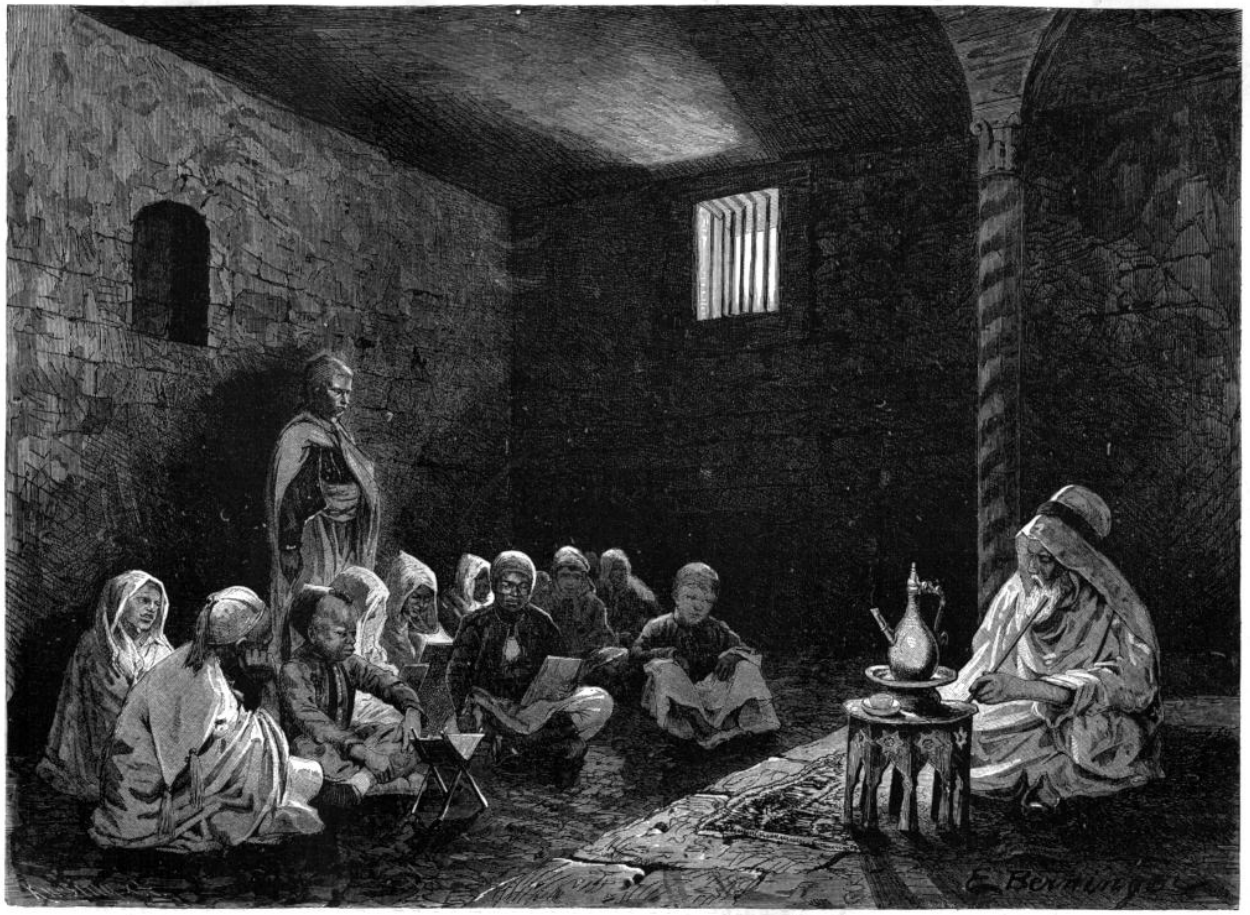
OOXQ - Object-Oriented XQuery *and* Declarative Documents

Liam Quin
Delightful Computing

Declarative Amsterdam 2026

Consulting & Training



Abstract

The author has started to design and work with an object-oriented class system for XQuery.

Limited experience so far suggests a tension between thinking about an application document-first and object-first.

Object-oriented markup can shield the developer from changes in markup, and even from the markup itself, but in that case, why bother?

The answer is a synergy that combines both views, and declarative object-oriented XQuery supports such a view.

The talk will present the object oriented class system both from a developer/query-writer point of view and from an implementer's perspective, and then discuss the implications.

Overview

An object-oriented class system.

Document-first or Object-first?

Shield from changes in markup (but then, why?)

Synergy: OOX supports both views

The technical stuffs & the discussion

Inspiration

The qt4 Community Group added a “method” operator to XPath and i wondered whether it was enough to make a class-based object oriented system.

Design Background

- Don Woods' NeWS Toolkit:

  theWindow newWidth newHeight /resize send

- TypeScript (JavaScript with types and classes)
- Python, Perl, others
- All share a mix of objects and non-object basic values (unlike e.g. SmallTalk).

What is this OO thing

See e.g. Grady Booch (2007):

- Programs are organized as cooperative collections of objects,
- each of which represents an instance of some class,
- and whose classes are all members of a hierarchy of classes
- united via inheritance relationships

What's a class?

- A *class* defines behaviour and properties of a kind of object - e.g chair, or a person, or a memory allocator
- Every class is itself an oject, an instance of a class, up to the top, the 'class' class, so e.g. an Author class might be a *subclass* of Person.

What's behaviour

- In most OO languages, when you invoke a method (or send a message, or whatever it's called) on an object, the *implementation* of that method lives in the class.
- Every object knows what class it's in...

Example

(not using OOX syntax yet)

```
let $author := new $authorClass(
    { name: "Albert", firstName: "Einstein" }
)
return $author.getFormattedName()
```

Why bother?

- Encapsulation: everything together;
- Visibility: limit access to improve maintainability;
- Implicit Dispatch: total headblowing wowness;
- More effective code (shorter/higher level);
- Better match to problem domain

Encapsulation

- Bundling all the data (properties) and the functions that operate on them (methods) into a single related class
- Controlling access through private fields and methods, forcing developers to use a public API.

Implicit Displatch

- This is like apply-templates for data...
- E.g. `$person → format-name()`

  will call the right format-name function on $person without having to check whether $person is from Japan (family-name first).

Implicit Dispatch is Awesome

- No more typeswitch on node type, ala DOM code;

  this means you can't forget a case, and if the implementation start supporting a new object type, your code is already prepared for it.

- The class system means you write a function once, and then put exceptions in sub-classes.

OOX

NOTE: the QT4 specs are in flux in this area, so this is a draft. There was a major change on the day i wrote these slides, too...

OOX – Object-Oriented XQuery/XPath

- Each class is a separate module with its own namespace (as XQuery expects). You could use the same technique with XSLT modules.
- An *object* is an XPath 4 record type: a map with fixed pre-defined fields:

OOX Object

```
Map {
   "isa" : ( $myClass, $otherClass, .. ),
   "public" : map { public fields here },
   "private" : map { for the use of methods only }
   "m" : $methodFinder
}
```

OOX Object

```
Map {
   "isa" : ( $myClass, $otherClass, .. ),

   The is-a field says this object is a member
   of class $myclass, and of the other classes
   listed.
}
```

OOX Object

Map {
   "isa" : ($myClass, $otherClass, ..),
   "m" : $methodFinder
   The *m* function takes a method name and finds the
   right function to call to implement it, looking
   at each class in *isa* in turn, and their parent
   classes, until it finds a match or raises an error.
}

Why the m function?

- A new operator in XPath 4, "method" inspired the work.

  `$obj =?> method(`*`arguments...`*`)`

- It looks in the $obj map for a *method* entry and calls it with $obj as the first argument
- But methods live in classes, not objects.

m

- In OOX methods (right now) are called with

```
$obj =?> m("method-name", { arguments } )
```

- The m function searches for an entry in each class' methods map for *method*-name.
- The map of keywords arguments is supplied to the given method function.

OOX and Declarative Markup

- Objects represent items in the problem domain, or in the solution domain;
- Methods relate to objects, *i.e.* to operations in the problem domain;
- As with XML itself, but *unlike* the middle function layer, our names connect with our information.

Declarative OOX

- A method can't change an object, but it can return a new object, optionally along with a value;
- You end up using *fold-left* a lot more, because each computed value can see the result so far;
- QT4 has new variants with early exit, but they are not yet final (October 2026).

Future Work / Limits

- The syntax is ungainly

```
let $obj := $obj =?> m("set-width", { "value" : 12 })
return $obj =?> m("get-width")
```

- No type checking when `set-width()` is called;
- Modules can't contain initialization code;
- The optimizer has to work hard!

Benefits

- An interesting thought experiment
- If it can be made less ugly and also faster and more robust, this may be really useful, especialy for larger projects.


Liam Quin
Delightful
Computing
liam@delightfulcomputing.com