ECMA/TC39-TG1/02/6

XML for ECMAScript

John Schneider

john.schneider@agiledelta.com

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Proposed Approach
- Conclusions
- Recommendations

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Proposed Approach
- Conclusions
- Recommendations

Scripters are Swamped with XML

The XML Programming Model

- Provides several options to solve a given problem (e.g., DOM, XSLT, XQuery)
- Introduces a steep learning curve
- Requires specialized knowledge and complex concepts (e.g., trees, nodes, recursive decent, functional lang.)
- Minimizes reuse of Scripter's skills and knowledge
- Often requires mixed models (objects, trees, templates, queries, paths)

The XML Programming Model
A Simple Example

Given an XML “order” document with the following shape, compute the total price and add it to the order:

- order
  - customer
    - name
    - address
  - item*
    - description
    - quantity
    - price

The scripter thinks:

```
function addTotal(order) {
total = 0;
for (i in order.item) {
      total += i.price * i.quantity;
}
order.total = total;
}
```

XSLT

XSLT requires:

```
<xsl:stylesheet version="1.0“
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:variable name=“total“ select=“0”/>
      <xsl:template match=“item“ priority=“1”>
            <xsl:set-variable name=“total”
                  select=“$total + ./price * ./quantity”/>
      </xsl:template>
      <xsl:template match=“*|/|comment()|processing-
      instruction()”>
            <xsl:value-of “.”/>
            <xsl:apply-templates/>
      </xsl:template>
      <xsl:template match=“/*[position() = last()]”>
            <xsl:value-of “.”/>
            <xsl:apply-templates/>
            <total><xsl:value-of select=“$total”/></total>
      </xsl:template>
</xsl:stylesheet>
```

The scripter thinks:

```
function addTotal(order) {
total = 0;
for (i in order.item) {
      total += i.price * i.quantity;
}
order.total = total;
}
```

XSLT

XSLT requires:

```
<xsl:stylesheet version="1.0“
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:variable name=“total“ select=“0”/>
      <xsl:template match=“item“ priority=“1”>
            <xsl:set-variable name=“total”
                  select=“$total + ./price * ./quantity”/>
      </xsl:template>
      <xsl:template match=“*|/|comment()|processing-
      instruction()”>
                                                  ()]”>
            <xsl:value-of “.”/>
            <xsl:apply-templates/>
            <total><xsl:value-of select=“$total”/></total>
      </xsl:template>
</xsl:stylesheet>
```

Ok, I cheated. You actually need scripting and the DOM too.

The scripter thinks:

```
function addTotal(order) {
total = 0;
for (i in order.item) {
      total += i.price * i.quantity;
}
order.total = total;
}
```

XSLT
What's Missing?

- A familiar processing model
  - Most scripters immediately subvert recursive flow to achieve procedural patterns -- results in more code
- A single model
  - To accomplish anything mildly complex requires mixing XSLT, XPath, scripting and the DOM
- A flat learning curve
  - Requires a lot of specialized knowledge and skills (templates, recursion, nodes, trees, priority rules, etc.)
- Reuse of familiar concepts
  - What happened to my objects, properties and methods?

The DOM

The DOM requires:

```
function addTotal(document) {
total = 0;
items = document.getElementsByTagName("item");
for (i = 0; i < items.length; i++) {
        item = items.item(i);
        price = item.getElementsByTagName("price").item(0);
        priceValue = price.item(0).getNodeValue();
        quantity = item.getElementsByTagName("quantity").item(0);
        quantityValue = quantity.item(0).getNodeValue();
        total += priceValue * quantityValue;
}
totalText = document.createTextNode(total);
totalElem = document.createElement("total");
totalElem.appendChild(totalText);
document.item(0).appendChild(totalElem);
}
```

The scripter thinks:

```
function addTotal(order) {
total = 0;
for (i in order.item) {
        total += i.price * i.quantity;
}
order.total = total;
}
```

The DOM
What's Missing?

- A single model
  - Mixes tree navigation metaphors and object navigation to achieve largely the same goal
- A flat learning curve
  - Requires specialized knowledge and skills (nodes, trees, a large, complex interface hierarchy, etc.)
- Reuse of familiar concepts
  - My objects, properties and methods feel a little funny

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Proposed Approach
- Conclusions
- Recommendations

The Opportunity

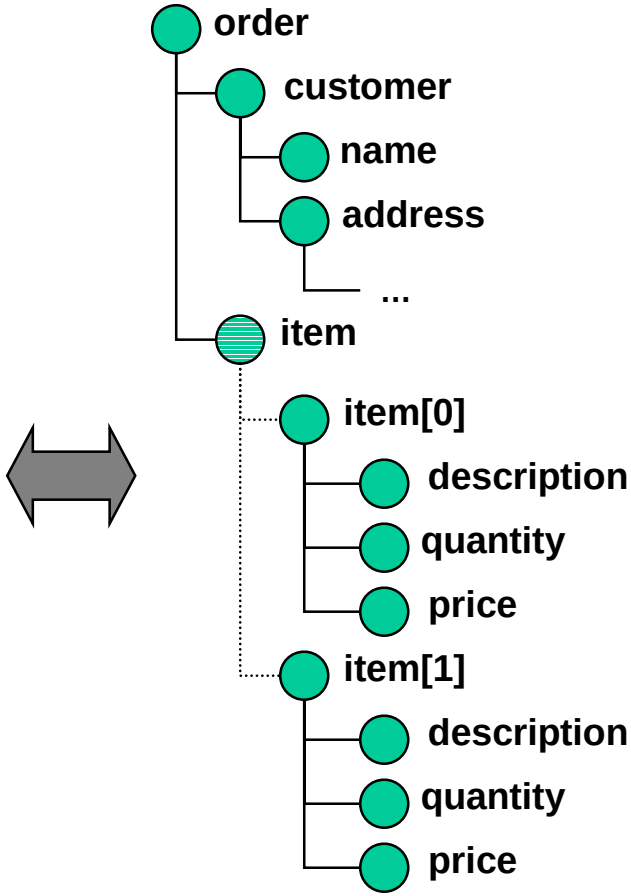
- Provide a simple, familiar, general purpose programming model for XML that:
  - Leverages existing skills and knowledge
  - Reuses familiar concepts, operators and syntax
  - Flattens the learning curve
  - Minimizes need for specialized skills and knowledge
  - Enables scripters immediately with little or no training
- Ultimately, provide a simple object abstraction for creating, navigating and manipulating XML

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Proposed Approach
- Conclusions
- Recommendations

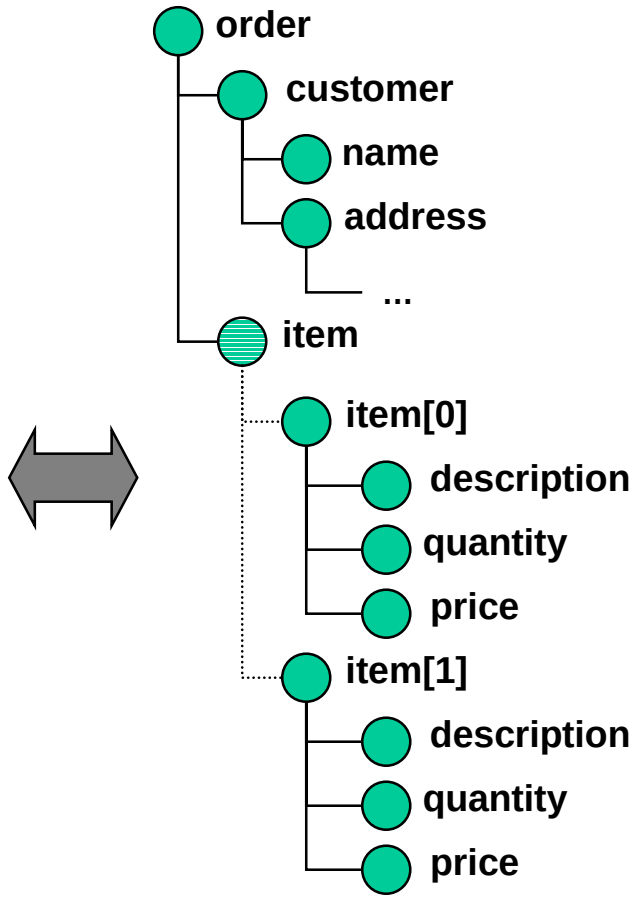
Mapping XML to Objects

```
<order>
    <customer>
        <name>I. Wannabuy</name>
        <address> … </address>
    </customer>
    <item>
        <description>Small Rodent, Generic</description>
        <quantity>35</quantity>
        <price>29.99</price>
    </item>
    <item>
        <description>Catapult</description>
        <quantity>1</quantity>
        <price>149.95</price>
    </item>
</order>
```



Mapping XML to Objects

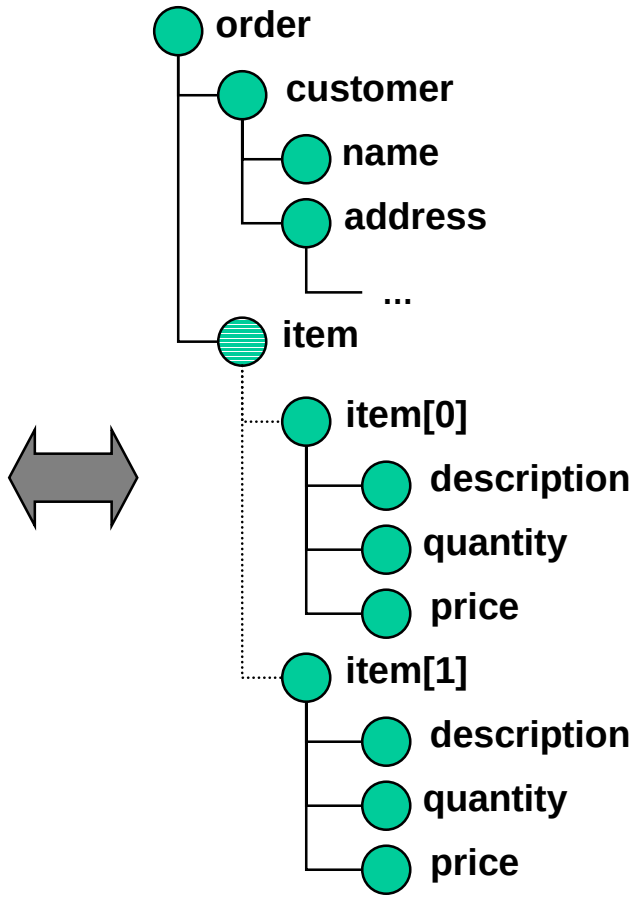
```
order = {
    customer: {
        name: “I. Wannabuy”,
        address: … ,
    }
    item [
        {
            description: “Small Rodent, Generic”,
            quantity: 35,
            price: 29.99
        },
        {
            description: “Catapult”,
            quantity: 1,
            price: 149.95
        }
    ]
}
```



Great! So we can just map XML onto ECMAScript Objects. Right?

Mapping XML to Objects

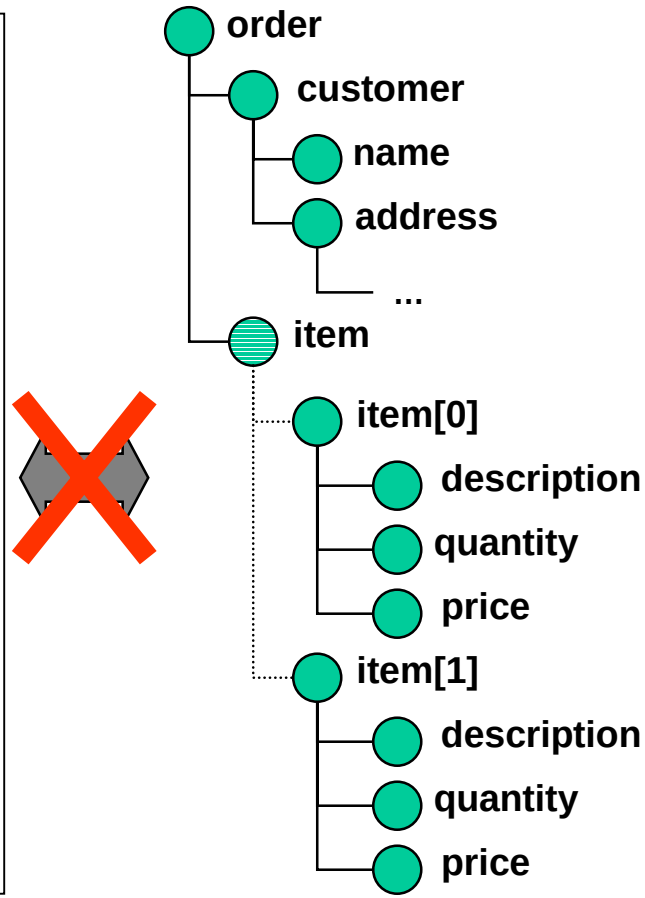
```
order = {
    item [
        {
            quantity: 35,
            price: 29.99
            description: “Small Rodent, Generic”,
        },
        {
            price: 149.95
            description: “Catapult”,
            quantity: 1,
        }
    ]
 customer: {
        name: “I. Wannabuy”,
        address: … ,
    }
}
```



Well, not quite. For starters, order is NOT important in Objects.

Mapping XML to Objects

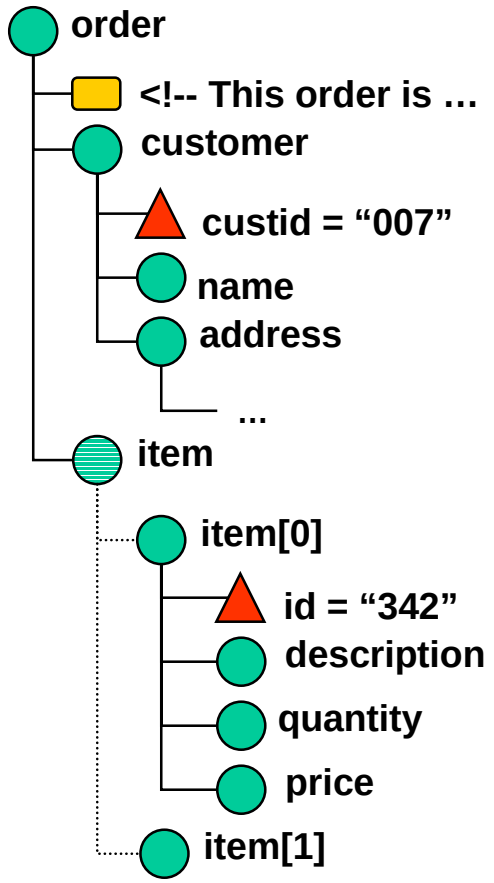
```
<order>
    <item>
        <quantity>35</quantity>
        <price>29.99</price>
        <description>Small Rodent, Generic</description>
    </item>
    <item>
        <price>149.95</price>
        <description>Catapult</description>
        <quantity>1</quantity>
    </item>
    <customer>
        <name>I. Wannabuy</name>
        <address> … </address>
    </customer>
</order>
```



But, order is critical in XML.

Mapping XML to Objects
What's Missing?

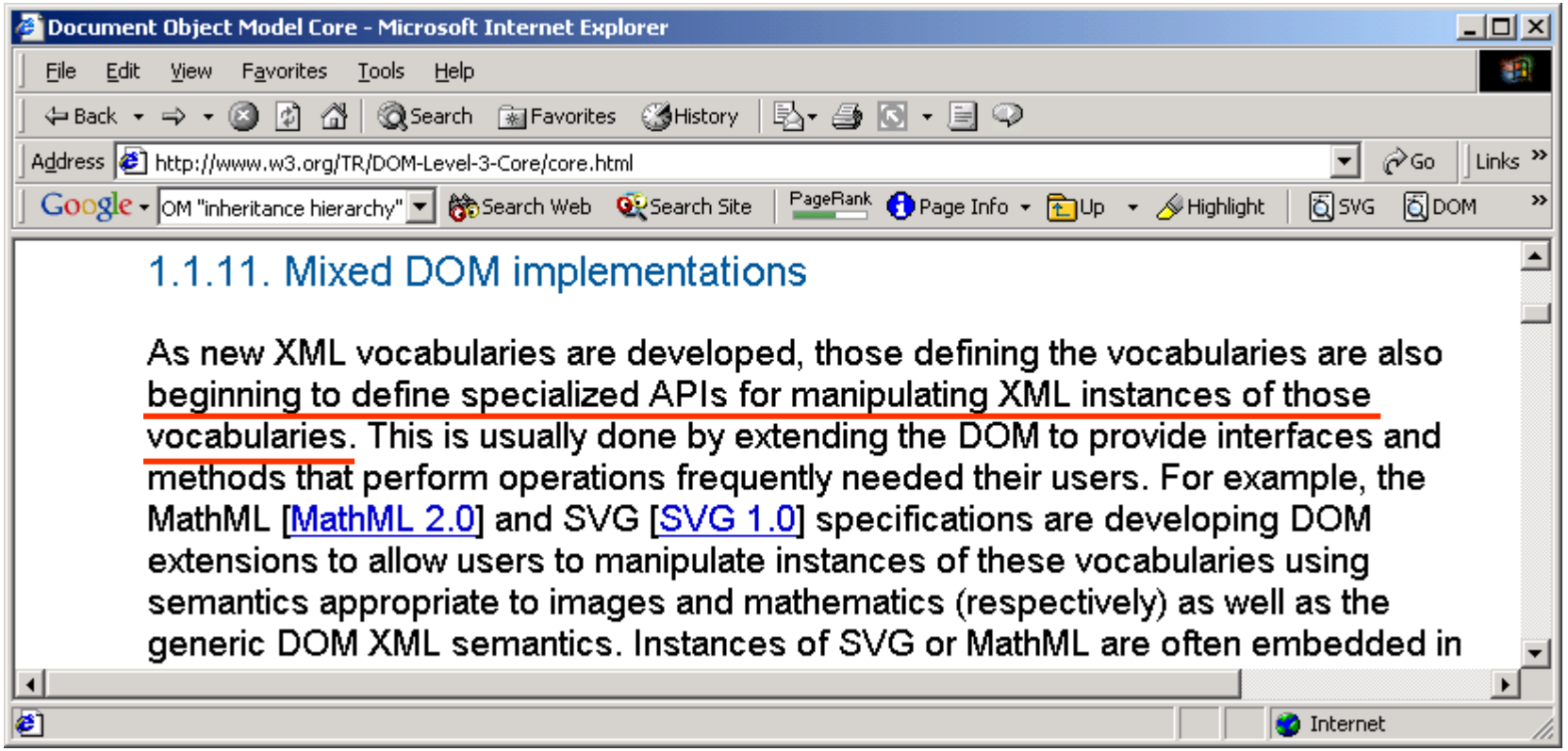
- Well defined order semantics
  - What is property order for new object?
  - Where are new properties added?
  - What is impact of deleting properties?
- Operators for controlling order
  - Specify property order
  - Modify property order
  - Preserve property order
- Operators for creating and manipulating additional XML artifacts
  - Attributes, Comments, PIs
  - Mixed content



Bottom line: ECMAScript object model is insufficient for XML data.

Specialized DOM APIs

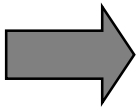
- Vocabulary specific DOM extensions



SVG DOM Example

- A great step forward!

```
function pointInsideRect(point, rect) {
      var px = Number(point.getAttribute("x"));
      var py = Number(point.getAttribute("y"));
      var rx = Number(rect.getAttribute("x"));
      var ry = Number(rect.getAttribute("y"));
      var rw = Number(rect.getAttribute("width"));
      var rh = Number(rect.getAttribute("height"));
      if ((px > rx) && (px < rx + rw)
        && (py > ry) && (py < ry + rh))
            return true;
        else
            return false;
}
```

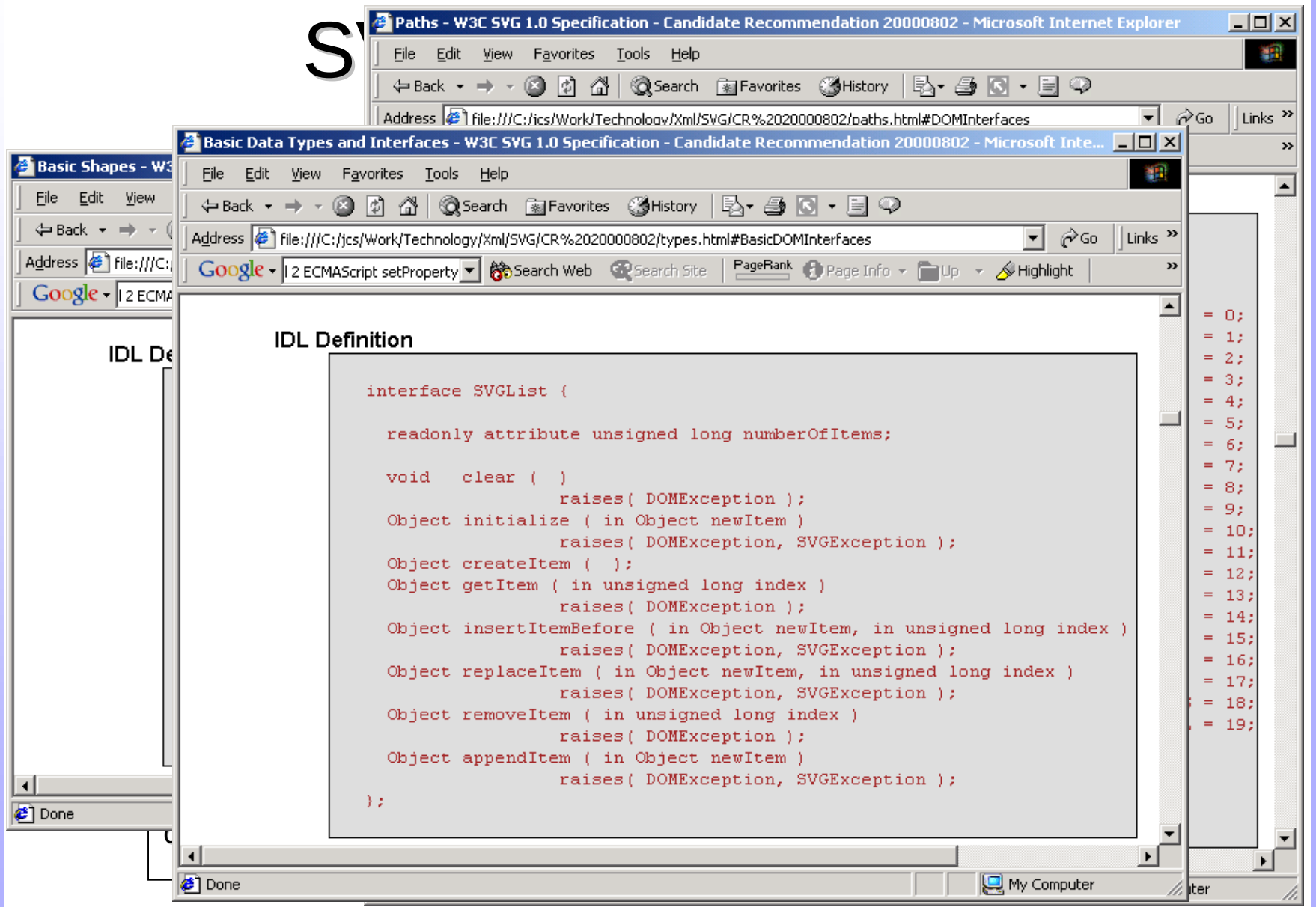


```
function pointInsideRect(point, rect) {
        if ((point.x > rect.x)
        && (point.x < rect.x + rect.width)
        && (point.y > rect.y)
        && (point.y < rect.y + rect.height))
            return true;
        else
            return false;
}
```

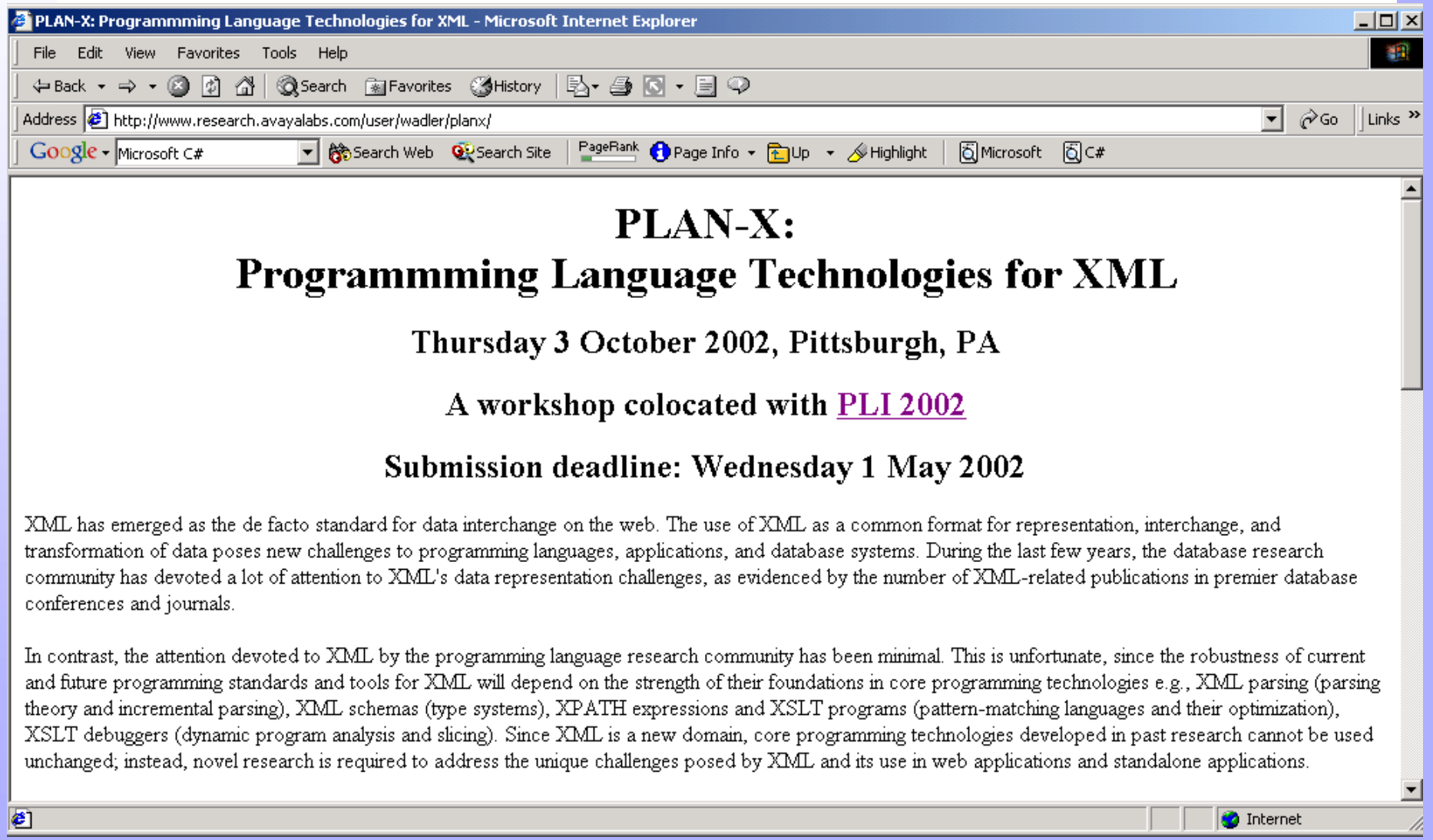
SVG DOM Example
What's Missing?

- A flat learning curve
  - Adds145 new data types!
  - Includes a complex inheritance hierarchy
  - Includes specialized types for lists, enumerations, units, etc.
- A general purpose solution
  - Requires highly specialized knowledge
  - Works only for navigating SVG
  - Each new vocabulary requires a new set of interfaces.
- Read-only
  - Requires DOM interfaces for modifying document.

```
circle.setAttribute("r", radius);        // NOT circle.r = radius;
```


Paths - W3C SVG 1.0 Specification - Candidate Recommendation 20000802 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address file:///C:/jcs/Work/Technology/Xml/SVG/CR%2020000802/paths.html#DOMInterfaces
Go Links
Basic Data Types and Interfaces - W3C SVG 1.0 Specification - Candidate Recommendation 20000802 - Microsoft Inte...
File Edit View Favorites Tools Help
Back Search Favorites History
Address file:///C:/jcs/Work/Technology/Xml/SVG/CR%2020000802/types.html#BasicDOMInterfaces
Go Links
Google 2 ECMAScript setProperty Search Web Search Site PageRank Page Info Up Highlight
IDL Definition
interface SVGList {
readonly attribute unsigned long numberOfItems;
void clear ( )
raises(DOMException);
Object initialize (in Object newItem)
raises(DOMException, SVGException);
Object createItem ( );
Object getItem (in unsigned long index)
raises(DOMException);
Object insertItemBefore (in Object newItem, in unsigned long index)
raises(DOMException, SVGException);
Object replaceItem (in Object newItem, in unsigned long index)
raises(DOMException, SVGException);
Object removeItem (in unsigned long index)
raises(DOMException);
Object appendItem (in Object newItem)
raises(DOMException, SVGException);
};
Done
My Computer
Basic Shapes - W3
File Edit View
Back
Address file:///C:
Google 2 ECMA
IDL De
Done
= 0;
= 1;
= 2;
= 3;
= 4;
= 5;
= 6;
= 7;
= 8;
= 9;
= 10;
= 11;
= 12;
= 13;
= 14;
= 15;
= 16;
= 17;
= 18;
= 19;

What's Next?


PLAN-X: Programmming Language Technologies for XML - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://www.research.avayalabs.com/user/wadler/planx/
Go Links
Google Microsoft C# Search Web Search Site PageRank Page Info Up Highlight Microsoft C#
PLAN-X:
Programmming Language Technologies for XML
Thursday 3 October 2002, Pittsburgh, PA
A workshop colocated with PLI 2002
Submission deadline: Wednesday 1 May 2002
XML has emerged as the de facto standard for data interchange on the web. The use of XML as a common format for representation, interchange, and transformation of data poses new challenges to programming languages, applications, and database systems. During the last few years, the database research community has devoted a lot of attention to XML's data representation challenges, as evidenced by the number of XML-related publications in premier database conferences and journals.
In contrast, the attention devoted to XML by the programming language research community has been minimal. This is unfortunate, since the robustness of current and future programming standards and tools for XML will depend on the strength of their foundations in core programming technologies e.g., XML parsing (parsing theory and incremental parsing), XML schemas (type systems), XPATH expressions and XSLT programs (pattern-matching languages and their optimization), XSLT debuggers (dynamic program analysis and slicing). Since XML is a new domain, core programming technologies developed in past research cannot be used unchanged; instead, novel research is required to address the unique challenges posed by XML and its use in web applications and standalone applications.
Internet

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Proposed Approach
- Conclusions
- Recommendations

Proposed Approach

- Add a native XML data type to ECMAScript
  - An *ordered* collection of XML properties with a name, base-object (i.e, parent) and set of XML attributes
  - XML Properties can be XML, comments, PIs or text
- Reuse existing operators and extend with semantics for XML (e.g., property accessors)
- Add a minimal set of new operators for common XML operations (e.g., searching and filtering)

Add a Native XML Object

```
// create a new XML object from a string
var order = new XML(“<order/>”);

// create an new XML object from a file
var doc = new XML(filename);

// create an XML wrapper for manipulating the document object
var doc = XML(document);
```

Reuse Familiar Operators

```
// get the customer's address from the order
var address = order.customer.address;

// get the second item from the order
var secondPrice = order.item[1];

// calculate the total price for the second item in the order
var secondTotal = order.item[1].price * order.item[1].quantity;

// change the quantity of the first item
order.item[0].quantity = 18;

// append a grand total to the order
order.total = grandTotal;
```

ToPrimative automatically gets values of leaf nodes

Assignment of primative value creates leaf node

New properties are always appended to end

Reuse Familiar Operators

```
// access children of second item by ordinal value instead of name
var description = order.item[1][0];          // get description
order.item[1][2] = 9.95;                     // change price

// access prices as an array
var prices = order.item.price;
var secondPrice = order.item.price[1];
order.item.price[1] = 19.95;

// iterate over all the items in the order
for (i in order.item) { … }

// create a list of selected items from the order
var selectItems = order.item[1] + order.item[3] + order.item[6];
```

Property accessor is also defined for XML lists

Addition operator concatenates XML values

Reuse Edition 4 Concepts

```
// declare XML typed variables
var order : XML;

// import specific XML types using an XML Schema
import PurchaseOrder.xsd;

// declare XML namespaces
namespace soap as “http://schemas.xmlsoap.org/soap/envelope/”;
namespace stock as “http://mycompany.com/stocks”;

// use qualified names to manipulate namespace qualified elements
var body = message.soap::Body;
message.soap::Body.stock::GetTradePrice.symbol = “MYCO”;
```

I still like the following syntax better:
XML order; // ;-)

New Operators

```
// attribute accessor: access XML attributes as specially named properties
var custid = order.customer.@custid;
order.item[1].@id = “123”;

// descendent operator: search without specifying full path
var prices = order..price;
var paragraphs = document..p;

// filtering predicate: e.g., get descriptions of items that cost less than $50
var cheapItems = order.item.(price < 50).description;

// get property list: get all the child elements of order
var orderData = order.*;

// get attribute list: get all XML attributes associated with the customer
var custAttributes = order.customer.@*;
```

XML Literals

```
// replace the customer address with a new one
order.customer.address = <address>
        <street>53 Party Lane</street>
        <city>Big Town</city>
        <state>Washington</state>
        <zip>98008</zip>
</address>;

// append a new empty item using nextItemNum as the id
order.item += <item id={nextItemNum++}/>;

// add a calculated prefix (e.g, Mr., Mrs.) in front of the customer name
order.customer.name = <prefix>{prefix}</prefix> + order.customer.name;

// replace the children of the customer element with empty elements
order.customer.* = <name/> + <address/>;
```

Parsing may be handled similar to RegEx literals

May embed expressions anywhere in literal

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Proposed Approach
- Conclusions
- Recommendations

Conclusions

- Scripters are inundated with XML processing tasks
- Current tools are complex and unfamiliar to scripter
- XML to Object mapping techniques are insufficient
- The proposed approach will
  - Empower more people to do more useful stuff with XML
    - Minimize the required knowledge, expertise, time and money
    - Scripters can start with little or no additional knowledge
  - Reduce code complexity, time to market and revision cycles
  - Decrease XML footprint requirements
  - Enable looser coupling between code and external data formats

Overview

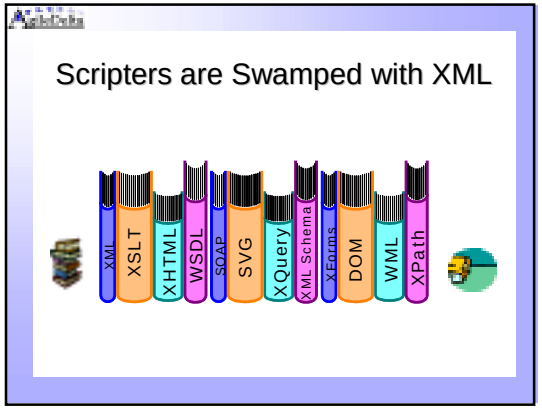
- The Problem
- The Opportunity
- Initial Attempts
- Proposed Approach
- Conclusions
- Recommendations

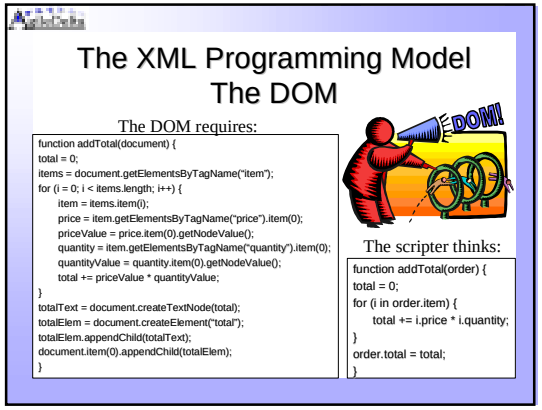
Recommended Actions

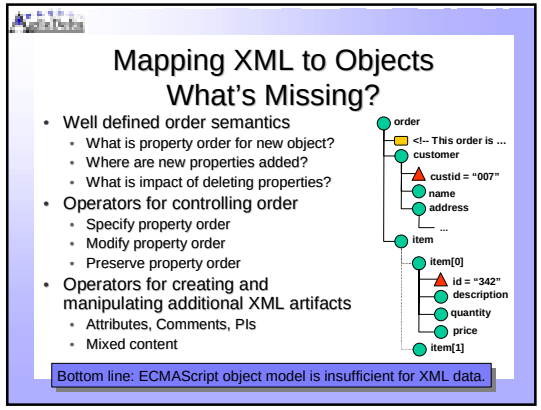
- Acknowledge importance of XML to scripters
- Recognize opportunity for ECMAScript
- Establish an XML sub-group of TC39/TG1 to
  - Review our proposal in detail
  - Develop ECMAScript extensions for XML
- Join our team in the interim to keep up with the latest developments!

Proposed Logistics

- Meetings: 1/2 - 1 day held coincident with TC39/TG1
  - Membership largely overlapping
  - Individuals may opt to participate in TG1, the XML sub-group or both groups
- Product:
  - Specification formalizing syntax and semantics required in addition to Edition 4
- Target date: June 2003
- Editor: John Schneider
- Convener: to be decided by TG1



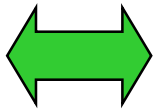




Questions and Discussion

The scripter thinks:

```
function addTotal(order) {
total = 0;
for (i in order.item) {
      total += i.price * i.quantity;
}
order.total = total;
}
```



XScript enables:

```
function addTotal(order) {
total = 0;
for (i in order.item) {
      total += i.price * i.quantity;
}
order.total = total;
}
```

Backup Slides

Why Standardize?

- Timing. If we don’t act now, market need will generate disparate, incompatible solutions
- Market. The benefits of this technology extend to a broad range of products
- Value. The network effects of an open standard are more valuable than a proprietary approach