Factonomy Ltd
The University of Edinburgh

Aleksejs Goremikins

Henry S. Thompson

CLIENT-SIDE XML SCHEMA VALIDATION

Edinburgh 2011

Motivation

- Key gap in the integration of XML into the global Web infrastructure is validation.
- Support for validation of more recent schema languages is virtually non-existent
- The growth of interest in rich client-based applications with its emphasis on XML
- There are no free/open-source solutions
- Promotion of the principle of ”software-as-a-service”

Aim of the Work

The aim of the work is to present a prototype JavaScript-based client-side W3C XML Schema validator, together with an API supporting online interrogation of validated documents.

Enabling an important improvement in XML-based client-side applications, extending as it does the existing data type-only validation provided by XForms to structure validation and supporting the development of generic schema-constrained client-side editors.

Schema Validation Algorithm

A technique to convert W3C XML Schema content models to Finite State Automata (FSA) with ranges, including handling of numeric exponents and wildcards (*H. S. Thompson and R. Tobin*).

- **Conversion to FSA** – converts regular expressions to FSAs
- **Unique Particle Attribution** constraint checking
- **Subsumption** – checks two FSAs to confirm that one accepts only a subset of what the other accepts

JavaScript Restrictions

A client-side language that was initially designed to provide dynamic websites.

- **Platform Limitations** – JavaScript does not support direct access for manipulation of the user system files (except for cookie files).
- **Functional Limitations** - JavaScript provides only basic functionality.
- **Interoperability and User Control** - DOM realization in browsers differs

Architecture

The validator operates on an XML DOM instance in the WEB browser. The component model reconstructed from an XML serialization of an object model.

- Schema compilation is more complex than validation itself. Handling this server-side allows to re-use the compilation phase of XSV, an existing W3C XML Schema processor.
- Our focus is on supporting client-side instance authoring environments, where schema change is infrequent.

Architecture

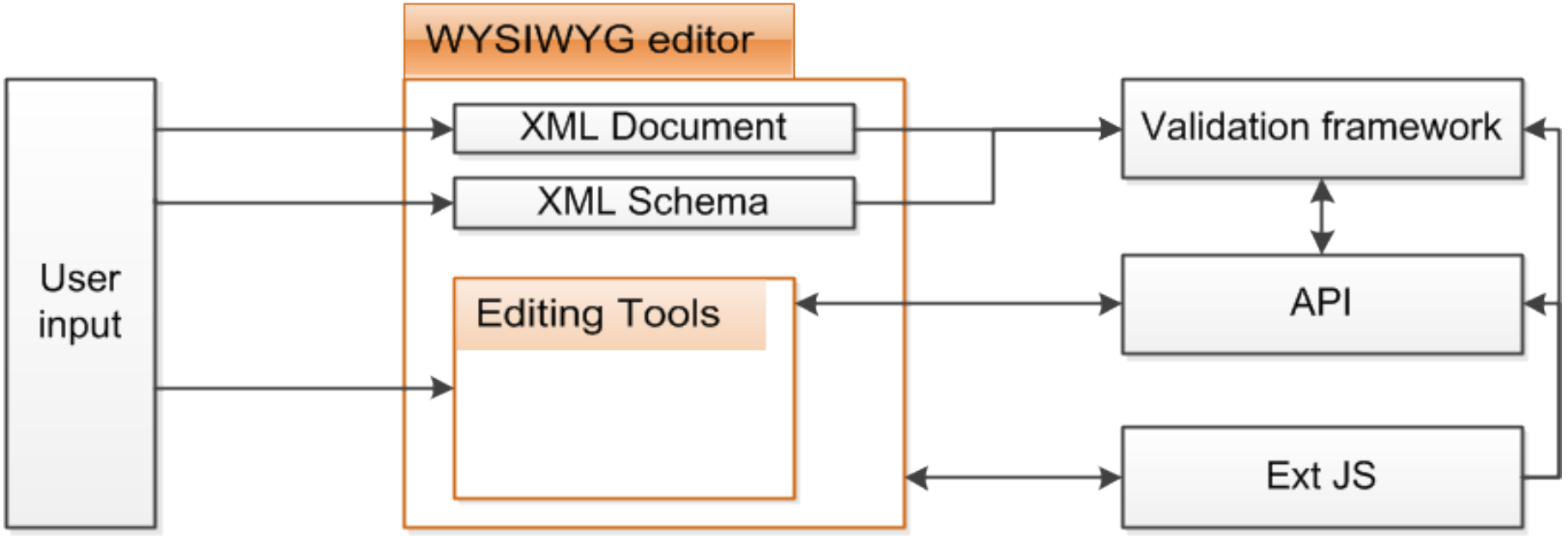

Figure 1. System Architecture

Editing Functionality

Restriction validation, meaning that the user could perform only allowed actions and the document is always valid. Before editing the document, the user is offered a selection of possible validity-preserving actions over the selected component.

Update Operations

- **Insert After** – insert a node immediately after the selected node. Not actual for the root node.
- **Insert Before** – insert a node immediately before the selected node. Not actual for the root node.
- **Insert Into** – [1] insert a node into the complex element. [2] Insert an attribute into the selected element.
- **Delete** – delete a selected node or attribute.
- **Edit** – edit a text node or attribute value.

Algorithms

Element insertions require a valid XML document. **Insert After** and **Insert Before** operation are similar and works with FSM. The **Insert After** algorithm:

- find the parent (P) of the selected element (SE). If P is of mixed type, we can always insert text
- find the selected element (SE) and element after (EA) in the FSM;
- compare *maxOccurs* field of SE and of all SE edges in FSM with real occurrences (RO). If RO < *maxOccurs*, then the SE or edge is a candidate. If the edge is the wild-edge, then the candidate is any defined element . If the edge is exit-edge, then continue;
- if a candidate is equivalent to SE or EA, we can always insert it after. Otherwise, we check candidate edges (CE). If CE has EA, then the candidate is valid for insertion.

Algorithms

Insert Before algorithm has the same actions, but over the element before the selected.

If the selected element is first, we can add any element that comes before the selected element in the FSM; text if the parent is of mixed type; or element itself if its occurrence is less then *maxOccurs*.

Algorithms

Insert Into [1] (for complex elements). Compare *maxOccurs* of possible children with real occurrences (*RO*). If RO < *maxOccurs* a child is a candidate. If selected element is empty, we can insert all candidate.

Otherwise for each child element we apply Insert After algorithm (Insert Before for first child); if Insert After allows candidate, we can insert it after the child (or before). If there are many insertion positions, take the first one.

Insert Into [2] (attributes) we can insert any attribute (from XML Schema) that is not present in the selected element.

Algorithms

The **Delete** action compares real (NoO) and supposed number of occurrences (*minOccurs*). If the *NoO* > *minOccurs* we can delete the element.

For attributes we check, whether it use is optional – can delete; required – cannot delete.

The **Edit** action is trivial: we need to get the text type from the schema and check it during the editing (e.g. using Ext JS framework).

Limitations

Not possible to edit several elements or attributes at once; rename elements or attributes; or to specify a sequence of actions. Not possible to "pass through" an invalid state.

Depends on the stability of the FSM on which the API is based. In our case this is not an issue, because we assume the FSM will not change during an editing session.

Insertion of an element with required content demands the construction of a valid skeleton sub-tree to preserve overall validity.

Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="address">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="town" type="xs:string" />
    <xs:element name="street" type="xs:string" maxOccurs="4" />
    <xs:element name="flat" type="xs:decimal" minOccurs="0" />
    <xs:element name="room" type="xs:string" minOccurs="0" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Example

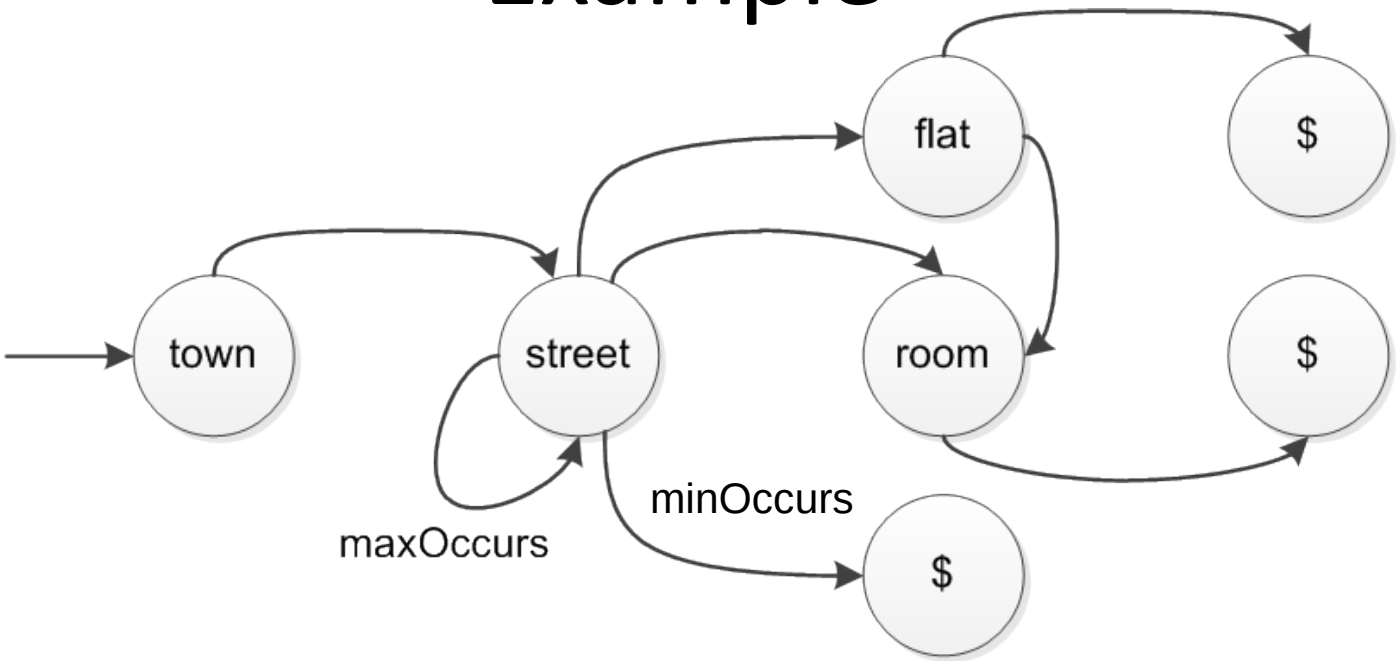

Figure 2. FSM of address element

```
<?xml version="1.0"?>
<address>
  <town>Edinburgh</town>
  <street>Parkside Terrace</street>
  <room>1</room>
</address>
```

Implementation

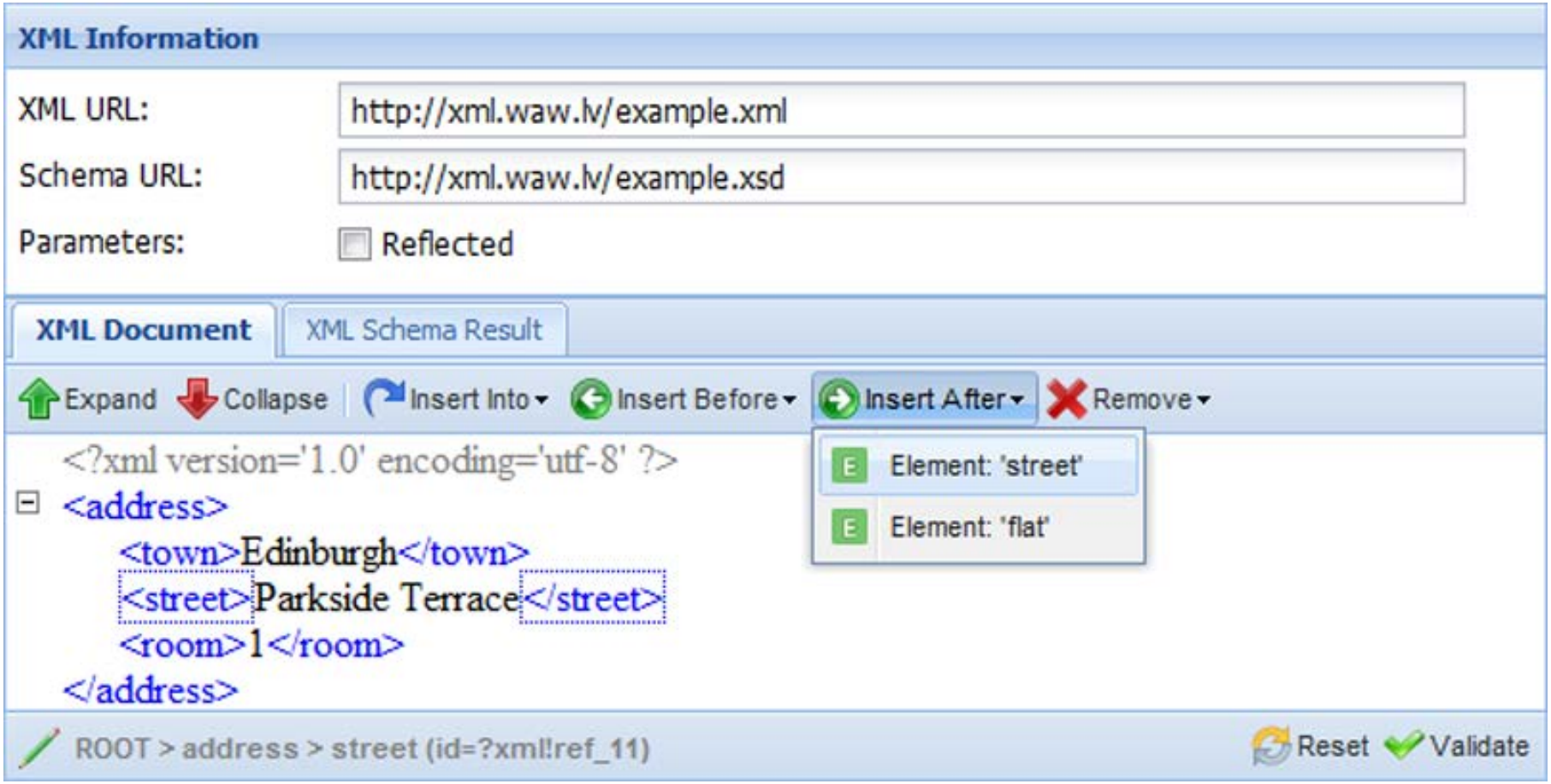

Figure 3. Implementation Example

Analysis

We ran two sets of experiments: in the first we evaluated the validation process; for the second set, we used documents that cover W3C XML Schema constructions to evaluate the editing process.

- For testing the validation process we used a set of 65 tests
- Editing functionality was tested using user-driven methodology
- Both the validation and editing tasks were performed successfully without noticeable delays.

Discussion

The provision of client-side schema validation functionality opens up a range of improved user-friendly XML-based applications.

- Our work enables open-source fully general development in schema-constrained editing tools
- The application provides an opportunity to detect the possible elements/attributes to insert or delete, as well as physically add or delete items from an XML DOM tree and edit text
- We resolved the problem of detection of selected elements within FSM, which implies linking XML DOM, XML Schema, FSM and graphical tree representation.

In future work, we plan to supplement the validation engine to support all W3C XML Schema constructions, explore the valid sub tree insertion problem and add support of a wide range XSL and/ or CSS transformations.

Thanks for your attention!