Bolero Comments

The guiding principles for bolero$_{XML}$ tools and methodologies are:

- Compliance with standard UML
- Compliance with standard XML
- Compliance with ebXML where relevant
- Technology independence
- Maximum automatic generation of computer artifacts
- Adaptability to change
- Adaptability to multiple and differing industry requirements

We have endeavoured to keep any extensions to the UML language, by use of stereotypes, to a minimum. We firmly believe that, if a large number of extensions is required to the UML modelling language to achieve our ends, then either we are using the wrong modelling language, or we are in some way misusing UML. Currently we have at most 4-5 stereotype extensions and have not needed more. This has been in no way constricting on our modeling, since we have produced 70+ complex electronic documents, using as a base, in excess of 1500 classes.

We have also attempted to keep all reference to any implementation of UML out of the model/s . Any explicit reference to say XML or ODBC , C++, Java etc. in the data model **must** be avoided. This is because the data model may be used in a number of ways. For instance, any information that is required by a program generator should be encapsulated in an area of the model devoted to the particular implementation concerned e.g., by using UML tags devoted to a particular generation tool. We speak out of hard experience, since we made a mistake of not doing this at one point and do not want to suffer from similar problems in the future.

"Tagged values also provide a way to attach implementation-dependent add-inn information to elements. For example, a code generator needs additional information about the kind of code to generate..…Certain tags can be used to tell the code generator which implementation to use" Rumbaugh, Jacobson, Booch

We have used the Book 'The Unified Modeling Language Reference Manual' by Messrs Rumbaugh, Jacobson & Booch Published by Addison Wesley (hereafter called UML Ref Man.) as our Bible for the correct usage of UML. We believe that in establishing any standard we should make the way we model as correct and conventional as possible. This will smooth acceptance of our methodology, by using familiar constructs, and make the way we have modeled readily understandable to those already conversant with UML. This need for acceptance

by the UML users, is also made easier, if the model has not been unnecessarily extended by the use of stereotypes. We do not want the UML users to be arguing about the **way** we have modeled, we really want them to discuss, sometimes to the point of argument, **what** we have modeled.

"The extensibility mechanisms are constraints, tagged values and stereotypes. Keep in mind that an extension, by definition, deviates from the Standard form of UML, and may therefore lead to interoperability problems. The modeler should carefully weigh benefits and costs before using extensions, especially when existing mechanisms will work reasonably well. Typically extensions are intended for particular application environments, but they result in UML dialect , with the advantages and disadvantages of all dialects" UML Ref Man p. 101

1. Normalized Model versus Document Models

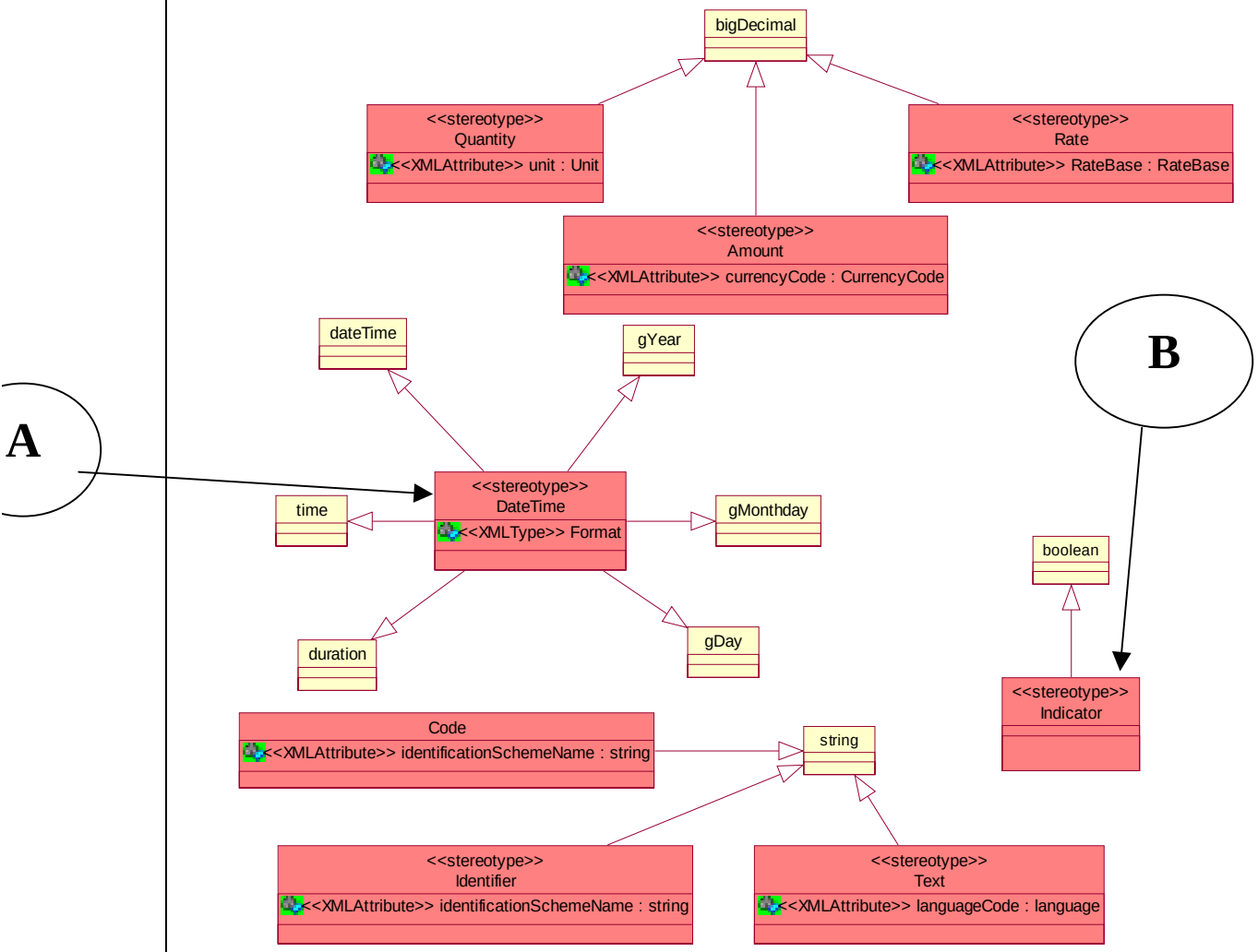
Having been modeling electronic documents for over 2 years we have drawn a number of useful conclusions. We have also gained a great amount of knowledge about data content and structures. We are in the process of changing our model in the light of our insights. A major exercise has been to produce a normalized model, using as a basis all our current stock of electronic documents. We 'tested' the model against our stock of some 1200 instance documents gathered from Clients, Bolero Association Members, Government Bodies, Outside Consultants and the Internet. We have changed the normalized model in the light of any discrepancies this testing produced.

This normalized model is an abstract model of a theoretical database, from whose tables, relationships and fields, subsets may be used to construct individual electronic documents. Thus an individual document may be regarded as a 'view' of the normalized database. The normalized model, is of course, much more than a data dictionary since the relationships between tables/classes are explicitly defined at the level of the normalized model. In an individual document it is necessary to add the multiplicities on the relationships as appropriate to the business use of the document, since the multiplicities between the same two classes may differ from document to document . In a particular document one may not see the totality of a class i.e. it is a view. The normalized model has been converted to a UML representation.

Detailed Analysis of SWIFTStandards XML design rules version 2.3

The following comments are based upon the '**SWIFTStandards XML design rules version 2.3 Technical Specification'**

Deviations from the UML Standard



A. In the diagram above the structure labeled **'A'** suggests that DateTime is composed of all the generalizations associated with DateTime. This is at least the usual UML interpretation (See UML ref. Man. P 53). However the following is stated later

"Stereotype <<XMLType>> (only used in representation class DateTime) indicates that any user defined data type will have to declared which of the primitive datatypes (Time, gDay, gMonth,….) it will use."

There are several points to make here:

1. The <<XMLType>> not only brings the implementation (XML) into the model but also changes the completely the normal meaning of UML.

2. The construct and stereotype are "(only used in representation class DateTime)" i.e this is an exception to normal modeling BUT is only used once. From a generator stance the fewer exceptions one finds to defaults the better, since this reduces the amount of code in the generator and reduces maintenance. Exceptions are sometimes necessary but if one has to produce one exception for one class only, I suggest

that we should try to achieve what is needed in a more standard way.

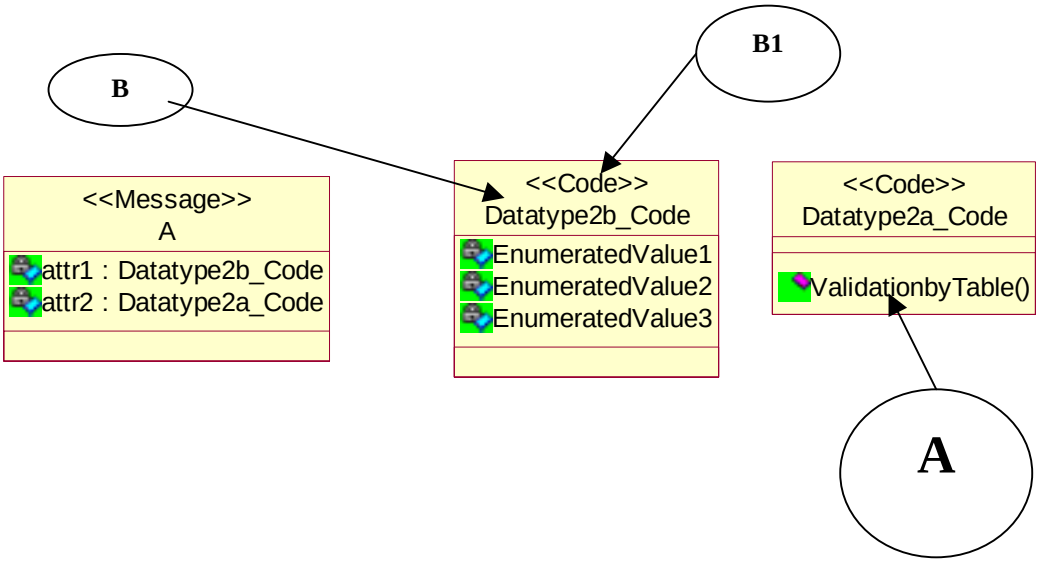
3. It is stated that categorically that SWIFT does not use multiple inheritance in the modeling, but structure labeled '**A**' points to a multiple inheritance – the only thing, according to the design rules, making other than multiple inheritance is the stereotype <<XMLType>>. Thus this stereotype says we should not read this UML usage as multiple inheritance. We venture to say this is more than confusing and makes UML something else.

B. In the diagram the label **'B'** points to a structure whose only purpose seems to be to re-label the Boolean UML data type to indicator. This hardly seems necessary.

C. Generally the diagram is 'littered' with reference to the implementation XML using stereotypes. This makes the whole model XML specific, and seems to suggest that the technical XML requirements may have been thought of first, and then retro fitted to UML. It also make the model very non-standard UML.

"*Models need to be handled by code generators, metrics calculators, report writers.…… Information for other tools needs to be included in the model, but it is not UML information. Tagged values are suitable for holding this information."* UML ref man p 109

D. The diagram above is called a "Representation Class meta-model" . We are not certain of the definition of 'representation class'. It may be a meta-model but we believe we should have a meta-model of the modeling i.e a model of how we model. This would allow an unambiguous definition of our modeling methodology, and would guide the modelers in their modeling.



A. In the diagram above the label '**A**' points to an operation in the class. This is said to be:

" the invariant 'ValdationBytable'. In UML this is not an invariant. In UML an invariant is described as follows:

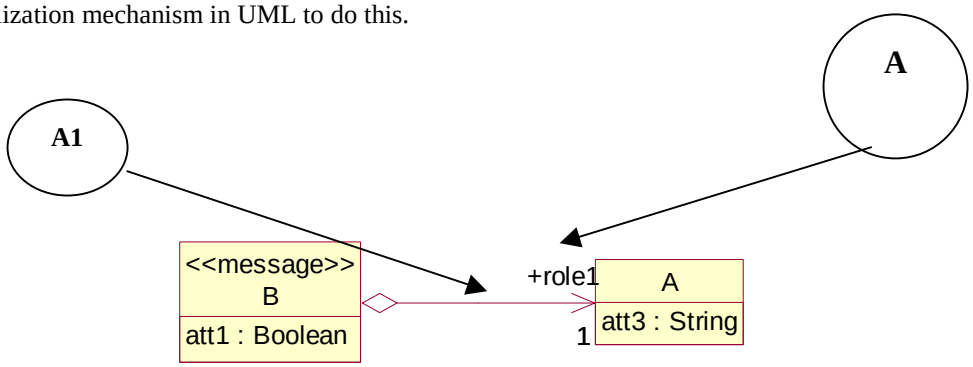
" An invariant is a boolean expression that must true at all times when no operation is active. It is an assertion, not an executable statement. …Structure: An invariant is modeled as a constraint with the stereotype <<***invariant***>> *attached to an element"* UML Ref Man p 317

1.Quite clearly the operation above is not an invariant,. The operation box is being used to store a flag which has a meaning peculiar to the code generator. As said above, this type of requirement would be much better stored as a UML tag.

2.By using the operation box in the class to indicate that this is not an internal enumeration, it may well cause problems for any one wanting to use the model for creating a system which needs to use the operations box for genuine operations.

B. Just a comment in passing**.** In the diagram above labels 'B' and 'B1' point to the stereotype and the name of the class. The stereotype is <<Code>> (we have commented on the many stereotypes before) but the word code is used in the class name 'Datatype2b_Code. This seems to be a 'holding the same information in two places', which does not seem to be good idea. From a practical point of view isone change the stereotype one would also have to change the class name, which if anyone forgot to make both changes could cause confusion. This type of problem is general throughout the technical document.

The constructs above seem to be indicating that Datatype2b_code is a type of Code. There is a perfectly good generalization mechanism in UML to do this.



Instance

```
<B>
    <att1>true</att1>
    <role1>
      <att3>data</att3>
    </role1>
</B>
```

A. In the above diagram the label 'A' points to the rolename (+role1) and the label 'A1' points to the aggregation association between Class B and Class A. This aggregation indicates that A is part of B

"*Aggregation:A form of Association that specifies a whole-part relationship between an aggregate (a whole) and a constituent part"* UML Ref Man p 146

Consider the rolename +role1
"Rolename: A name for a particular association end within an association See also PseudoAttribute" UML Ref Man p 414

"PseudoAttribue: A value related to a class that behaves like an attribute – namely it has a unique value for each instance….An association rolename is a pseudoattribute in the class on the other end of the association " UML Ref Man 397

" Because a rolename can be used like an attribute name to extract values a rolename enters the namespace on the fa rside of the of the association. It goes in the same name space as attribute names" UML Ref Man p 415

Therefore: +role1 is in the namespace of class B and +role1 is a pseudoattribute of B

Now in the XML generated code role1 contains att3 which means that role1 has 'replaced' class A i.e att3 is part of one of the (pseudo)attributes of Class B not of B itself. We believe that the attributes of B, the role, and the class A should be at the same hierarchical level.

It may be very difficult to justify the generated XML code and its relationship to the UML model.

Other Issues:

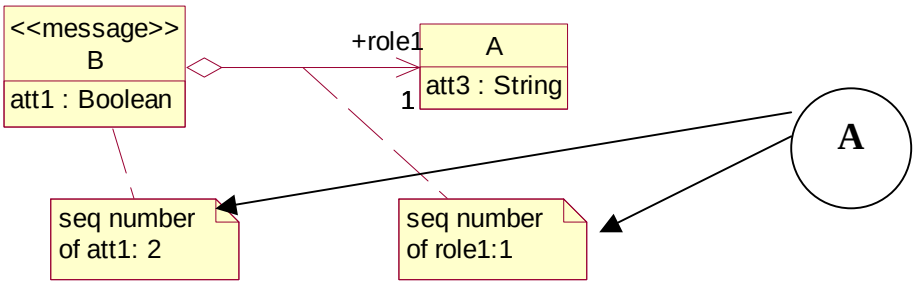
1. It appears, from discussion, that the interface icon is used to specify an 'interface' between the business Model. There is indeed an iterface between the two – in colloquial parlance, however an interface in the UML sense is defined as follows:

   "*The name set of operations that characterize the behaviour of an element … An interface is a descriptor for the visible operations of a class, component or other entity"* UML Ref Man

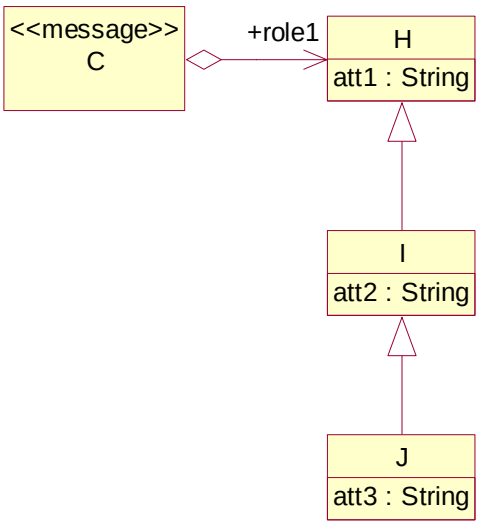
The methodology includes unnecessary modeling extensions.

A. This heading refers to the plethora of stereotypes used in the modeling which will be very much extended if Bolero uses the SWIFT methodology.

The model represents an implementation rather than an abstract design.



A. The label 'A' in the document above points to a UML note which indicates the order in which the generator should generate the artifacts in the diagram. This is generator information and should, we feel, be removed to a tag. E.g the sequence would be quite irrelevant to a generator for an ODBC database. The order might not be needed in an XML Schema !!



A. The above structure is confusing in that **all of the structure is generated in XML**.
The structure above contains a generalization J-> I-> H and thus enables inheritance.

Inheritance is *"The mechanism by which more specific elements incorporate structure and behavior defined by more general elements"* UML Ref Man p299

The usage in XML shows mechanism – which is not needed in an implementation.

"The parent element in a generalization can be defined without knowledge of its children, but children must generally know the structure of their parents to work correctly" UML REF Man p290

In the light of the second quote above it is hard to see how the complete generation of the structure can be justified. We understand if the class C was the aggregate of Class J then J would have all the inherited attributes, but even then it is difficult to see why I and J should be generated. The generalization is a

"Taxonomic relationship between amore general element and a more specifi element. The more specific element is fully consistent with the more general element and contains more information. A more specific element may be used where the more general element is allowed." UML Ref Man p 287

There is no need to display one's Taxonomy in the implementation i.e the XML

XML (Perl type) formats for Data

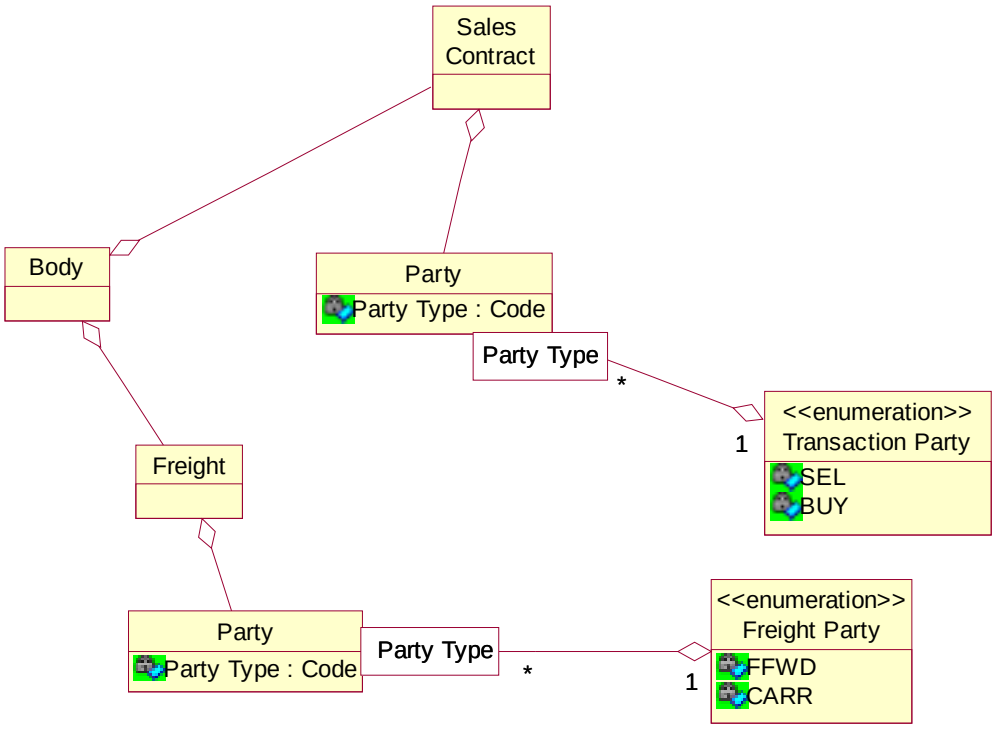
A. We note that the above formatting will be used in the modeling. This is of course XML specific and will tie the model to a particular technology. Given our requirement to remove all reference to a particular technology we would prefer that the model would store the format, in a standard, but language neutral format which could be used to generate a particular format for a technology e.g XML, SQL, Java, C++ etc.

As we are of the opinion that our model should be computer processable we are in fact producing a computer language – albeit a visual one. We believe we should therefore define our 'language' as one would define any other computer language e.g. we need a formal definition of say the characters which can occur in a Class.Name. The standard for this is EBNF (Extended Backus Naur Form) (ISO/IEC 14977:1996(E)) because of this we favour the use of EBNF to define formats as well.

Facilities required by Bolero are not present in the modeling standard

1. **Enumerations (code lists)**
   The use of enumerations by SWIFT is not wholly adequate for the proposed needs of Bolero. We have currently changed our Parties to a transaction from individual classes to one class 'Party' with an attribute of 'PartyType' which point to an enumeration of types of party. The reason for this was that we have very many parties and to have one per class was similar to having one class per Country. We have also discovered as a result of this change that we may need to use a different party code list depending on the context of within a document . As shown below we suggest the use of qualifiers to achieve this. This would have implications for the way enumerations are handled in general by a SWIFT/Bolero work station.

Modeling CodeLists ?



In addition we will need to have several other attributes associated with 'codes' like logical deletion date. In our environment we cannot physically delete a code, since our transactions are not 'atomic' and ships can be at sea for many weeks/months and the documents can be presented to Bolero weeks after they we first created by someone else on the Bolero network.

2. **Recursion**

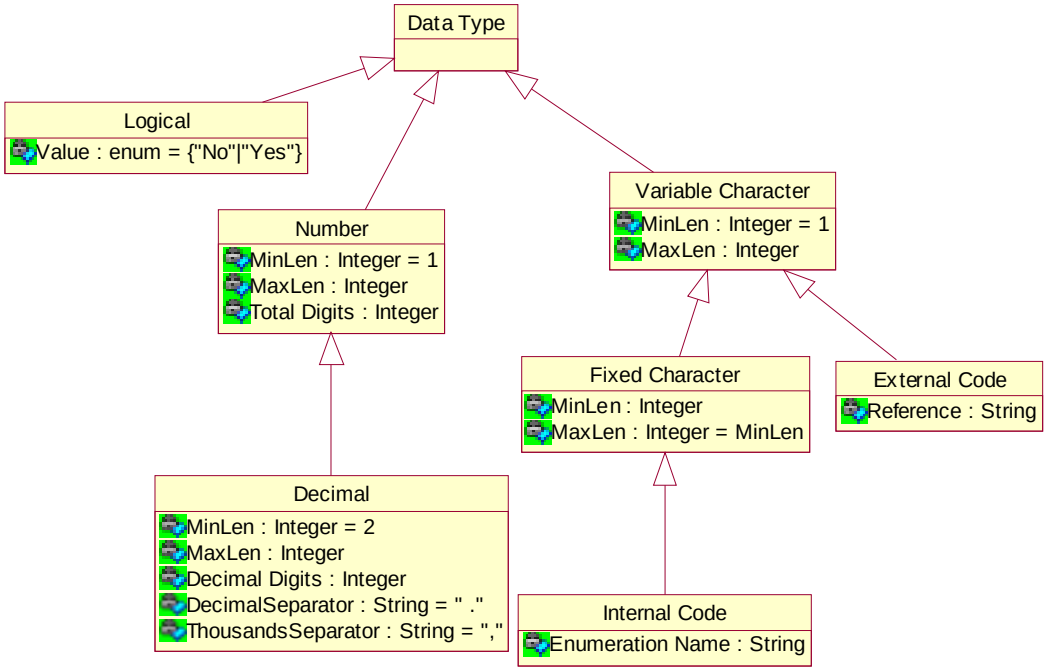
Current modeling by SWIFT assumes there are no loops in the acyclic graphs used to represent Schemas. Bolero has had to invent a stereotype to indicate that a loop in the graph is intentional. This is to represent packing lists which have Boxes within Boxes within Boxes etc. We could review this and use some other way, but the new way would need to be understood by any generator even if we did this.

3. **Rigorously Defined Meta-Model**

The advantage of a rigorously defined meta-(meta) model is that it defines unambiguously, in a computer processable format, how modeling is done and how the elements of the model are defined . This will simplify the production of any tools that are needed to check or process the model. The definition of this meta-model will probably involve UMLmodels and a formal definition language EBNF which is the de facto ISO standard. (ISO/IEC 14977:1996(E))

N.B. The model below gives no indication how the structure will 'realized'

Example of Possible Meta-Model for Data



EBNF Possible definition of Class.Name

Class.Name=[A-Z] ([a-zA-Z0-9_ | '-')+

A Class.Name has the first letter as a capital followed by any Numeric character or Uppercase Lowercase character or an Underline or a '-' sign

4.Schemas

Bolero have been considering the use of Schemas and which facilities within the W3C offering we would want to use. SWIFT appears to be content to use Schemas to define a traditional XML hierarchy which may be quite adequate for their environment. Having worked with Hierarchical databases between 1973-1979 I note that XML hierarchies have all the same problems as Hierarchical databases, which led to the invention of Relational databases. On a practical basis we have already encountered these problems in XML with our modeling. We therefore conclude that we will need to be able to reflect the relational type model in XML. This is possible and there are a number of books which show how this can be done. At this point in time our modeling conventions do not include this type of data structure.

We realize that we will need to use XML namespaces if we use schemas – to allow our customers to access or interface other standards. Currently SWIFT does not have a mechanism for this in their schemas.

We also have detected a need to pass spreadsheet type structures in our XML but again there is nothing in ours or SWIFTs conventions to show how we would do this.

These particular needs for Schemas suggest strongly that we should work on meta-modeling of our data constructs.