ECMAScript for XML (E4X)

John Schneider

ECMA TC39/TG1/SG1

john.schneider@agiledelta.com

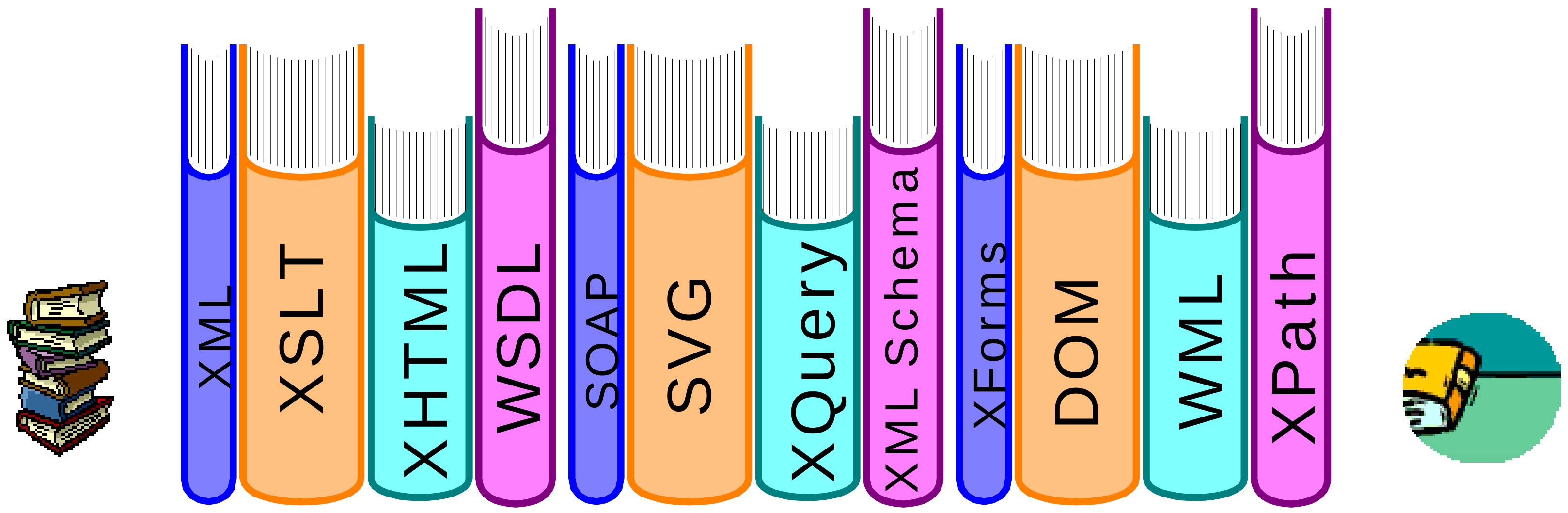
Overview

- The Problem
- The Opportunity
- Initial Attempts
- Our Approach
- Conclusions
- Current Status

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Our Approach
- Conclusions
- Current Status

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=“$total + ./price * ./quantity”/>
    </xsl:template>
    <xsl:template match=“*|/|comment()|processing-
    instruction()”>
                                             t()]”>
          <xsl:value-of “.”/>
          <xsl:apply-templates/>
          <total><xsl:value-of select=“$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
- Our Approach
- Conclusions
- Current Status

The Opportunity

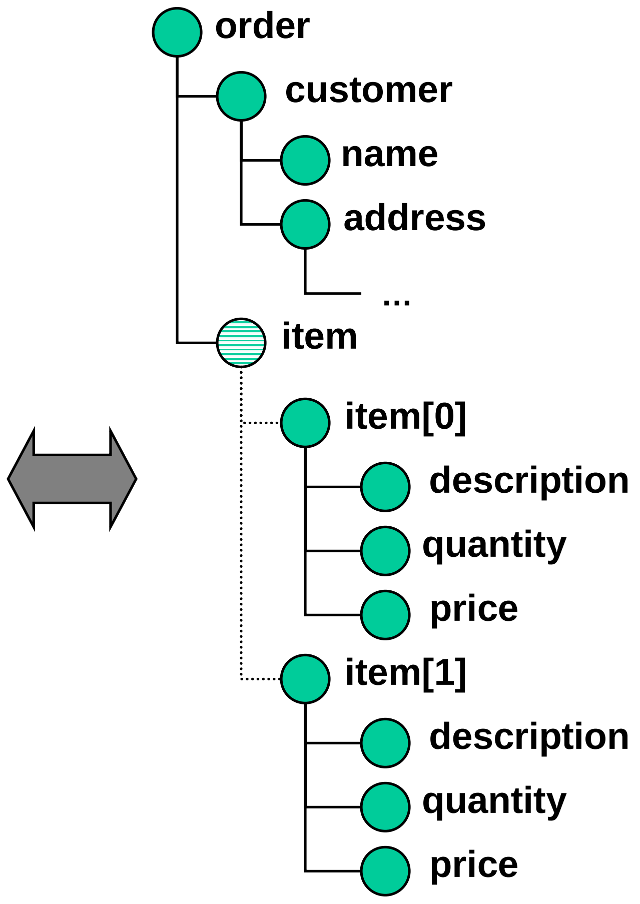
- Provide a simple, familiar, general purpose programming model for XML that:
  - Leverages existing skills and knowledge
  - Reuses familiar concepts, operators and syntax
  - Flattens the learning curve
  - Minimizes need for specialized skills and knowledge
  - Enables scripters immediately with little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

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Our Approach
- Conclusions
- Current Status

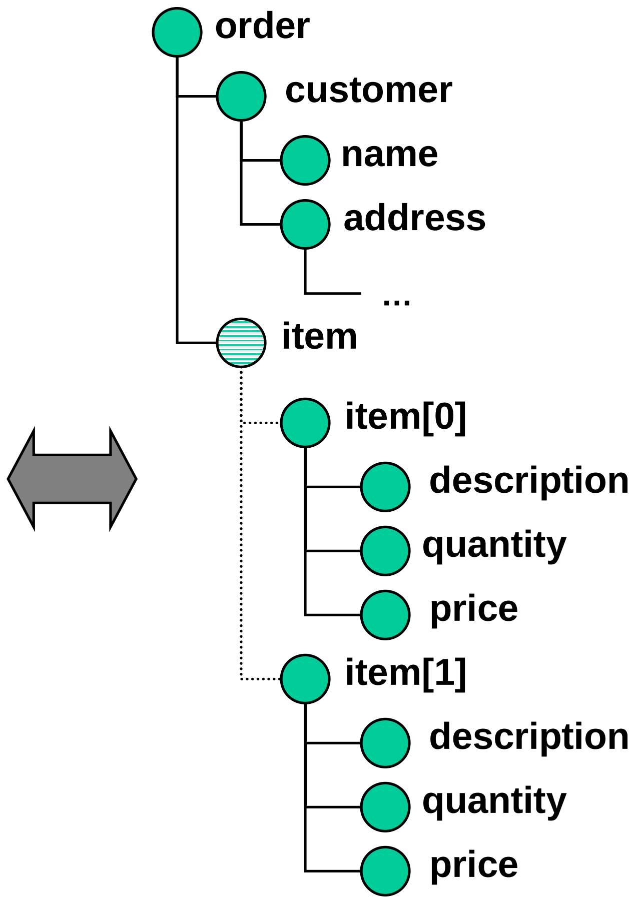
Mapping XML to Objects

```
<order>
    <customer>
        <name>I. Wannabuy</name>
        <address> … </address>
    </customer>
    <item>
        <description>Small Rodent, Generic</description>
        <quantity>35</quantity>
        <price>6.99</price>
    </item>
    <item>
        <description>Catapult</description>
        <quantity>1</quantity>
        <price>149.95</price>
    </item>
</order>
```



Mapping XML to Objects

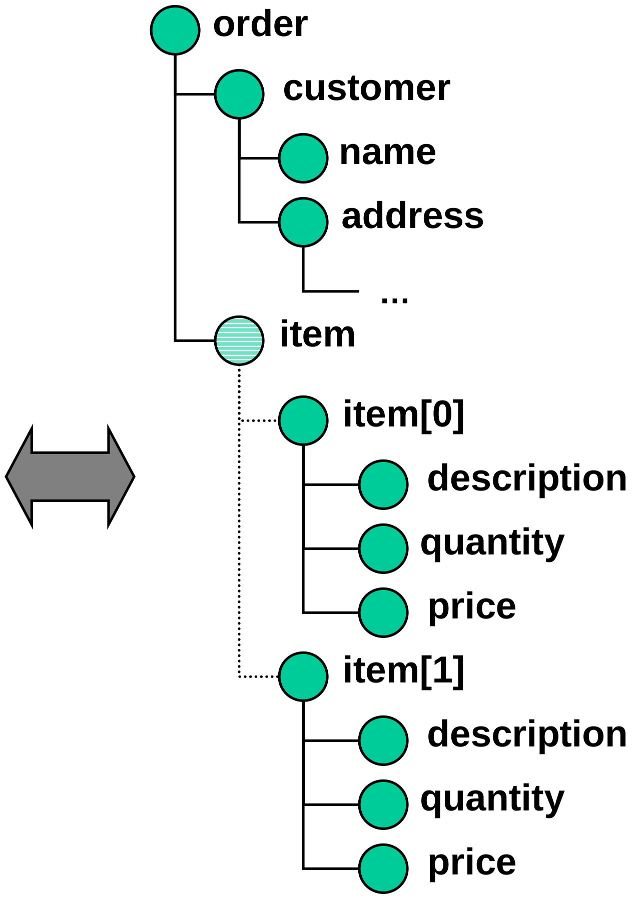
```
order = {
    customer: {
        name: “I. Wannabuy”,
        address: … ,
    },
    item [
        {
            description: “Small Rodent, Generic”,
            quantity: 35,
            price: 29.99
        },
        {
            description: “Catapult”,
            quantity: 1,
        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



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

Mapping XML to Objects

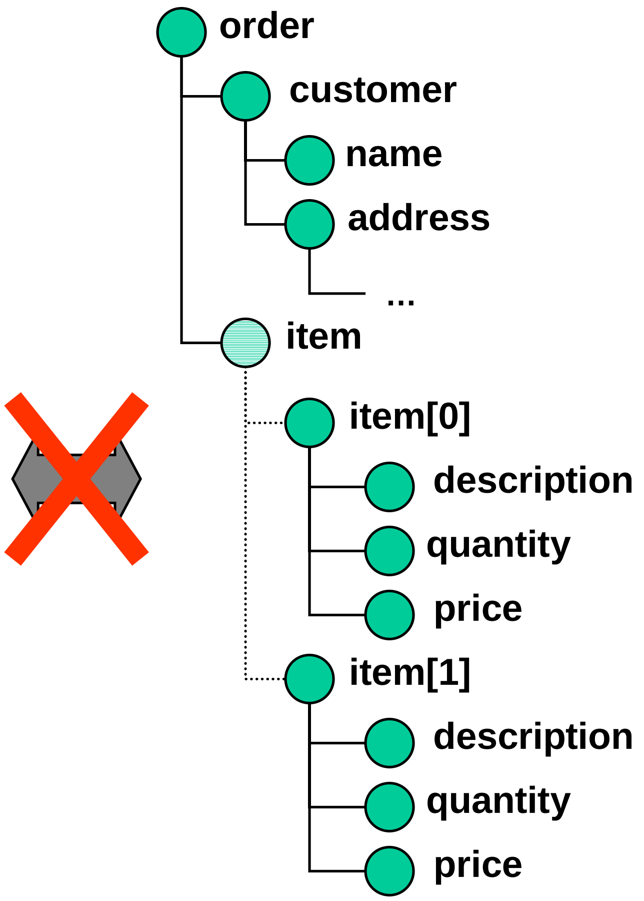
```
order = {
    item [
        {
            quantity: 35,
            price: 29.99
            description: “Small Rodent, Generic”,
        },
        {
            price: 149.95
            description: “Catapult”,
            quantity: 1,
        }
    ]
 customer: {
        name: “I. Wannabuy”,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



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

Mapping XML to Objects

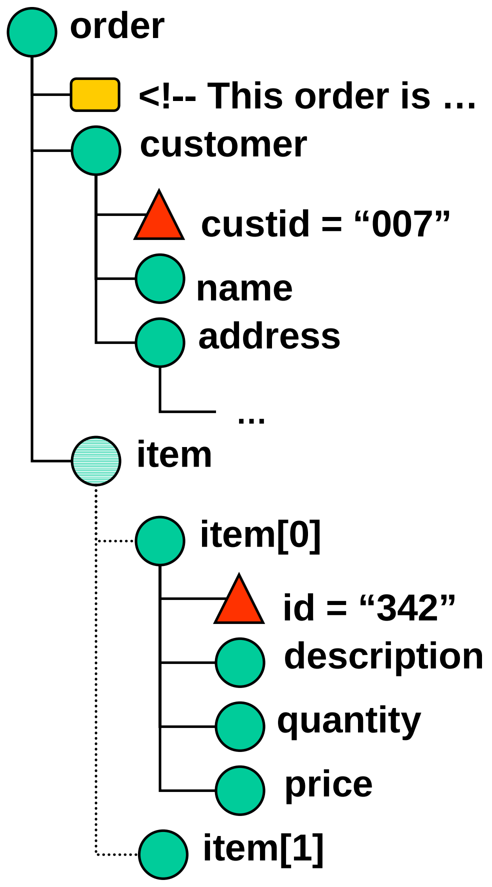
```
<order>
    <item>
        <quantity>35</quantity>
        <price>29.99</price>
        <description>Small Rodent, Generic</description>
    </item>
    <item>
        <price>149.95</price>
        <description>Catapult</description>
        <quantity>1</quantity>
    </item>
    <customer>
    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



But, order is critical in XML.

Mapping XML to Objects
What's Missing?

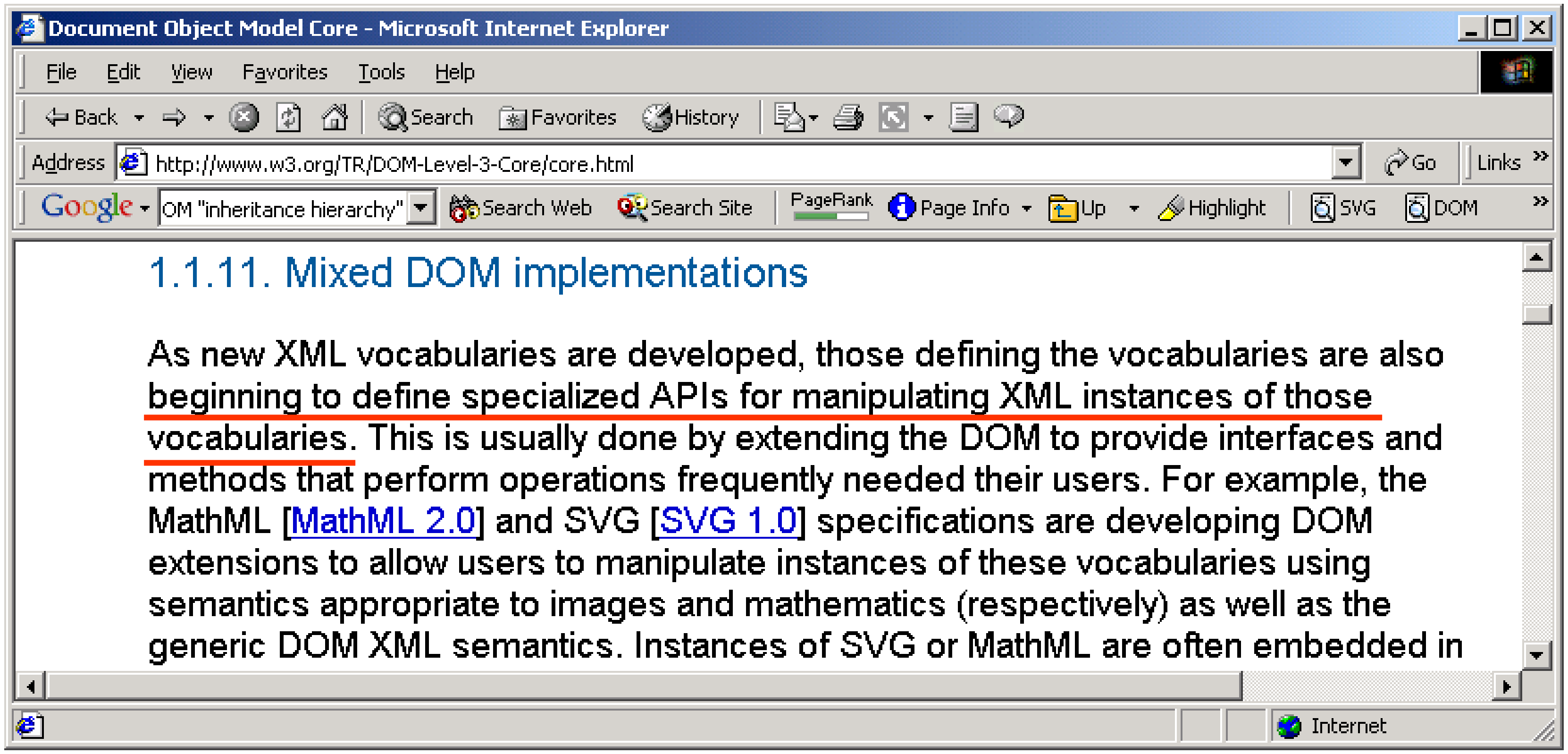
- Well defined order semantics
  - What is property order for new object?
  - Where are new properties added?
  - What is impact of deleting properties?
- Operators for controlling order
  - Specify property order
  - Modify property order
  - Preserve property order
- Operators for creating and manipulating 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


Document Object Model Core - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://www.w3.org/TR/DOM-Level-3-Core/core.html
Go Links
Google OM "inheritance hierarchy" Search Web Search Site PageRank Page Info Up Highlight SVG DOM
Internet

1.1.11. Mixed DOM implementations

As new XML vocabularies are developed, those defining the vocabularies are also beginning to define specialized APIs for manipulating XML instances of those vocabularies. This is usually done by extending the DOM to provide interfaces and methods that perform operations frequently needed their users. For example, the MathML [MathML 2.0] and SVG [SVG 1.0] specifications are developing DOM extensions to allow users to manipulate instances of these vocabularies using semantics appropriate to images and mathematics (respectively) as well as the generic DOM XML semantics. Instances of SVG or MathML are often embedded in

SVG DOM Example

- A great step forward!

Standard DOM:

```
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;
}
```

SVG DOM:

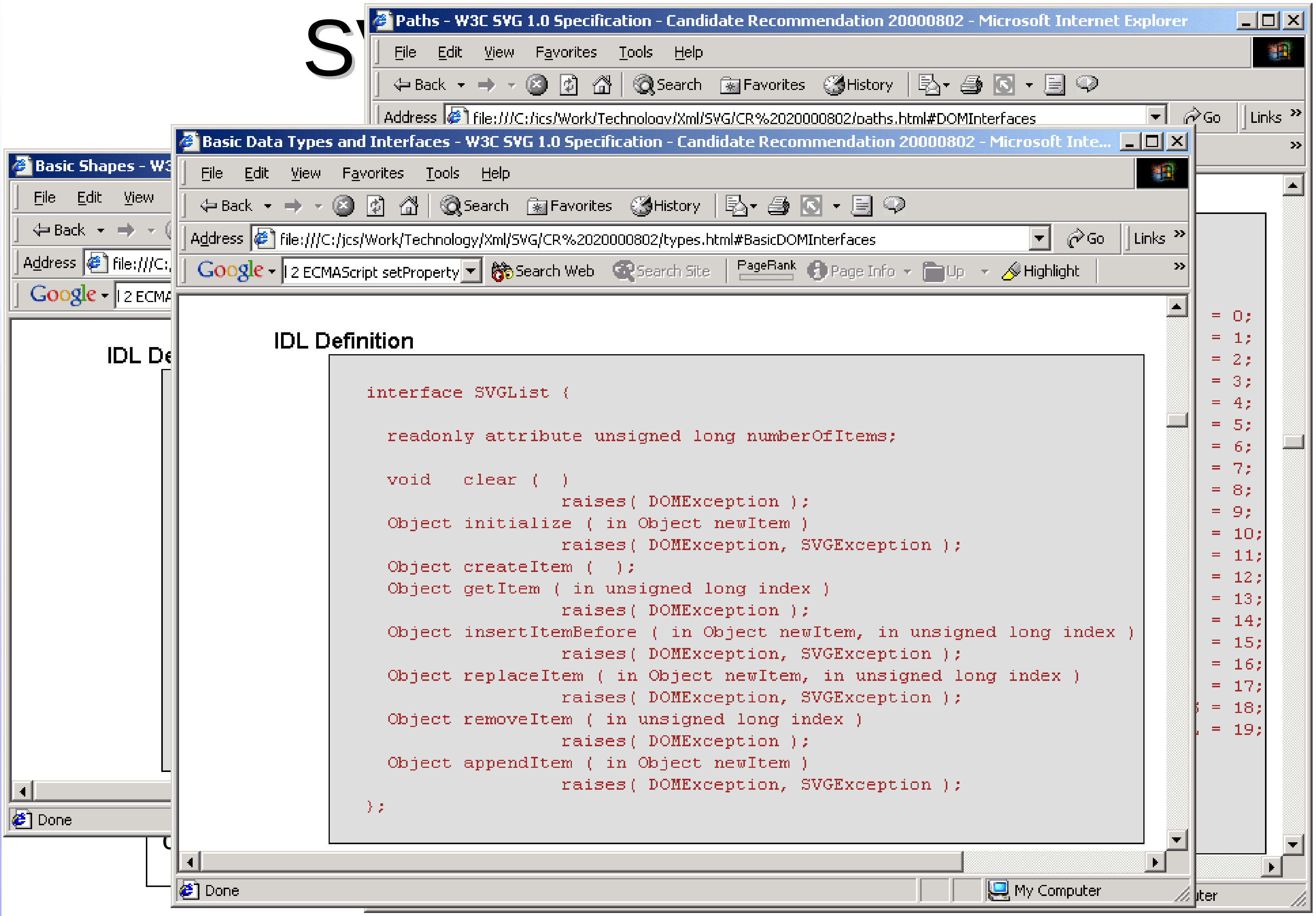
```
function pointInsideRect(point, rect) {
        if ((point.x > rect.x)
        && (point.x < rect.x + rect.width)
        && (point.y > rect.y)
        && (point.y < rect.y + rect.height))
              return true;
        else
              return false;
}
```

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;
```

S



Paths - W3C SVG 1.0 Specification - Candidate Recommendation 20000802 - Microsoft Internet Explorer

Address file:///C:/jcs/Work/Technology/Xml/SVG/CR%2020000802/paths.html#DOMInterfaces

Basic Shapes - W3

IDL De

Basic Data Types and Interfaces - W3C SVG 1.0 Specification - Candidate Recommendation 20000802 - Microsoft Inte...

Address file:///C:/jcs/Work/Technology/Xml/SVG/CR%2020000802/types.html#BasicDOMInterfaces

IDL Definition

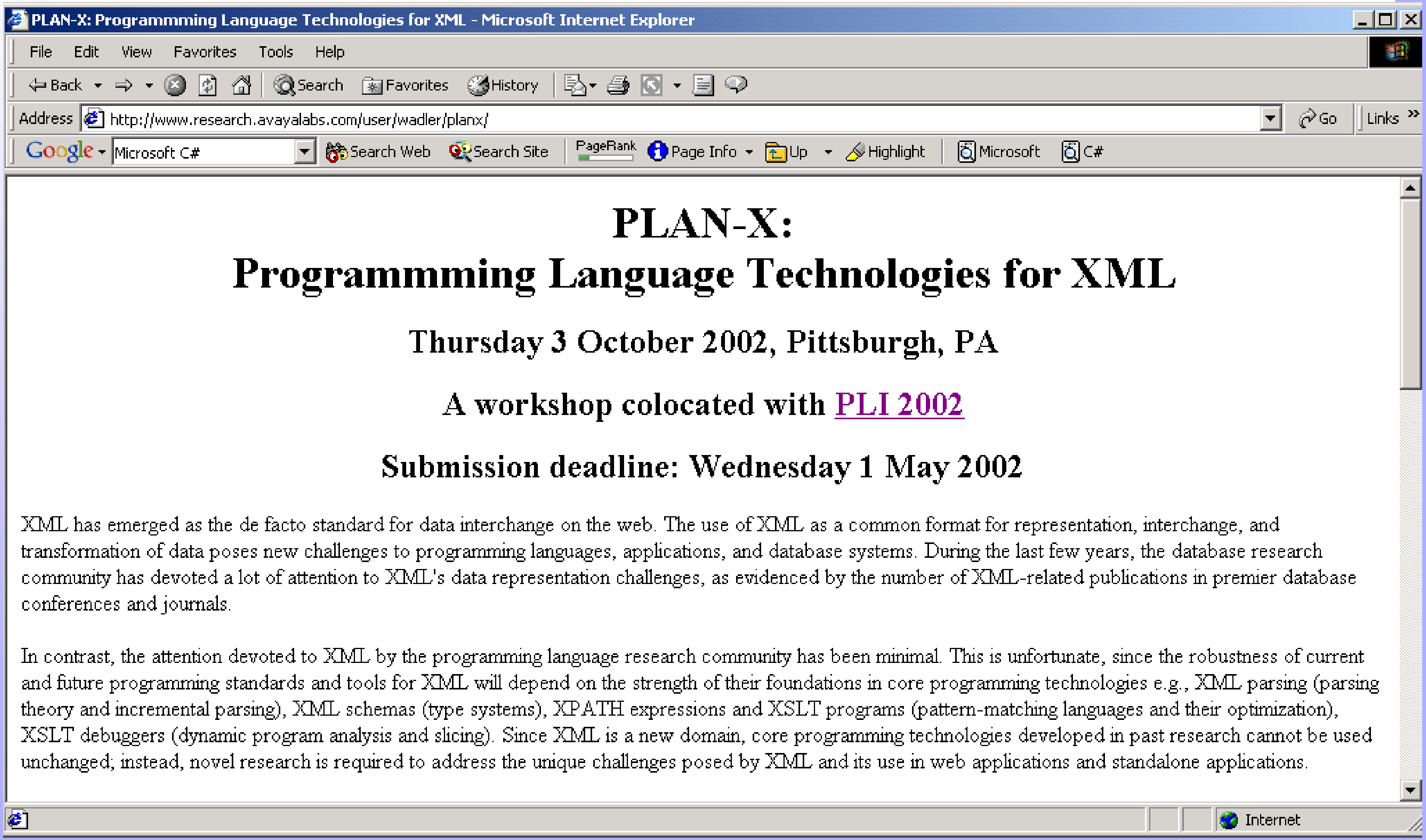
```
interface SVGList {

  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 );
};
```

```
= 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#
Internet

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.

Overview

- The Problem
- The Opportunity
- Initial Attempts
- Our Approach
- Conclusions
- Current Status

E4X 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(file);

// 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 secondItem = 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;
```

ToPrimitive automatically gets values of leaf nodes

Assignment of primitive value creates leaf node

New properties are always appended to end

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”;
```

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.@*;
```

Reduces dependencies on containment hierarchy

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
- Our Approach
- Conclusions
- Current Status

Conclusions

- Scripters are inundated with XML processing tasks
- Current tools are complex and unfamiliar to scripter
- XML to Object mapping techniques are insufficient
- ECMAScript for XML will
  - Empower more people to do more useful things 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
- Our Approach
- Conclusions
- Current Status

E4X Sub-group

- Established by TC39/TG1 on 13 June 2002
- Logistics
  - Meetings: Every 6 weeks following TG1 meeting
  - Convener: Peter Torr
  - Editor: John Schneider
- Decoupled from Edition 4 work with separate meetings and resources

Terms of Reference

Name: ECMAScript for XML (E4X)

Scope:

To standardize the syntax and semantics of a general purpose, cross platform, vendor neutral set of programming language extensions adding native XML support to ECMAScript

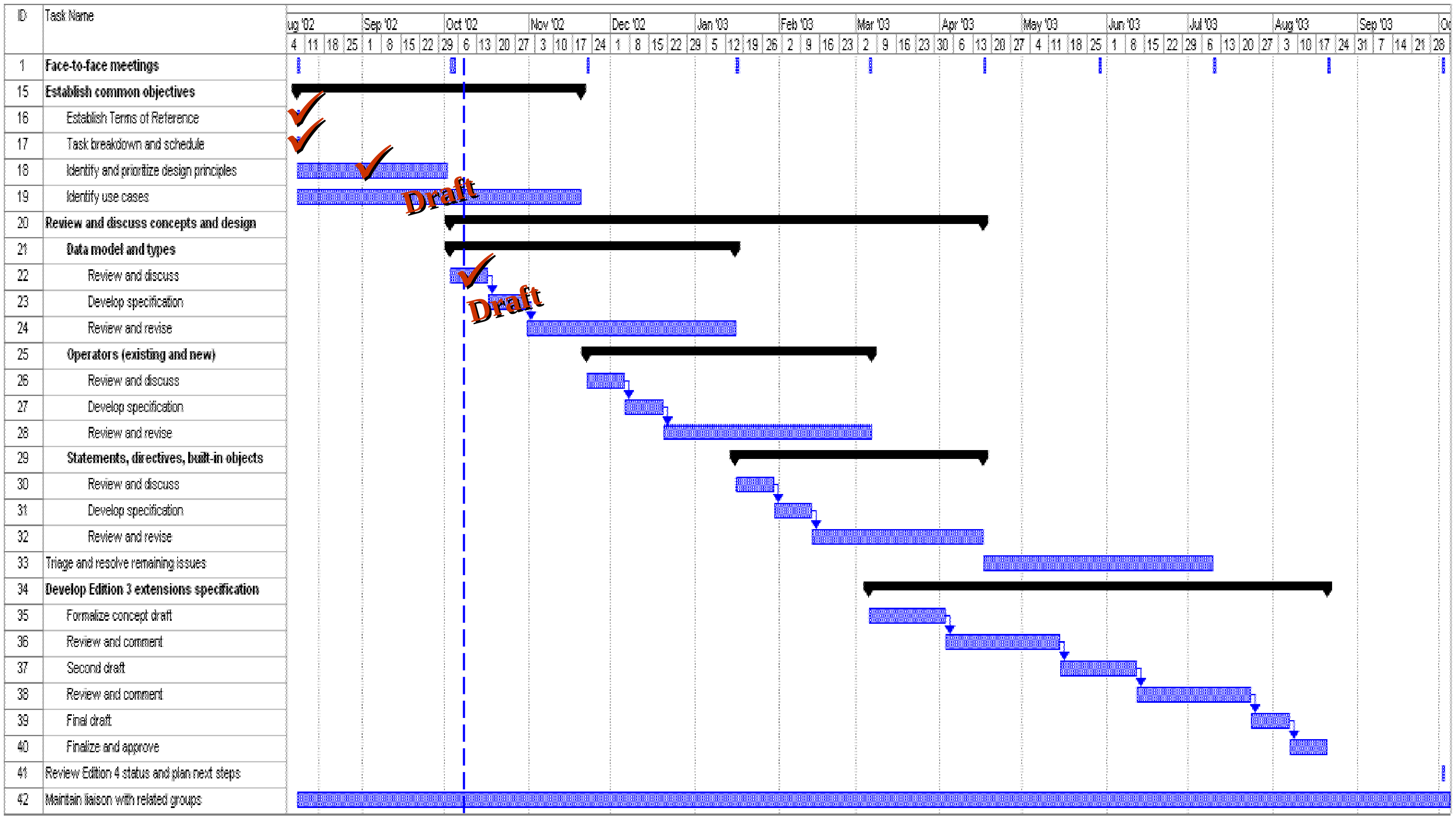
Program of Work:

- Develop a standard set of language extensions to add native XML support to ECMAScript
- Facilitate integration of developed extensions into the ECMAScript Language Specification
- On completion of tasks 1 and 2, investigate the future direction of XML support in ECMAScript and consider proposals for complementary or additional technology
- Maintain liaison with appropriate ECMA and external standards bodies

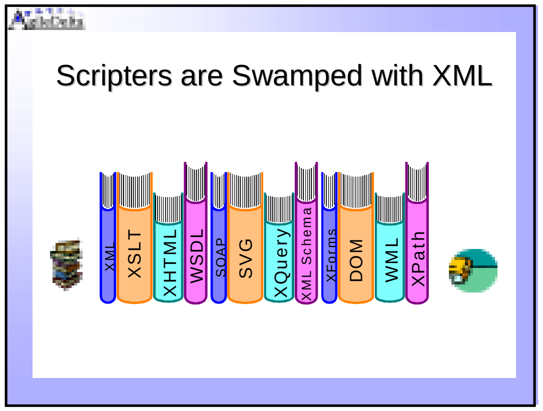
Task Breakdown

- Establish objectives, design principles and use cases
- Review and agree on language extension concepts
  - Data model and type extensions
  - Extended semantics for existing operators
  - New operators
  - Statements and directives
  - Built-in classes (properties and methods)
- Develop specification formalizing syntax and semantics of language extensions for ECMAScript Edition 3
- Develop specification formalizing syntax and semantics of language extensions for ECMAScript Edition 4
- Integrate with ECMAScript Language Specification

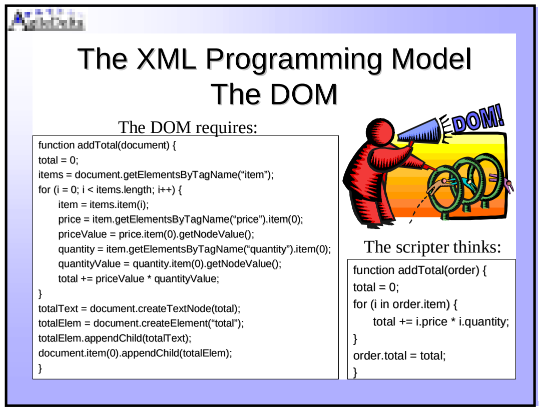
Schedule



ID	Task Name
1	**Face-to-face meetings**
15	**Establish common objectives**
16	Establish Terms of Reference
17	Task breakdown and schedule
18	Identify and prioritize design principles
19	Identify use cases
20	**Review and discuss concepts and design**
21	**Data model and types**
22	Review and discuss
23	Develop specification
24	Review and revise
25	**Operators (existing and new)**
26	Review and discuss
27	Develop specification
28	Review and revise
29	**Statements, directives, built-in objects**
30	Review and discuss
31	Develop specification
32	Review and revise
33	Triage and resolve remaining issues
34	**Develop Edition 3 extensions specification**
35	Formalize concept draft
36	Review and comment
37	Second draft
38	Review and comment
39	Final draft
40	Finalize and approve
41	Review Edition 4 status and plan next steps
42	Maintain liaison with related groups



Scripters are Swamped with XML



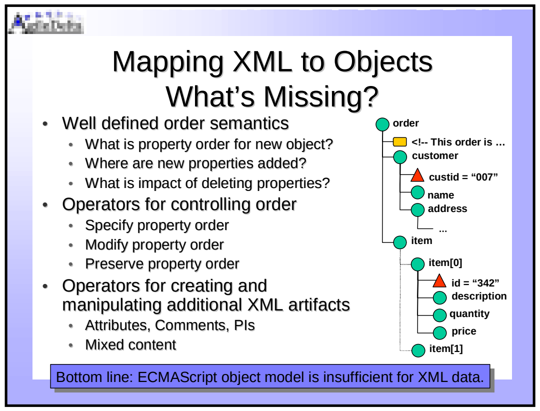
The XML Programming Model
The DOM

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;
}
```



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.

Questions and Discussion

The scripter thinks:

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

E4X 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