in order not to get warnings.

A.1 The Position Report as XML document

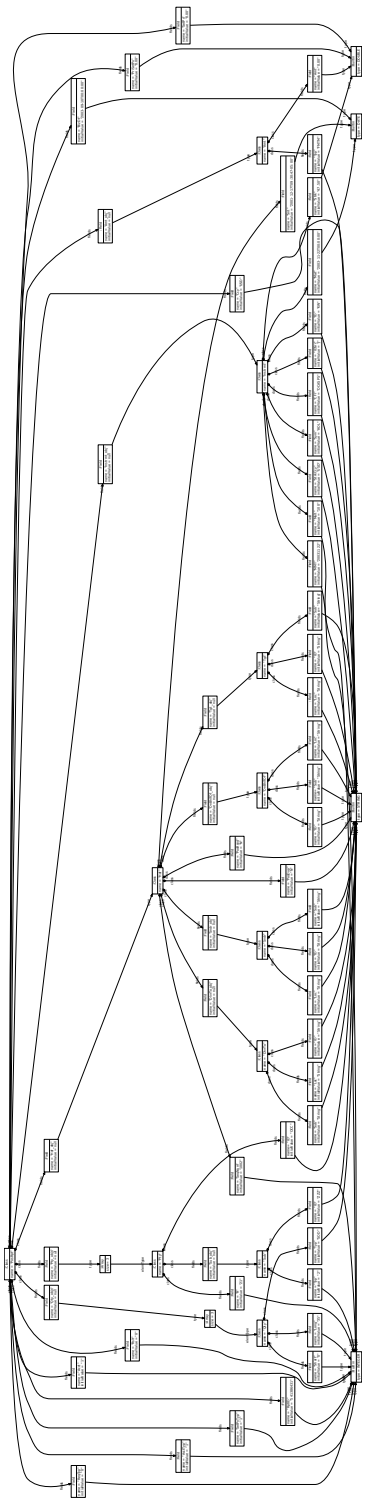
```
<?xml version="1.0" encoding="ASCII"?>
<FIXML>
  <PosRpt RptID="541386431" Rslt="0"
          BizDt="2003-09-10T00:00:00" Acct="1" AcctTyp="1"
          SetPx="0.00" SetPxTyp="1" PriSetPx="0.00" ReqTyp="0" Ccy="USD">
    <Hdr Snt="2001-12-17T09:30:47-05:00" PosDup="N" PosRsnd="N" SeqNum="1002">
      <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>
```

[14] http://openjdk.java.net/
[15] http://www.mono-project.com
[16] http://gcc.gnu.org/

A.2 The Position Report as XML Graph



A.3 The Position Report as OO Model



A.4 The Position Report as Java Class

```
1  package test2;
2
3  import java.util.Date;
4
5  class PosRpt {
6      private int ReqTyp;
7      private String Ccy;
8      private int Rslt;
9      private int AcctTyp;
10     private int SetPxTyp;
11     private double PriSetPx;
12     private int RptID;
13     private double SetPx;
14     private Date BizDt;
15     private int Acct;
16     private Amt Amt_obj;
17     private Instrmt Instrmt_obj;
18     private Qty[] Qty_objs;
19     private Pty[] Pty_objs;
20     private Hdr Hdr_obj;
21
22     public PosRpt() {
23         this.ReqTyp = 0;
24         this.Ccy = "USD";
25         this.Rslt = 0;
26         this.AcctTyp = 1;
27         this.SetPxTyp = 1;
28         this.PriSetPx = 0.00;
29         this.RptID = 541386431;
30         this.SetPx = 0.00;
31         this.BizDt = Util.parseDate("2003-09-10T00:00:00");
32         this.Acct = 1;
33         this.Amt_obj = new Amt();
34         this.Instrmt_obj = new Instrmt();
35         this.Qty_objs = new Qty[] {new Qty(), new Qty(), new Qty()};
36         this.Pty_objs = new Pty[] {new Pty(), new Pty(), new Pty()};
37         this.Hdr_obj = new Hdr();
38     }
39
40     public PosRpt(int ReqTyp, String Ccy, int Rslt, int AcctTyp, int SetPxTyp, double PriSetPx, int RptID, double SetPx,
41         this.ReqTyp = ReqTyp;
42         this.Ccy = Ccy;
43         this.Rslt = Rslt;
44         this.AcctTyp = AcctTyp;
45         this.SetPxTyp = SetPxTyp;
46         this.PriSetPx = PriSetPx;
47         this.RptID = RptID;
48         this.SetPx = SetPx;
49         this.BizDt = BizDt;
50         this.Acct = Acct;
51         this.Amt_obj = Amt_obj;
52         this.Instrmt_obj = Instrmt_obj;
53         this.Qty_objs = Qty_objs;
54         this.Pty_objs = Pty_objs;
55         this.Hdr_obj = Hdr_obj;
56     }
57
58     public int getReqTyp() {
59         return ReqTyp;
60     }
61
62     public void setReqTyp(int ReqTyp) {
63         this.ReqTyp = ReqTyp;
64     }
65
66     public String getCcy() {
67         return Ccy;
68     }
69
70     public void setCcy(String Ccy) {
71         this.Ccy = Ccy;
```

```
72      }
73
74      public int getRslt() {
75          return Rslt;
76      }
77
78      public void setRslt(int Rslt) {
79          this.Rslt = Rslt;
80      }
81
82      public int getAcctTyp() {
83          return AcctTyp;
84      }
85
86      public void setAcctTyp(int AcctTyp) {
87          this.AcctTyp = AcctTyp;
88      }
89
90      public int getSetPxTyp() {
91          return SetPxTyp;
92      }
93
94      public void setSetPxTyp(int SetPxTyp) {
95          this.SetPxTyp = SetPxTyp;
96      }
97
98      public double getPriSetPx() {
99          return PriSetPx;
100     }
101
102     public void setPriSetPx(double PriSetPx) {
103         this.PriSetPx = PriSetPx;
104     }
105
106     public int getRptID() {
107         return RptID;
108     }
109
110     public void setRptID(int RptID) {
111         this.RptID = RptID;
112     }
113
114     public double getSetPx() {
115         return SetPx;
116     }
117
118     public void setSetPx(double SetPx) {
119         this.SetPx = SetPx;
120     }
121
122     public Date getBizDt() {
123         return BizDt;
124     }
125
126     public void setBizDt(Date BizDt) {
127         this.BizDt = BizDt;
128     }
129
130     public int getAcct() {
131         return Acct;
132     }
133
134     public void setAcct(int Acct) {
135         this.Acct = Acct;
136     }
137
138     public Amt getAmt_obj() {
139         return Amt_obj;
140     }
141
142     public void setAmt_obj(Amt Amt_obj) {
143         this.Amt_obj = Amt_obj;
144     }
```

```
145
146     public Instrmt getInstrmt_obj() {
147         return Instrmt_obj;
148     }
149
150     public void setInstrmt_obj(Instrmt Instrmt_obj) {
151         this.Instrmt_obj = Instrmt_obj;
152     }
153
154     public Qty[] getQty_objs() {
155         return Qty_objs;
156     }
157
158     public void setQty_objs(Qty[] Qty_objs) {
159         this.Qty_objs = Qty_objs;
160     }
161
162     public Pty[] getPty_objs() {
163         return Pty_objs;
164     }
165
166     public void setPty_objs(Pty[] Pty_objs) {
167         this.Pty_objs = Pty_objs;
168     }
169
170     public Hdr getHdr_obj() {
171         return Hdr_obj;
172     }
173
174     public void setHdr_obj(Hdr Hdr_obj) {
175         this.Hdr_obj = Hdr_obj;
176     }
177 }
```

```
1  package test2;
2
3  import java.text.SimpleDateFormat;
4  import java.text.ParseException;
5  import java.util.Date;
6
7  class Util {
8      private static final SimpleDateFormat dateFormat
9          = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssXXX");
10
11     public static Date parseDate(String date) {
12         try {
13             return dateFormat.parse(date);
14         } catch (ParseException e) {
15             throw new RuntimeException(e);
16         }
17     }
18 }
```

A.5 The Position Report as C# Class

```
using System;

namespace test2{
  class PosRpt {
    public int _ReqTyp { get; set; }
    public string _Ccy { get; set; }
    public int _Rslt { get; set; }
    public int _AcctTyp { get; set; }
    public int _SetPxTyp { get; set; }
    public double _PriSetPx { get; set; }
    public int _RptID { get; set; }
    public double _SetPx { get; set; }
    public DateTime _BizDt { get; set; }
    public int _Acct { get; set; }
    public Amt _Amt_obj { get; set; }
    public Instrmt _Instrmt_obj { get; set; }
    public Qty[] _Qty_objs { get; set; }
    public Pty[] _Pty_objs { get; set; }
    public Hdr _Hdr_obj { get; set; }

    public PosRpt() {
      this._ReqTyp = 0;
      this._Ccy = "USD";
      this._Rslt = 0;
      this._AcctTyp = 1;
      this._SetPxTyp = 1;
      this._PriSetPx = 0.00;
      this._RptID = 541386431;
      this._SetPx = 0.00;
      this._BizDt = Util.parseDate("2003-09-10T00:00:00");
      this._Acct = 1;
      this._Amt_obj = new Amt();
      this._Instrmt_obj = new Instrmt();
      this._Qty_objs = new Qty[] {new Qty(), new Qty(), new Qty()};
      this._Pty_objs = new Pty[] {new Pty(), new Pty(), new Pty()};
      this._Hdr_obj = new Hdr();
    }

    public PosRpt(int ReqTyp, string Ccy, int Rslt, int AcctTyp, int SetPxTyp, double PriSetPx, int RptID, double SetPx,
      this._ReqTyp = ReqTyp;
      this._Ccy = Ccy;
      this._Rslt = Rslt;
      this._AcctTyp = AcctTyp;
      this._SetPxTyp = SetPxTyp;
      this._PriSetPx = PriSetPx;
      this._RptID = RptID;
      this._SetPx = SetPx;
      this._BizDt = BizDt;
      this._Acct = Acct;
      this._Amt_obj = Amt_obj;
      this._Instrmt_obj = Instrmt_obj;
      this._Qty_objs = Qty_objs;
      this._Pty_objs = Pty_objs;
      this._Hdr_obj = Hdr_obj;
    }
  }
}
```

```
using System;
using System.Globalization;

namespace test2 {
  class Util {
    public static DateTime parseDate(string date) {
      return DateTime.Parse(date,  null, DateTimeStyles.RoundtripKind);
    }
  }
}
```

A.6 The Position Report as C++ Class

```
#ifndef _test2_PosRpt_H_
#define _test2_PosRpt_H_

#include "Amt.hpp"
#include "Pty.hpp"
#include "Instrmt.hpp"
#include "Util.hpp"
#include <string>
#include "Qty.hpp"
#include "Hdr.hpp"
#include <ctime>

namespace test2 {
  class PosRpt {
  private:
    long _ReqTyp;
    std::string _Ccy;
    long _Rslt;
    long _AcctTyp;
    long _SetPxTyp;
    double _PriSetPx;
    long _RptID;
    double _SetPx;
    std::tm _BizDt;
    long _Acct;
    Amt* _Amt_obj;
    Instrmt* _Instrmt_obj;
    Qty** _Qty_objs;
    Pty** _Pty_objs;
    Hdr* _Hdr_obj;

  public:
    PosRpt();
    PosRpt(long _ReqTyp, std::string _Ccy, long _Rslt, long _AcctTyp, long _SetPxTyp, double _PriSetPx, long _RptID, dou
    ~PosRpt();
    long getReqTyp();
    void setReqTyp(long ReqTyp);
    std::string getCcy();
    void setCcy(std::string Ccy);
    long getRslt();
    void setRslt(long Rslt);
    long getAcctTyp();
    void setAcctTyp(long AcctTyp);
    long getSetPxTyp();
    void setSetPxTyp(long SetPxTyp);
    double getPriSetPx();
    void setPriSetPx(double PriSetPx);
    long getRptID();
    void setRptID(long RptID);
    double getSetPx();
    void setSetPx(double SetPx);
    std::tm getBizDt();
    void setBizDt(std::tm BizDt);
    long getAcct();
    void setAcct(long Acct);
    Amt* getAmt_obj();
    void setAmt_obj(Amt* Amt_obj);
    Instrmt* getInstrmt_obj();
    void setInstrmt_obj(Instrmt* Instrmt_obj);
    Qty** getQty_objs();
    void setQty_objs(Qty** Qty_objs);
    Pty** getPty_objs();
    void setPty_objs(Pty** Pty_objs);
    Hdr* getHdr_obj();
    void setHdr_obj(Hdr* Hdr_obj);
  };
}

#endif // _test2_PosRpt_H_
```

```
#include "PosRpt.hpp"

namespace test2 {
  PosRpt::PosRpt() {
    this->_ReqTyp = 0;
    this->_Ccy = "USD";
    this->_Rslt = 0;
    this->_AcctTyp = 1;
    this->_SetPxTyp = 1;
    this->_PriSetPx = 0.00;
    this->_RptID = 541386431;
    this->_SetPx = 0.00;
    this->_BizDt = Util::parseDate("2003-09-10T00:00:00");
    this->_Acct = 1;
    this->_Amt_obj = new Amt();
    this->_Instrmt_obj = new Instrmt();
    this->_Qty_objs = new Qty*[3] {new Qty(), new Qty(), new Qty()};
    this->_Pty_objs = new Pty*[3] {new Pty(), new Pty(), new Pty()};
    this->_Hdr_obj = new Hdr();
  }

  PosRpt::PosRpt(long _ReqTyp, std::string _Ccy, long _Rslt, long _AcctTyp, long _SetPxTyp, double _PriSetPx, long _RptI
    this->_ReqTyp = _ReqTyp;
    this->_Ccy = _Ccy;
    this->_Rslt = _Rslt;
    this->_AcctTyp = _AcctTyp;
    this->_SetPxTyp = _SetPxTyp;
    this->_PriSetPx = _PriSetPx;
    this->_RptID = _RptID;
    this->_SetPx = _SetPx;
    this->_BizDt = _BizDt;
    this->_Acct = _Acct;
    this->_Amt_obj = _Amt_obj;
    this->_Instrmt_obj = _Instrmt_obj;
    this->_Qty_objs = _Qty_objs;
    this->_Pty_objs = _Pty_objs;
    this->_Hdr_obj = _Hdr_obj;
  }

  PosRpt::~PosRpt() {
    delete _Amt_obj;
    delete _Instrmt_obj;
    delete[] _Qty_objs;
    delete[] _Pty_objs;
    delete _Hdr_obj;

  }

  long PosRpt::getReqTyp () {
    return _ReqTyp;
  }

  void PosRpt::setReqTyp (long ReqTyp) {
    _ReqTyp = ReqTyp;
  }

  std::string PosRpt::getCcy () {
    return _Ccy;
  }

  void PosRpt::setCcy (std::string Ccy) {
    _Ccy = Ccy;
  }

  long PosRpt::getRslt () {
    return _Rslt;
  }

  void PosRpt::setRslt (long Rslt) {
    _Rslt = Rslt;
  }

  long PosRpt::getAcctTyp () {
```

```
74    return _AcctTyp;
75  }
76
77  void PosRpt::setAcctTyp (long AcctTyp) {
78    _AcctTyp = AcctTyp;
79  }
80
81  long PosRpt::getSetPxTyp () {
82    return _SetPxTyp;
83  }
84
85  void PosRpt::setSetPxTyp (long SetPxTyp) {
86    _SetPxTyp = SetPxTyp;
87  }
88
89  double PosRpt::getPriSetPx () {
90    return _PriSetPx;
91  }
92
93  void PosRpt::setPriSetPx (double PriSetPx) {
94    _PriSetPx = PriSetPx;
95  }
96
97  long PosRpt::getRptID () {
98    return _RptID;
99  }
100
101 void PosRpt::setRptID (long RptID) {
102   _RptID = RptID;
103 }
104
105 double PosRpt::getSetPx () {
106   return _SetPx;
107 }
108
109 void PosRpt::setSetPx (double SetPx) {
110   _SetPx = SetPx;
111 }
112
113 std::tm PosRpt::getBizDt () {
114   return _BizDt;
115 }
116
117 void PosRpt::setBizDt (std::tm BizDt) {
118   _BizDt = BizDt;
119 }
120
121 long PosRpt::getAcct () {
122   return _Acct;
123 }
124
125 void PosRpt::setAcct (long Acct) {
126   _Acct = Acct;
127 }
128
129 Amt* PosRpt::getAmt_obj () {
130   return _Amt_obj;
131 }
132
133 void PosRpt::setAmt_obj (Amt* Amt_obj) {
134   _Amt_obj = Amt_obj;
135 }
136
137 Instrmt* PosRpt::getInstrmt_obj () {
138   return _Instrmt_obj;
139 }
140
141 void PosRpt::setInstrmt_obj (Instrmt* Instrmt_obj) {
142   _Instrmt_obj = Instrmt_obj;
143 }
144
145 Qty** PosRpt::getQty_objs () {
146   return _Qty_objs;
```

```
147   }
148
149   void PosRpt::setQty_objs (Qty** Qty_objs) {
150     _Qty_objs = Qty_objs;
151   }
152
153   Pty** PosRpt::getPty_objs () {
154     return _Pty_objs;
155   }
156
157   void PosRpt::setPty_objs (Pty** Pty_objs) {
158     _Pty_objs = Pty_objs;
159   }
160
161   Hdr* PosRpt::getHdr_obj () {
162     return _Hdr_obj;
163   }
164
165   void PosRpt::setHdr_obj (Hdr* Hdr_obj) {
166     _Hdr_obj = Hdr_obj;
167   }
168 }
```

```
1  #ifndef _test2_Util_H_
2  #define _test2_Util_H_
3
4  #include <string>
5  #include <ctime>
6
7  namespace test2 {
8    class Util {
9    public:
10     static std::tm parseDate(const char* date);
11   };
12 }
13
14 #endif // _test2_Util_H_
```

```
1  #include "Util.hpp"
2
3  namespace test2 {
4    std::tm Util::parseDate(const char* date) {
5      std::tm tmp;
6      strptime(date, "%FT%TZ", &tmp);
7      return tmp;
8    }
9  }
```

A.7 The Position Report as C Struct

```c
#ifndef _PosRpt_H_
#define _PosRpt_H_

#include "Util.h"
#include "Pty.h"
#include <time.h>
#include "Amt.h"
#include "Instrmt.h"
#include "Qty.h"
#include "Hdr.h"

typedef struct {
  long ReqTyp;
  char* Ccy;
  long Rslt;
  long AcctTyp;
  long SetPxTyp;
  double PriSetPx;
  long RptID;
  double SetPx;
  struct tm BizDt;
  long Acct;
  Amt* Amt_obj;
  Instrmt* Instrmt_obj;
  Qty** Qty_objs;
  Pty** Pty_objs;
  Hdr* Hdr_obj;
} PosRpt;

PosRpt* make_default_PosRpt();

PosRpt* make_PosRpt(long _ReqTyp, char* _Ccy, long _Rslt, long _AcctTyp, long _SetPxTyp, double _PriSetPx, long _RptID,

void free_PosRpt(PosRpt* x);

#endif // _PosRpt_H_
```

```c
#include "PosRpt.h"
#include <stdlib.h>

PosRpt* make_default_PosRpt() {
  PosRpt* tmp = malloc(sizeof(PosRpt));
  tmp->ReqTyp = 0;
  tmp->Ccy = "USD";
  tmp->Rslt = 0;
  tmp->AcctTyp = 1;
  tmp->SetPxTyp = 1;
  tmp->PriSetPx = 0.00;
  tmp->RptID = 541386431;
  tmp->SetPx = 0.00;
  tmp->BizDt = parseDate("2003-09-10T00:00:00");
  tmp->Acct = 1;
  tmp->Amt_obj = make_default_Amt();
  tmp->Instrmt_obj = make_default_Instrmt();
  tmp->Qty_objs = (Qty**) make_pointer_array(3, make_default_Qty(), make_default_Qty(), make_default_Qty());
  tmp->Pty_objs = (Pty**) make_pointer_array(3, make_default_Pty(), make_default_Pty(), make_default_Pty());
  tmp->Hdr_obj = make_default_Hdr();
  return tmp;
}

PosRpt* make_PosRpt(long _ReqTyp, char* _Ccy, long _Rslt, long _AcctTyp, long _SetPxTyp, double _PriSetPx, long _RptID,
  PosRpt* tmp = malloc(sizeof(PosRpt));
  tmp->ReqTyp = _ReqTyp;
  tmp->Ccy = _Ccy;
  tmp->Rslt = _Rslt;
  tmp->AcctTyp = _AcctTyp;
  tmp->SetPxTyp = _SetPxTyp;
  tmp->PriSetPx = _PriSetPx;
  tmp->RptID = _RptID;
```

```
  tmp->SetPx = _SetPx;
  tmp->BizDt = _BizDt;
  tmp->Acct = _Acct;
  tmp->Amt_obj = _Amt_obj;
  tmp->Instrmt_obj = _Instrmt_obj;
  tmp->Qty_objs = _Qty_objs;
  tmp->Pty_objs = _Pty_objs;
  tmp->Hdr_obj = _Hdr_obj;
  return tmp;
}

void free_PosRpt(PosRpt* sp) {
  free_Amt(sp->Amt_obj);
  free_Instrmt(sp->Instrmt_obj);

  Qty* tmp_Qty_objs = *sp->Qty_objs;
  while (tmp_Qty_objs != NULL) {
    free_Qty(tmp_Qty_objs);
    tmp_Qty_objs++;
  }
  free(sp->Qty_objs);

  Pty* tmp_Pty_objs = *sp->Pty_objs;
  while (tmp_Pty_objs != NULL) {
    free_Pty(tmp_Pty_objs);
    tmp_Pty_objs++;
  }
  free(sp->Pty_objs);
  free_Hdr(sp->Hdr_obj);

  free(sp);
}
```

```
#ifndef _Util_H_
#define _Util_H_

#include <time.h>

void** make_pointer_array(int size, ...);
struct tm parseDate(const char* date);

#endif // _Util_H_
```

```
#include "Util.h"
#include <stdarg.h>
#include <stdlib.h>

void** make_pointer_array(int size, ...) {
  va_list ap;
  va_start(ap, size);
  void** ary = malloc(sizeof(void*) * size + 1);
  int i;
  for (i = 0; i < size; i++) {
    ary[i] = va_arg(ap, void*);
  }
  ary[i] = NULL;
  va_end(ap);
  return ary;
}

struct tm parseDate(const char* date) {
  struct tm tmp;
  strptime(date, "%FT%TZ", &tmp);
  return tmp;
}
```