Declarative Extensions of XML Languages

Peter King
Department of Computer Science
University of Manitoba, Canada
prking@cs.umanitoba.ca

Simon Thompson
Computing Laboratory
University of Kent, UK
s.j.thompson@kent.ac.uk

This work is a continuation of previous, published joint work [KST] towards a set of XML [XML] language extensions that bring tools from the functional programming world to web authors, extending the power of declarative modeling for the web. Our previous work concentrated on two aspects, *expressions* and *user-defined* events, and deliberately left aside other questions regarding data types, encapsulation and so forth.

Work addressed here includes the addition of various facilities to XML languages including definitions and parameterization; complex data and data types; reactivity, events and continuous 'behaviours'. We have considered these extensions in the light of existing and emerging World Wide Web Consortium [W3C] standards.

Definitions and parameterization □

We have concluded that XML-based languages should contain a general mechanism to support definition and parameterization. We present this by means of a simple example.

Consider the following SVG/SMIL [SMIL, SVG] fragment:

```
<circle cx="20" cy="20" r="100" fill="red">
<animateMotion dur="5s" from="0,0" to="50,50"/>
</circle>
```

This fragment defines a red circle and a motion animation, moving the circle down and right over the course of 5 seconds. Suppose now that one requires an animation comprising 100 such circles of varying colours and animation durations. In XML as it stands, one is restricted to reproducing this code fragment 100 times, making 100 sets of changes to the attributes *fill* and *dur* . This code reproduction would require extensive use of an editor or of, say, XSLT [XSLT] to enact the necessary transformations to make these 100 edited copies.

Our proposed extensions are inspired by the SVG `<symbol>` and `<use>` but the semantics are different enough that we define new elements `<template>` for definitions and `<instance>` for instances of these definitions. These constructs include a simple mechanism for the specification of formal parameters within `<template>` elements, and the provision of actual parameter values within `<instance>` elements. Parameterization allows for more flexible instantiation, increasing the domain of application for templates and making the language more efficient as more cases can be represented as templates, thereby reducing file size.

□ We acknowledge the collaboration of Patrick Schmitz in the work discussed in this sub-section of the report.

Within the template element:

- each (formal) parameter is specified using a `<param>` element and its `name` attribute;
- a default value may optionally be assigned to the parameter using the `value` attribute; omitting this is equivalent to specifying the special value `""` (the empty string), and allows *no value* to be specified. When used for an attribute value within the template, this omission effectively specifies that the language default be used;
- a formal parameter may be referenced anywhere within the template content that defines it; the reference is designated by prefixing the parameter name with the '$' character.

The following example illustrates these notions:

```
<template id="button">
   <param name="color"  value="blue" />
   <param name="label" />
   <param name="num"  value ="0" />
   <rect id="bg" width="100" height="40"
         style="fill:$color"
         x="10" y="calc(25+$num*(40+5)">
      <text>$label</text>
      <animateColor id="rollover"
          begin="bg.mouseover" end="bg.mouseover"
          attributeName="fill" to="yellow" />
   </rect>
</template>

<instance id="homeBtn" xlink:href="#button">
   <param name="label" value="Home"/>
</instance>
<instance id="goBackBtn" xlink:href="#button">
   <param name="label" value="Go Back"/>
   <param name="num"  value="1" />
</instance>
<instance id="searchBtn" xlink:href="#button">
   <param name="color"  value="green" />
   <param name="label" value="Search"/>
   <param name="num"  value="2" />
</instance>
```

This template describes a generic button with default color, rollover behavior etc. Each instance defines a button in the menu, specifying the label, etc. The position is calculated as a dynamic expression based upon the button number, and shows how we can combine our extension features; an earlier proposal on the addition of calculation (`calc`) to XML languages is reported in [KST].

In order to make it possible to refer to each instantiated copy independently, a mechanism is required to associate a local identifier space with each such instance. Support for local id-spaces has two further advantages.

- In the first place, it will enable the use of each instance to be exposed as a true DOM copy, rather than as a shadow copy as used by SVG [SVG, §5.6]; any tools that query the DOM need not be modified to work with instances.
- Further, the presence of local ID name-spaces enables the children to be selected by style sheets, to be targeted by external animations or XMLE event bindings [9], and to be referenced by scripts.

We have investigated two possible approaches to the provision of such separate identifier spaces.

The first is a general solution using structured ID references (e.g. `homeBtn/bg` where the instance introduces a new ID scope, and so `bg` is found as a descendent of `homeBtn`), analogous to that used by a compiler when instantiating objects in a scope-based object-oriented language. In the longer term, the DOM and XML Info Set models will need to address the issues associated with compound documents and fragment transclusion, and may well incorporate such a model for local ID-spaces.

A second solution requires no XML parser changes. Our proposal for our template extension translates local IDs and references. The language interpreter will change all local (within the template) ID definitions and local ID references (i.e. ID-REFs to local IDs), inserting the value of the `<instance>` ID as a prefix. Thus we can express the two distinct references to the background `rect` element for the *home* and *search* button instances in the menu example, as in the following animation declarations:

```
<animate targetElement="homeBtn.bg" …/>
<animate targetElement="searchBtn.bg" …/>
```

Data types and evaluation

In our earlier work towards an augmented XML language [KST], we proposed a limited expression language, which supported calculations of values of a limited repertoire of types, namely the simple, atomic, types of integers, floats, Booleans and strings. Moreover, calculations were restricted to expressions formed from a set of built-in operations, and expressions for calculation could only occur in a limited number of contexts.

We have examined extensions of this earlier proposal in a number of directions, and we have concluded that:

- complex, composite, types should be supported;
- we should be able to support evaluation of expressions over these types;
- expressions for evaluation should be able to include user-defined functions;
- expression evaluation should be possible in a wider set of evaluation contexts.

We now examine these conclusions in turn.

Data types. Our earlier proposal was restricted to the addition of three atomic types: floats, integers and Booleans to the existing XML type of strings. We now propose the introduction of structured "concrete" data types along the lines of those supported in a modern typed functional programming language such as Haskell [Haskell98]. This single mechanism includes finite (or "enumerated") types, records; disjoint unions and their combination in variant records. Types may also be recursive.

There is a question of what *concrete syntax* should be used to describe these structured values. Clearly they can be expressed in the syntax of a language like Haskell, or indeed in the s-expressions of lisp. However, they can be represented in a straightforward way in XML, particularly if attributes are allowed to take typed values. This proposal is implicit in [KST], where numeric attributes were introduced.

To give a concrete example, a record containing a colour and a simple geometric shape, where different shapes are specified in different ways, could be represented thus

```
<geometric-object>
  color=<red/>
  shape=<circle radius=4.3>
          <center x=2.1 y=4.3/>
        </circle>
</geometric-object>
```

Note, however, that the use of typed attributes can simply be seen as *syntactic sugar* for standard XML, since attributes themselves are superfluous. The form

```
<elem a1=v1 a2=v2>
  … content …
</elem>
```

Can be replaced by the (admittedly less user-friendly)

```
<elem>
  <a1>v1</a1>
  <a2>v2</a2>
  … content …
</elem>
```

and such a translation is equally valid for `v1` and `v2` as arbitrarily complex structured values as it is for them as strings. Applying this to the geometric shape example above, we obtain

```
<geometric-object>
  <color>
    <red/>
  </color>
  <shape>
    <circle>
          <radius>
            4.3
          </radius>
          <center>
            <x>
              2.1
            </x>
            <y>
              4.3
            </y>
          </center>
    </circle>
  </shape>
</geometric-object>
```

Other examples include collection types, which can be represented by Haskell-style lists or indeed by multiple items within a single element, and arrays, which could be represented by ordered lists of elements, or by collections of index-value pairs (which can also be seen as finite maps). Recursion also permits tree-structured data.
Such types could be used in the example of the animated circle presented earlier, for instance to represent all one hundred circles by means of an array or a collection type.

The introduction of types allows us to work with complex structured data within an XML document. With types comes the prospect of *type errors*, which break the type discipline of the system. Type errors can manifest themselves dynamically, by means of constraints being broken by computation, which is itself triggered by the need to render a document, for example. *Type checking* can also be performed statically, and this can point to type errors at the time that a document is authored rather than when it is rendered. It remains as future work to examine a number of aspects of type checking XML documents; we expect to investigate, among other points, at least the following.

- The role of type definitions: should these be extracted from a document, or should types be defined explicitly?
- The role of type declarations: should types for parameters, functions and so forth be given in an explicit way, or should some form of *type inference* be used (to the extent that this is possible?
- XML schema and types: should types simply be seen as particular fragments of XML schemata, or should they have a separate identity. Whilst economy favours the former, the latter emphasises the fact that the values passed around have a different, computational, status to the general XML markup in a document. Moreover, it is possible that making an explicit distinction supports or is indeed a necessary prerequisite of static type checking.

Expressions and expression evaluation

The introduction of values and types allows data representing complex objects to be constructed and passed around between elements in a document. Our intention is to introduce computations into the values of attributes. Calculation was discussed in earlier work [KST] where there were two essential restrictions:

- computed (or 'calculated') expressions could only occur in a limited, fixed, collection of contexts;
- the operators and functions that could be used in expressions come from a limited repertoire that explicitly excludes *user-defined functions*.

In the extensions proposed here we lift both of these restrictions.

There are a variety of proposals for user-defined functions in existing XML-based languages.

- XSLT 2.0 allows the definition of functions (as opposed to templates); these are called 'stylesheet functions' and are described in detail in [XSLT, §10.3].
- Standard libraries of functions are defined in a number of XML-based languages, including XPath [XPath], as well as in ECMAScript [ECMAScript].

We would wish to be consistent with these models to the extent that is feasible.

The proposal thus far includes computations which are static. We propose in the following section that dynamically changing *behaviours* can also be included in the computation model.

Reactivity: events and behaviours

Our previous work [KST] outlined a proposal for the support of user-defined events, how such events are raised and how they are handled. In the light of recent developments we see that these events can subsumed under the general event description and handling mechanism of XML Events [XMLE].

The XML Events model views events as atomic, and being initiated externally to the browser; it is possible to build a set of combining forms for events, the most obvious one taking a set of events into a single event which fires when and only when one of the set fires.

Later in this section we will argue that atomic events can also be defined by the user, but first we have to introduce the notion of *behaviours,* which were discussed in [CKT] and were in turn inspired by the Fran model of reactivity [Fran].

A behavior is a data value that *evolves in time*. Motivating examples of *external* behaviours would include

- a numerical value arising from a *sensor* in the environment, measuring something like temperature or pressure;
- a tuple of values (R,G,B) representing the *colour* of an artifact.

Behaviours can also be internal, arising as an artifact of computation, such as

- a *sawtooth* function such representing the fractional part of the current time;
- a combination of simpler behaviours, such as the (pointwise) *difference* between two numerical behaviours, or
- a Boolean behaviour which is *true* if and only if the value of one numerical behaviour is greater than or equal to the value of another. A Boolean behaviour we term a *condition.*

Behaviours can depend on events, and vice versa.

- A composite behaviour can switch repeatedly between values of two behaviours, for example the value of two temperature sensors, on the occurrence of a particular event, such as a left mouse button press. The mechanism underlying this uses an event handler in the definition of a behaviour.
- An event can be triggered by a condition: the event is said to happen at each point where the condition moves from *false* to *true.* In other words, this has the effect of firing an event each time a logical property becomes *true.* By this means, new ‘internal’ events are created, as mentioned earlier in this section.

The presence of a behaviour in a computed value will naturally force a different computation model on those parts of the document that are dependant on the behaviour. For instance, an image which depends on a behavior will be an animated image.

The representation of behaviours in functional languages typically treats them as infinite objects, whereas in SMIL applications one needs definite (or indefinite) finite durations for values. Indeed, this question also has implications for combinations of values. We have discussed the relationship between finite and infinite behaviours in earlier work [CKT].

Encapsulation

Our previous work identified two options, one in which events and handlers are integrated and the other in which they kept separate:

1. A value is like a Fran behaviour:

```
<value type="int"
       initial = "345"
       change  = "increment"
       trigger = "foo.click">
```

2. Values and their "handlers" are separated as follows:

```
<value type="int"
       initial = "345"
       handler = "fred">

<handler name = "fred"
         change  = "increment"
         trigger = "foo.click">
```

Given we are adopting the *XML Events* approach, which separates events and handlers, we have determined that the second form will be adopted.

Future work

Some aspects of our proposed future work in this area have been identified in the foregoing descriptions. These include

- further work on the precise nature of type definitions and parameters, and the relationship between types in our sense) and XML DTD's and Schemata;
- the precise form of our function definition feature, to ensure that we are consistent with the analogous feature in other XML languages;

Furthermore, as this report suggests, our research of the matters discussed in this report has been based on two considerations (a) purely linguistic questions (b) in relation to a further set of use cases. A third aspect (c) using and appropriately extending the prototype translator for the authors" current extended version of XML implemented at Manitoba, has been identified as a matter for future study.

We are very grateful to the Royal Society for their support of this work through funding an Incoming Short Visit to the UK for Dr. King.

Bibliography

[CKT] *Modelling Reactive Multimedia: Events and Behaviours.* Helen Cameron, Peter King, and Simon Thompson. Multimedia Tools and Applications, 19(1), January 2003.

[ECMAScript] *ECMAScript Language Specification, 3rd edition* (December 1999), http://www.ecma-international.org/publications/standards/Ecma-262.htm, last accessed 23/3/07.

[Fran] *Fran version 1.16*, http://conal.net/fran/, last accessed 23/3/07.

[Haskell98] *Haskell 98 Language and Libraries, The Revised Report*, December 2002, http://www.haskell.org/onlinereport/, last accessed 22/3/07.

[KST] *Behavioural Reactivity and Real Time Programming in XML: Functional Programming meets SMIL animation.* Peter King, Patrick Schmitz, and Simon Thompson. In Jean-Yves Vion-Dury, editor, ACM Symposium on Document Engineering 2004, pages 57-66. ACM, January 2004.

[SMIL] *Synchronized Multimedia Integration Language (SMIL 2.1)*, http://www.w3.org/TR/SMIL2/, last accessed 22/3/07.

[SVG] *Scalable Vector Graphics (SVG) Full 1.2 Specification*, http://www.w3.org/TR/SVG12/, last accessed 22/3/07.

[W3C] *The World Wide Web Consortium*, http://www.w3.org/, last accessed 22/3/07.

[XML] *Extensible Markup Language (XML)*, http://www.w3.org/XML/, last accessed 22/3/07.

[XMLE] *XML Events 2, An Events Syntax for XML,* W3C Working Draft 16 February 2007, http://www.w3.org/TR/xml-events, last accessed 23/3/07.
[XPath] *XML Path Language (XPath) 2.0,* http://www.w3.org/TR/xpath20/, last accessed 23/3/07.

[XSLT] *XSL Transformations (XSLT) Version 2.0*, http://www.w3.org/TR/xslt20/, last accessed 22/3/07.