Origami

Table of Contents

Introduction

- Marc van Grootel (Xokomola)
- Software developer at APS Group in Eindhoven
- HQ in Manchester UK (~700 employees)
- Marketing Operations Systems
- Publishing / Print Management
- Content Management
- Content APIs using XQuery / BaseX

What is Origami?

- Micro-templating library for XQuery
- Started one year ago
- Rewrite after 0.4
- Functional Programming & XQuery 3.1

Github: https://github.com/xokomola/origami

XQuery 3.1: Functions as data

```
declare function list($items, $xf)
{
    $xf(string-join($items, ', '))
};

list(('a','b','c'), upper-case#1)

=: "A, B, C"

declare variable $fn :=
    function($items) {
        sum($items)
    };
```

XQuery 3.1: Maps

```
map {
    'a': 'Apples',
    'b': 'Bananas',
    'c': 'Pears'
}

$map('a')
$map?a
```

XQuery 3.1: Arrays

```
['Apples', 'Bananas', 'Pears']

array { 'Apples', 'Bananas', 'Pears'}

$array(1)
$array?1
```

Origami

- Mu data structure
- Pure code templates
- Document builders
- Node transformers
- Template builders

Mu, meh, micro what?

```
declare variable $xml :=
  <p>Hello,
    <span class="name">Origami</span>!
  </p>;

o:doc($xml)

=:

['p',
    'Hello, ',
    ['span', map { 'class': 'name' },
        'Origami'],
    '!'
]
```

Mu (2)

```
declare variable $mu := o:doc($xml);

o:xml($mu)

=:

<p>Hello, <span class="name">Origami</span>!</p>
```

Mu (3)

- A data structure
- A kind of micro-XML
- Can contain function items and XML fragments
- Compose fragments, navigate with code

> If this were a game of rock-paper-scissors then Mu would be the paper and XML the rock.

Templates as code: result

```
<ul class="groceries">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Pears</li>
</ul>
```

Templates as code: traditional

```
declare function list($items)
{
  <ul class='groceries'>{
    for $item in $items
    return <li>{ $item }</li>
  }</ul>
};

list(('Apples', 'Bananas', 'Pears'))
```

Templates as code: using Mu

```
declare function list($items)
{
  ['ul', map { 'class': 'groceries' },
    for $item in $items
    return ['li', $item]
  ]
};

list(('Apples', 'Bananas', 'Pears'))

=:

['ul', map { 'class': 'groceries' },
  ['li', 'Apples'],
  ['li', 'Bananas'],
  ['li', 'Pears']
]
```

Templates as code: using Mu, to XML

```
declare function list($items)
{
  ['ul', map { 'class': 'groceries' },
    for $item in $items
    return ['li', $item]
  ]
};

o:xml(list(('Apples', 'Bananas', 'Pears')))

=:

<ul class="groceries">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Pears</li>
</ul>
```

Templates as code

> DEMO

Builders

- Convert XML to Mu
- Extract and remove nodes from XML/HTML
- Attach handlers (functions) to nodes
- Convert Mu to XML

Builders: input document

```
declare variable $html :=
  <html>
    <body>
      <p>This is a table</p>
      <table>
        <tr class="odd">
          <th>hello <b>world</b>!</th>
          <th>foobar</th>
        </tr>
        <tr class="even">
          <td>bla <b>bla</b></td>
          <td>foobar</td>
        </tr>
      </table>
    </body>
  </html>
```

Builder extraction rules

```
['table']

['table', ['@*', ()]]

['table',
    ['tr[th]', ()],
    ['tr[td]',
        ['td/node()', ()]
    ]
]
```

Builders: extract nodes

```
o:xml(
  o:doc(
    $html,
    ['table']
  )
)
```

Builders: node extraction

> DEMO

Node transformers

- Origami contains many functions for small scale node transformations.
- They are used inside node handlers (functions attached to elements, attributes or inline)
- They are also useful as tools to manipulate Mu data structures

Node info

```
declare variable $name := 'origami';
declare variable $node :=
  ['p', map { 'class': 'greeting' }
    'hello ', $name];

o:tag($node)
o:attrs($node)
o:children($node)
```

Node transformers

```
declare variable $name := 'origami';
declare variable $node :=
  ['p', 'hello ', $name];

$node => o:insert('foobar')
$node => o:set-attrs(map { 'class': 'greeting' })
$node => o:rename('foo')
$node => o:insert-after('bla')
```

Node transformers: flow

```
$node => o:choose(1,['a','b','c'])
$node => o:choose((1,3),['a','b','c'])
$node => o:repeat(1 to 10, o:copy())
```

Node transformers

> DEMO

Template builder

- Builders are also used for attaching node handlers to elements, attributes or text nodes.
- They use the same rules data structure as shown before but functions can be attached to the returned nodes
- These handlers receive the current node and optional data
- The `o:apply` function is used to evaluate the handler functions

Template builder: the input

```
<html>
  <head>
    <title>Template</title>
    <meta charset="UTF-8" />
    <link rel="stylesheet"
        type="text/css" href="base.css"></link>
  </head>
  <body>
    <ul class="groceries">
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ul>
  </body>
</html>
```

Template builder: tasks

1. Changing the title

Adding a custom CSS link in the header

3. Displaying a list of groceries

The builder uses rules to attach handlers (functions) to nodes selected by an XPath match expression.

Template builder: change title

```
declare variable $rules :=
  ['html',
    ['head',
      ['title',
        function($node, $data) {
          $node => o:insert($data?title)
        }
      ]
    ]
  ];
```

Template builder: add CSS link

```
declare variable $rules :=
  ['html',
    ['head',
      ['link[@rel="stylesheet"][last()]',
        function($node, $data) {
          let $link :=
            $node => o:set-attr(
                map { 'href': $data?css }
            )
          return
            $node => o:after($link)
        }
      ]
    ]
  ];
```

Template builder: build the list

2.

```
declare variable $rules :=
  ['html',
    ['ul[@class="groceries"]',
      ['li[1]',
        function($node, $data) {
          for $item in $data?items
          return
            $node => o:insert($item)
        }
      ],
      ['li', ()]
    ]
  ];
```

Template builder

```
declare variable $rules := ...;
declare variable $template :=
  o:doc(
    o:read-html('groceries.html'),
    o:builder($rules)
  );

let $data :=
  map {
    'title': 'Shopping List',
    'css': 'shopping-list.css',
    'items': ('Apples', 'Bananas', 'Pears')
  }

return
  o:xml(
    o:apply($template, $data)
  )
```

Template builder: the result

```
<html>
  <head>
    <title>Shopping List</title>
    <meta charset="UTF-8"/>
    <link href="base.css"
        rel="stylesheet" type="text/css"/>
    <link href="shopping-list.css"
        rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <ul class="groceries">
      <li>Apples</li>
      <li>Bananas</li>
      <li>Pears</li>
    </ul>
  </body>
</html>
```

Template builder

> DEMO

Plans

- Improve namespace handling
- More tests and documentation
- Hopefully before 1.0 support for other XQuery engines

But also many other ideas

- Fold, HTTP routing library
- Validation of JSON with RelaxNG
- Schema driven transforms
- SVG templating
- JSON-LD

Thanks

- There's much more to tell about Origami, see Github wiki pages

- Origami 0.6 soon on Github
- Want to contribute?

Happy to talk about this and other stuff during a break