http://www.londonmarketsystems.com/

Reaching Consensus on Code Lists

Anthony B. Coates
London Market Systems

OASIS Symposium 2005, New Orleans

Contents

- Introduction
- What a difference a day makes
- Turning the tables on code lists
- The code list is in the eye of the beholder
- The only constant is change
- I'm a model, you know what I mean
- Example code list document
- Conclusion

Introduction

- It is not just XML element & attribute names that need to be semantically unambiguous & aligned for interoperability
- Content models for elements need to be aligned
- The lexical form of element and attribute text content also needs to be aligned, i.e. simple data items need to be represented the same way
- This latter point is more important for applications than is alignment of content models or element & attribute names

Introduction

- The idea that alignment of the lexical form of element and attribute text content is more important than alignment of content models or element & attribute names isn't obvious
- However, when you talk to the people who build and maintain back-end systems (e.g. in finance, where my background is), it becomes clear that changing code that handles low-level datatypes (often closely tied to a database) is typically more expensive than changing code to manage a change in a content model or to element or attribute names
- So simple datatype formats have to be aligned

Introduction

- The wide popularity of the W3C XML Schema datatypes has helped somewhat, in terms of aligning date/time and numeric formats
- This presentation focusses on another key simple datatype – code lists, *aka* enumerations, *aka* controlled vocabularies
- For data-oriented XML particularly, code lists are as important as element & attribute names – they form part of the complete vocabulary of the document
- However, every new XML spec takes its own approach to code lists – we need to change that

What a difference a day makes

- What is a code list, then?
- Most people would agree that the following is a code list:
  `{'SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT'}`
- This is a perfectly reasonable set of alphabetic codes for representing days of the week
- However, so is:
  `{'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'}`

What a difference a day makes

- These two code lists are very similar, but certainly not identical
- That said, they can both be used to represent the days of the week
- Of course, you could also use:

```
{'Dim', 'Lun', 'Mar', 'Mer', 'Jeu',
'Ven', 'Sam'}
```

What a difference a day makes

- Then again, you could use:
  `{0, 1, 2, 3, 4, 5, 6}`
  which is suitable as a computer representation, e.g. for a database column
- On the other hand:
  `{'S', 'M', 'T', 'W', 'T', 'F', 'S'}`
  is not suitable as a code list for the days of the week, because the values are not unique

What a difference a day makes

- There is a **choice** of code lists that can be used
- The answer to the question "which choice is the best?" depends on the needs of each **particular situation**
- No wonder it is so hard to agree on “what is a code list”
- Even agreeing on “**what are the codes**” is non-trivial, because there is no globally unique, unambiguous, one-size-fits-all answer
- There are choices to be made, and any solution has to offer the necessary **freedom of choice**

Turning the tables on code lists

- What the “days of the week” example showed was that for each **conceptual code** in a list, there are many possible **associated values**
- Some of those associated values are suitable for use as unique codes, some are not
- This leads to a tabular model, where each row of the table represents a conceptual code, and each column represents an associated value:

Turning the tables on code lists

numeric (key)	english, uppercase (key)	english, mixed case (key)	french, mixed case (key)	english, single character
0	SUN	Sun	Dim	S
1	MON	Mon	Lun	M
2	TUE	Tue	Mar	T
3	WED	Wed	Mer	W
4	THU	Thu	Jeu	T
5	FRI	Fri	Ven	F
6	SAT	Sat	Sam	S

Turning the tables on code lists

- Notice that the first 4 of the 5 columns are labelled as **key** columns
- This means that the values in those columns can be used to uniquely identify the rows, and hence they can be used as code list values
- The term key is used here is the same fashion as for a **relational database table**
- This is the most common case, where a single column can be used as a key. However, consider the following modification:

Turning the tables on code lists

numeric (key)	english, uppercase (key)	english, single character #1	english, single character #2
0	SUN	S	U
1	MON	M	O
2	TUE	T	U
3	WED	W	E
4	THU	T	H
5	FRI	F	R
6	SAT	S	A

0
1
2
3

Turning the tables on code lists

- Here, the first two columns are each a key column
- The last two columns are not individually key columns, but together they form a compound key
- While the two individual columns do not contain unique values, the pair of values is unique within each row
- This is again similar to what happens in some relational databases, that a key for the rows need not be constructed from a single column, but instead may be constructed by combining two or more columns

The code list is in the eye of the beholder

- Once you see the tabular nature that underlies the information that can be associated with code lists, it becomes clear why they can be a source of so much debate
- Different users need different subsets of the code list information
- People are inclined to assume that the information they need is all the information that anyone needs

The code list is in the eye of the beholder

- That kind of thinking doesn't work with code lists, because code lists are sufficiently **generic** a concept that they are used across messages/documents, applications, and databases
- The code list details that you need for the XML schemata often will **not be exactly the same** as the details that you need for your database or your application

The code list is in the eye of the beholder

- The **XML schema** may only require a set of 3-letter codes to represent the code list
- The **database** may require a set of numeric codes, plus display labels (possibly in different languages)
- The **application** may need to know which 3-letter code corresponds to which numeric code, so that it can process the XML and update the database
- All of this code list information needs to be able to be stored together in a **single representation** of the code list, so that all usages of the code list can be generated from the same source information

The only constant is change

- Code lists change
- For a code list model to be useful, it has to account for the fact that the code lists will change over time
- There is little use in having a code list model that works only for a code list that is frozen in time
- The code list model has to support changes between versions of a code list
- In terms of the tabular model of code lists that has been shown, the following are typical local customisations:

The only constant is change

- Add or remove a **row** (the set of values associated with a conceptual code);
- Add or remove a **column** (a type of value associated with each conceptual code);
- Add or remove a **key**. Adding a key involves specifying the column(s) to be used for the key;

The only constant is change

- Add one or more rows from a set of code lists together. This implies that the columns are the same, or that null values will be used as required;
- Add one or more columns from a set of code lists together. This implies that there is at least one key in common in each row, and that there are no keys with conflicting values;
- Remove rows for which a key value matches the key value in another code list (this can be used to delete a particular pre-defined subset of the codes);

The only constant is change

- Modify a cell **value** (one of the values associated with a code). Note that this may impact whether the affected column can be used as a key or not;
- Create a **derived column** whose values are generated from the values in other columns

The only constant is change

- There should also be a way for users to understand easily **how** their local code list was **derived** from one or more other code lists
- There needs to be an **audit trail**
- This means that the code list model needs to model the ways that code lists are **modified**
- It also needs to be clear how to **regenerate** a derived code list when the underlying code lists change

I'm a model, you know what I mean

- We will shortly publish a UBL code list document with diagrams of the code list model
- However, detailed diagrams don't work so well in presentations, so I will summarise the content of the diagrams in a series of slides
- There is both a model, in UML, and a W3C XML Schema
- I will refer to the Schema to make things concrete, but understand that there is a separate model which underpins it, and could be implemented in other (schema) languages

I'm a model, you know what I mean

- There are 3 kinds of documents that the Schema supports, and 3 global elements that can be the document root
- `<ColumnSet>` — defines a set of columns and keys that can be re-used in code list definitions
- `<CodeList>` — definition of a **simple** or **derived** code list
- `<CodeListSet>` — a set of code list versions, used to define a code list **configuration**

I'm a model, you know what I mean

- Think back to the earlier tabular examples:

numeric (key)	english, uppercase (key)	english, single character #1	english, single character #2
0	SUN	S	U
1	MON	M	O
2	TUE	T	U
3	WED	W	E
4	THU	T	H
5	FRI	F	R
6	SAT	S	A

0
1
2
3

I'm a model, you know what I mean

- A column is a class of value that can be associated with a code in a list, e.g. the code itself, or a human readable name
- A column has an identifier, and can have human readable and machine processable metadata (e.g. names)
- The Schema annotation structure is copied for this
- Each column has a data type. By default, the W3C XML Schema datatypes are assumed, but the RELAX NG datatype library mechanism is copied so that alternative sets of datatypes could be used

I'm a model, you know what I mean

- A **key** is a set of one or more columns whose value(s) can be used to **uniquely identify** an item in a code list
- Any code list must have at least one key (and hence at least one column), but there is **no upper limit** on the number of keys that can be defined
- Also, there is no concept of a **preferred key**. Choice of a key is assumed to be a **late binding** between a code list and a particular contextual usage
- Like columns, each key has a unique identifier and optional metadata

I'm a model, you know what I mean

- Columns and keys are defined **either** in a column set or code list definition. They can be re-used from either in any code list definition
- A simple code list is a tabular definition of the contents of a code list
- Each item in the code list is represented by a **<Row>** element, and each column is represented by a **<Value>** in that row
- A value can be associated with an explicit column; if not, the column is **inferred** from the order of both the values and columns in the table

I'm a model, you know what I mean

- A value must be provided for each **required** column, but need not be for an **optional** column
- Only **required columns** can be used for keys

I'm a model, you know what I mean

- A **derived** code list is actually a repeatable sequence of code list operations that can be audited and repeated
- Operation types: `<ColumnSetExclusion>`, `<ColumnSetInclusion>`, `<ColumnSetMatch>` (required inclusion), `<ColumnSetUnion>`, `<RowExclusion>`, `<RowInclusion>`, `<RowMatch>`, `<RowUnion>`
- A single derived code list definition can contain an **arbitrary depth** of operations, including embedded code list definitions

I'm a model, you know what I mean

- All definitions have the following common features:
  - user-defined human and/or machine readable metadata can be added (validatable)
  - notional lists and versions are identified by URIs
  - URIs can be provided as possible locations from which to download a definition
  - dereferencing of identifier URIs is not encouraged
- Applies to values (not URIs), columns, keys, code lists, and code list sets

Example code list document (if time permits)

```
<?xml version = "1.0"
      encoding = "UTF-8"?>
<gcl:CodeList
 xmlns:gcl="
  http://xml.genericode.org/2004/ns/CodeList/0.2/
 ">
 <Identification>
  <ShortName>
   CountryIdentificationCode
  </ShortName>
  <Version>1.0</Version>
```

Example code list document

```
<CanonicalUri>
 urn:oasis:names:specification:ubl
 :schema:xsd:CountryIdentificationCode
</CanonicalUri>
<CanonicalVersionUri>
 urn:oasis:names:specification
 :ubl:schema:xsd
 :CountryIdentificationCode-1.0
</CanonicalVersionUri>
</Identification>
```

Example code list document

```
<ColumnSet>
 <Column
  Id="CountryIdentificationCodeContent"
  Use="required">
  <ShortName>
   CountryIdentificationCodeContent
  </ShortName>
  <Data Type="token"/>
 </Column>
 <Column Id="CodeName" Use="required">
  <ShortName>CodeName</ShortName>
  <Data Type="string"/>
 </Column>
```

Example code list document

```
<Key Id="
 CountryIdentificationCodeContentKey
">
 <ShortName>
  CountryIdentificationCodeContentKey
 </ShortName>
 <ColumnRef Ref="
  CountryIdentificationCodeContent
 "/>
</Key>
</ColumnSet>
```

Example code list document

```
<SimpleCodeList>
 <Row>
  <Value ColumnRef="
   CountryIdentificationCodeContent
  ">
   <SimpleValue>AD</SimpleValue>
  </Value>
  <Value ColumnRef="CodeName">
   <SimpleValue>ANDORRA</SimpleValue>
  </Value>
 </Row>
```

Example code list document

```
    <Row>
     <Value ColumnRef="
      CountryIdentificationCodeContent
     ">
      <SimpleValue>ZW</SimpleValue>
     </Value>
     <Value ColumnRef="CodeName">
      <SimpleValue>ZIMBABWE</SimpleValue>
     </Value>
    </Row>
   </SimpleCodeList>
  </gcl:CodeList>
```

Conclusion

- In this presentation, we have examined the reasons why code list models are often too narrow
- A tabular model of a code list, designed to cover a broad range and requirements and usages, has been presented
- The model captures the fact that code lists may be represented by different sets of codes in different circumstances, and that the choice of what is the code and what is associated content can change from one usage to another

Conclusion

- The model is also capable of representing the steps required to **derive a code list** from a set of existing code lists
- Relationships between code lists are **auditable**, and derivations are **repeatable**
- Be aware that the work presented here is still an **early draft**
- Some items will certainly change as the model and associated WXS Schema are refined

Conclusion

- Also, the work will not be complete until there is a reference **software implementation** that can process a derived code list definition to produce the equivalent simple code list

Conclusion

- Questions, comments? Ask now, or come chat with me afterwards
- E-mail me: abcoates@londonmarketsystems.com with “OASIS Symposium 2005” in the subject line