Injective, Surjective, and Bijective XML Data Models

- - -

Using the Concepts of Category Theory to Create Better XML Data Models

Roger Costello
August 2012

Introduction

When you create an XML data model, do you ask yourself:

> *What properties do I want this XML data model to possess?*

Until recently, I didn't.

Mathematicians frequently talk about properties such as the commutative property and the associative property. But do data modelers talk about properties?

For a couple years I have wondered how properties apply to XML. What properties would I want an XML data model to possess?

Category Theory has opened my eyes to the idea of applying properties to XML data models. In this paper I describe three properties that you may want your XML data models to possess: injective, surjective, and bijective. I describe what these terms mean and how they are relevant to XML.

ID and IDREF are Really Important

I now recognize the power of the ID and IDREF data types. With them you turn XML into functions. And functions can be composed. And functions have properties.

Terminology

In this paper I use the term *map*. Other words for map are: function, transformation, operator, arrow, and morphism.

Motivation

When I go hiking, I want to be able to retrace my steps and get back to my starting point.

When I drive to a store, I want to be able to return home.

When I swim out from the shore, I want to be able to get back to the shore.

Going somewhere and then coming back to where one started is important in life.

And it is important in designing XML data models.

Domain, Codomain, and Inverse

If an XML element maps a set of elements (the *domain*) to a set of values (the *codomain*) then it is often useful to have another element that can take the elements in the codomain and send them back to their original domain values. The latter is called the inverse element.

In order for a map to have an inverse element, it must possess two important properties, which I explain now.

The Injective Property

Let the domain be the set consisting of these book Titles:

```
<Titles>
   <Title id="XSLT">XSLT 2.0 and XPath 2.0 Programmer's Reference</Title>
   <Title id="SVG">SVG Programming: The Graphical Web</Title>
   <Title id="Office">Office 2003 XML</Title>
   <Title id="XML">XML Bible</Title>
</Titles>
```

Let the codomain be the set consisting of these Authors:

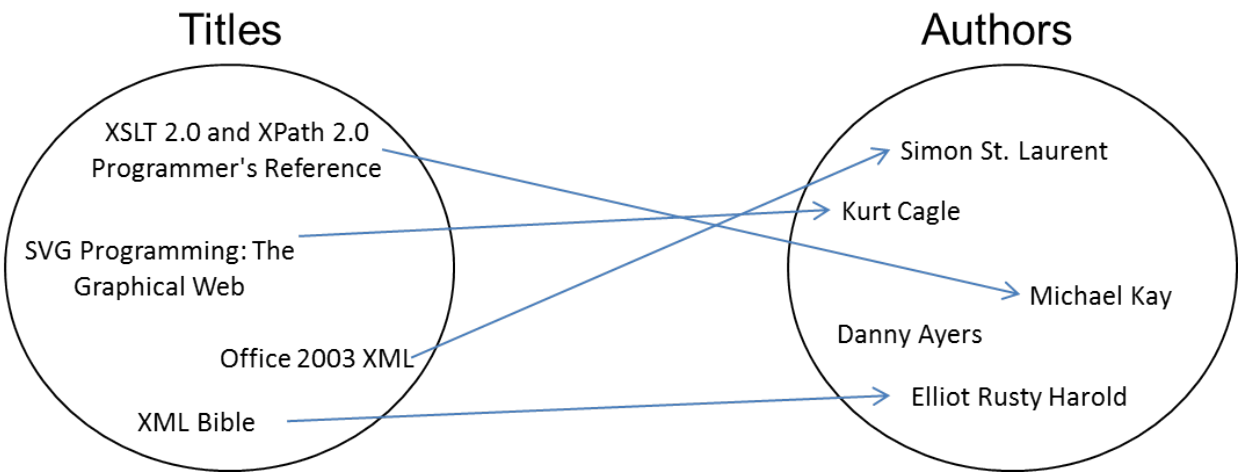
```
<Authors>
   <Author id="MK">Michael Kay</Author>
   <Author id="KC">Kurt Cagle</Author>
   <Author id="SSL">Simon St. Laurent</Author>
   <Author id="ERH">Elliot Rusty Harold</Author>
   <Author id="DA">Danny Ayers</Author>
</Authors>
```

Now I create a map from each element of the domain to a value in the codomain.

Here is one such map:

```
<map id="Titles-to-Authors">
   <singletonMap from="XSLT" to="MK"/>
   <singletonMap from="SVG" to="KC"/>
   <singletonMap from="Office" to="SSL"/>
   <singletonMap from="XML" to="ERH"/>
</map>
```

A graphic makes it clear what the map is doing:



An important thing to observe about the map is that no two elements in the domain map to the same codomain value. This map is called an *injective* map.

[Definition] An injective map is one such that no two elements in the domain map to the same value in the codomain.

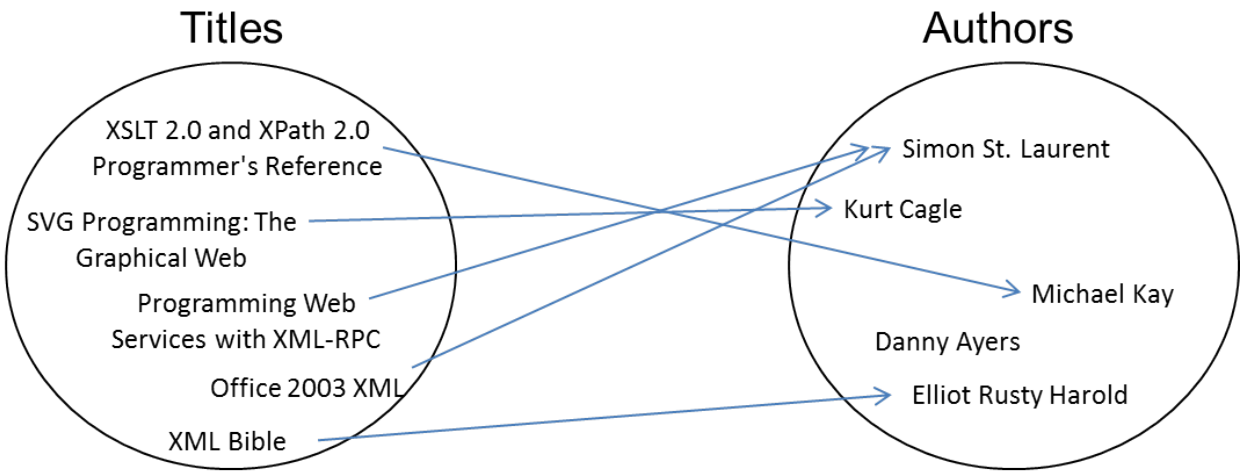
Contrast with the following map, where two elements in the domain – *Office 2003 XML* and *Programming Web Services with XML-RPC* – both map to the same codomain value, Simon St. Laurent.

```
<Titles>
   <Title id="XSLT">XSLT 2.0 and XPath 2.0 Programmer's Reference</Title>
   <Title id="SVG">SVG Programming: The Graphical Web</Title>
   <Title id="Office">Office 2003 XML</Title>
   <Title id="XML">XML Bible</Title>
   <Title id="XML-RPC">Programming Web Services with XML-RPC</Title>
</Titles>

<Authors>
   <Author id="MK">Michael Kay</Author>
   <Author id="KC">Kurt Cagle</Author>
   <Author id="SSL">Simon St. Laurent</Author>
   <Author id="ERH">Elliot Rusty Harold</Author>
   <Author id="DA">Danny Ayers</Author>
</Authors>
```

```
<map id="Titles-to-Authors">
   <singletonMap from="XSLT" to="MK"/>
   <singletonMap from="SVG" to="KC"/>
   <singletonMap from="Office" to="SSL"/>
   <singletonMap from="XML" to="ERH"/>
   <singletonMap from="XML-RPC" to="SSL"/>
</map>
```

A graphic makes it clear that two elements in Titles maps to Simon St. Laurent in Authors:



The map is not injective.

Can you see a problem with creating an inverse map, Authors-to-Titles?

Specifically, what would an inverse map do with Simon St. Laurent? Map it to *Office 2003 XML*? Or map it to *Programming Web Services with XML-RPC*? That is a problem.

[Important] If an XML data model does not have the injective property then it cannot have an inverse map.

In other words, I can't find my way back home.

The Surjective Property

There is a second property that a map must possess in order for it to have an inverse map. I explain that next.

Did you notice in the codomain that there are 5 values:

```
<Authors>
   <Author id="MK">Michael Kay</Author>
   <Author id="KC">Kurt Cagle</Author>
   <Author id="SSL">Simon St. Laurent</Author>
   <Author id="ERH">Elliot Rusty Harold</Author>
   <Author id="DA">Danny Ayers</Author>
</Authors>
```

So there are more values in the codomain than in the domain.

In the Titles-to-Authors map there is no domain element that mapped to the codomain value Danny Ayers.

So what would an inverse map do with Danny Ayers? Map it to *XSLT 2.0 and XPath 2.0 Programmer's Reference*? *SVG Programming: The Graphical Web*? What?

The map is not *surjective*.

[Definition] A surjective map is one such that for each element in the codomain there is at least one element in the domain that maps to it.

[Important] If an XML data model does not have the surjective property, then it does not have an inverse map.

[Important] In order for a map to have an inverse map, it must be both injective and surjective.

Injective + Surjective = Bijective

One final piece of terminology: a map that is both injective and surjective is said to be *bijective*. So, in order for an XML data model to have an inverse map it must be bijective.

This XML data model is bijective:

```
<Titles>
   <Title id="XSLT">XSLT 2.0 and XPath 2.0 Programmer's Reference</Title>
   <Title id="SVG">SVG Programming: The Graphical Web</Title>
   <Title id="Office">Office 2003 XML</Title>
   <Title id="XML">XML Bible</Title>
</Titles>
```

```
<Authors>
   <Author id="MK">Michael Kay</Author>
   <Author id="KC">Kurt Cagle</Author>
   <Author id="SSL">Simon St. Laurent</Author>
   <Author id="ERH">Elliot Rusty Harold</Author>
</Authors>

<map id="Titles-to-Authors">
   <singletonMap from="XSLT" to="MK"/>
   <singletonMap from="SVG" to="KC"/>
   <singletonMap from="Office" to="SSL"/>
   <singletonMap from="XML" to="ERH"/>
</map>
```

Recap

If you want to be able to come back home after your XML map has taken you somewhere, then design your XML to possess the properties of injectivity and surjectivity.

Composing Maps

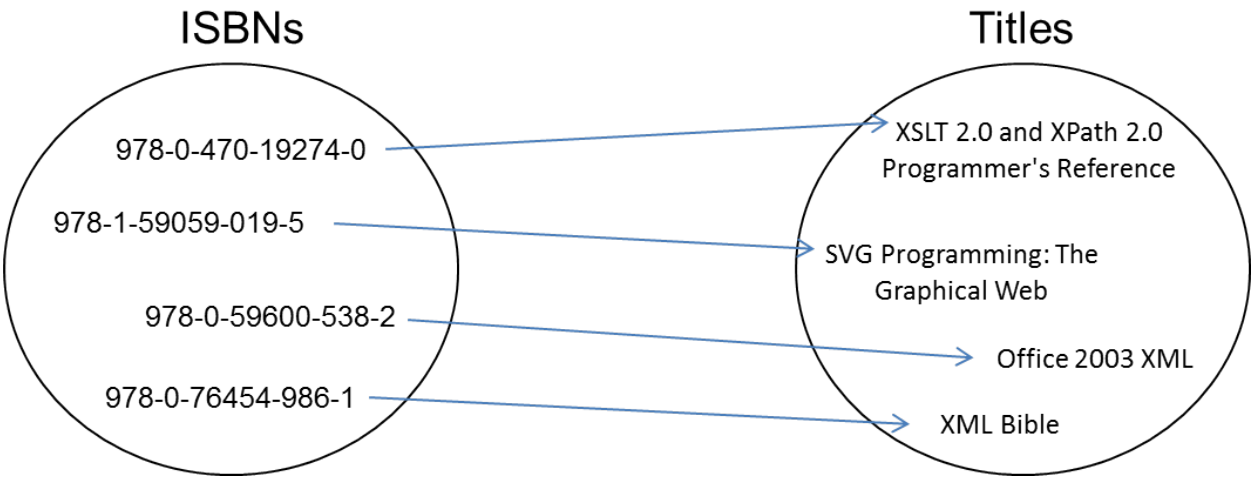
Function composition is central in Category Theory. I think composition is (or should be) central to XML data modeling. Let's see how composition works in XML.

Here is a set consisting of ISBNs and a map from ISBNs to Titles:

```
<ISBNs>
   <ISBN id="isbn-978-0-470-19274-0" />
   <ISBN id="isbn-978-1-59059-019-5" />
   <ISBN id="isbn-978-0-59600-538-2" />
   <ISBN id="isbn-978-0-76454-986-1" />
</ISBNs>

<map id="ISBNs-to-Titles">
   <singletonMap from="isbn-978-0-470-19274-0" to="XSLT" />
   <singletonMap from="isbn-978-1-59059-019-5" to="SVG"/>
   <singletonMap from="isbn-978-0-59600-538-2" to="Office"/>
   <singletonMap from="isbn-978-0-76454-986-1" to="XML"/>
</map>
```

Again, a graphic is useful:



The ISBNs-to-Titles map can be composed with the Titles-to-Authors map to generate a new map: ISBNs-to-Authors.

```
<map id="ISBNs-to-Authors">
   <singletonMap from="isbn-978-0-470-19274-0" to="MK" />
   <singletonMap from="isbn-978-1-59059-019-5" to="KC"/>
   <singletonMap from="isbn-978-0-59600-538-2" to="SSL"/>
   <singletonMap from="isbn-978-0-76454-986-1" to="ERH"/>
</map>
```

This composition is beautifully expressed using notation from Category Theory. Let h represent the ISBNs-to-Titles map, and g represent the Titles-to-Authors map. Then the composition is expressed:

$$g \circ h$$

It is read: g follow h; or, apply g after applying h.

The composed functions can then be applied to each element in the (ISBN) domain:

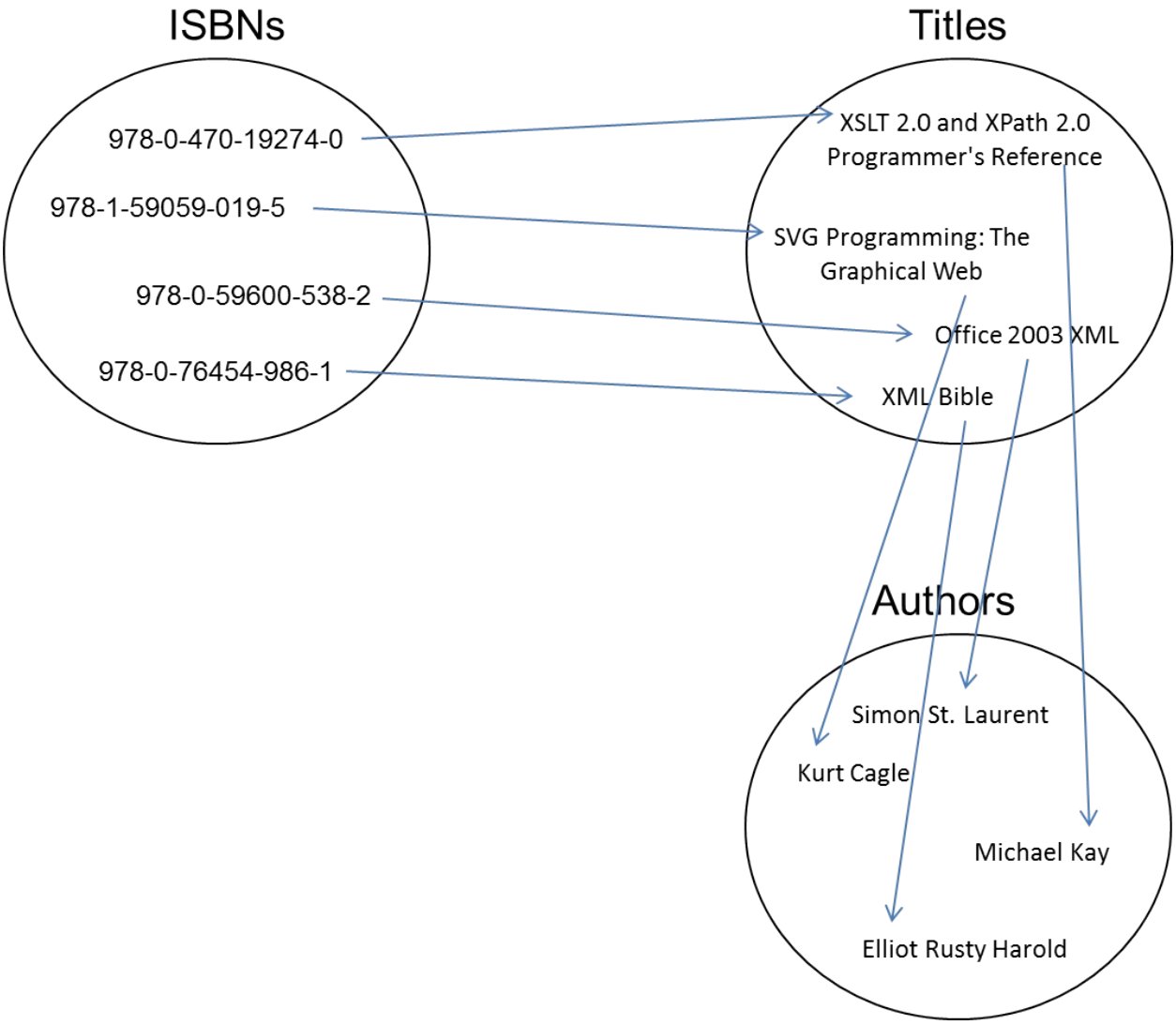
$$(g \circ h)\,(978\text{-}0\text{-}470\text{-}19274\text{-}0) = \text{<Author id="MK">Michael Kay</Author>}$$

$$(g \circ h)\,(978\text{-}1\text{-}59059\text{-}019\text{-}5) = \text{<Author id="KC">Kurt Cagle</Author>}$$

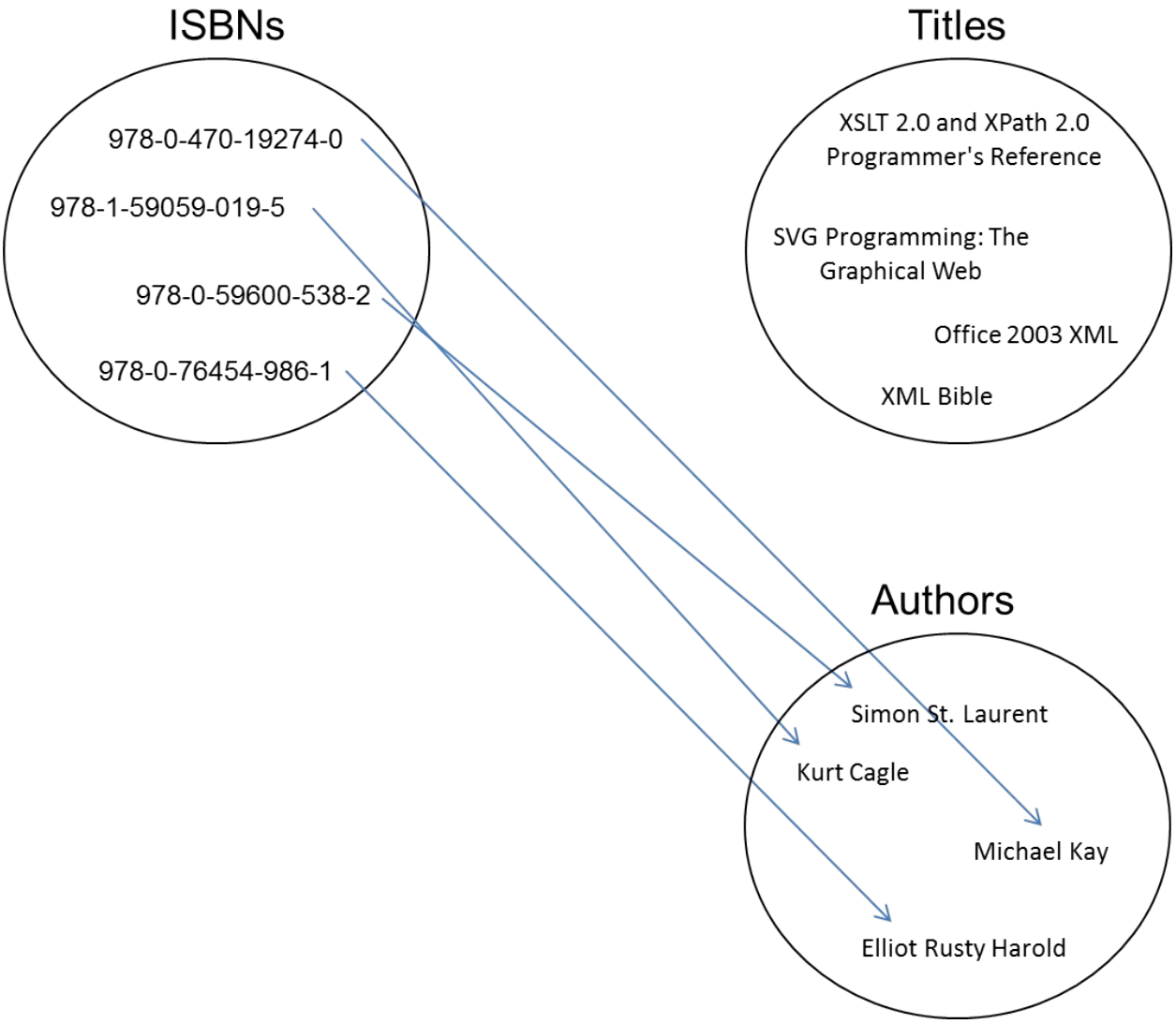
$$(g \circ h)\,(978\text{-}0\text{-}59600\text{-}538\text{-}2) = \text{<Author id="SSL">Simon St. Laurent</Author>}$$

$$(g \circ h)\,(978\text{-}0\text{-}76454\text{-}986\text{-}1) = \text{<Author id="ERH">Elliot Rusty Harold</Author>}$$

Here is a graphic showing how the maps are composed:



This graphic shows the new composed map:



Lessons Learned

1. The ID and IDREF mechanisms in XML are very powerful. Create XML data models that use them.
2. ID and IDREF enable the creation of XML data models that behave as functions. In fact, math textbooks define functions as a set of pairs so ID and IDREF create actual functions.
3. They enable XML data models to possess properties.
4. Given a set of properties, reasoning can be done on XML data models.
5. For an XML data model to be invertible, the data model must possess these two properties: injective and surjective.
6. Function composition is powerful.
7. Function composition can be used in XML data models. Create XML data models in which composition can be used.