Ecma/TC39-TG1/2004/4

Ecma/TC39/2004/16

Ecma/GA/2004/25

ECMAScript for XML (E4X) Proposal for TC39 Approval

ECMA TC39/TG1

John Schneider

john.schneider@agiledelta.com

Overview

- The Problem (review)
- The E4X Solution
- Current Status
- Conclusions
- Recommendations

Overview

- The Problem (review)
- The E4X Solution
- Current Status
- 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 each (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 each (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=“$total + ./price * ./quantity”/>
      </xsl:template>
      <xsl:template match=“*|/|comment()|processing-
      instruction()”>
            <xsl:value-of select=“.”/>
            <xsl:apply-templates/>
            <total><xsl:value-of select=“$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 each (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 each (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

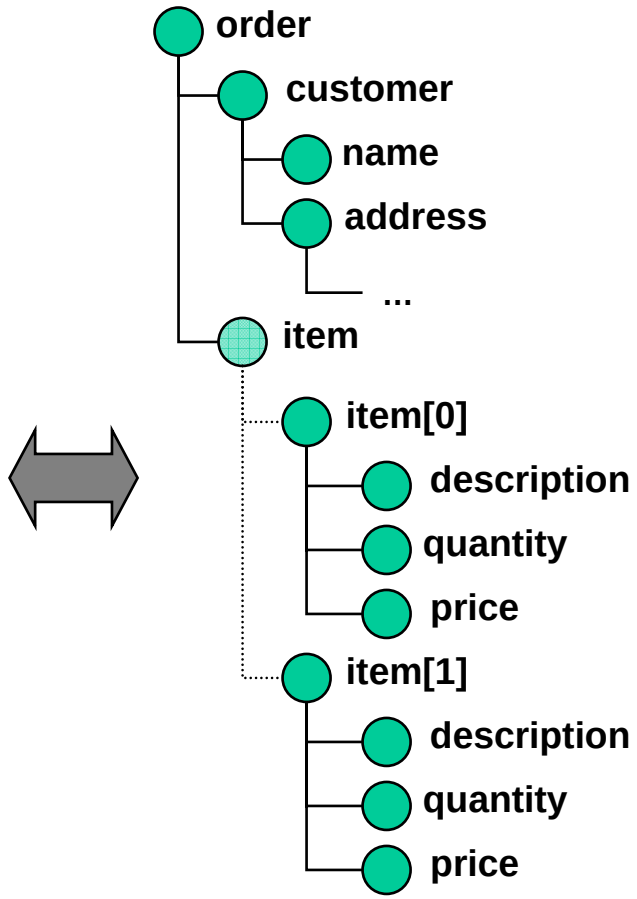
- The Problem (review)
- The E4X Solution
- Current Status
- Conclusions
- Recommendations

E4X Objective

- 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

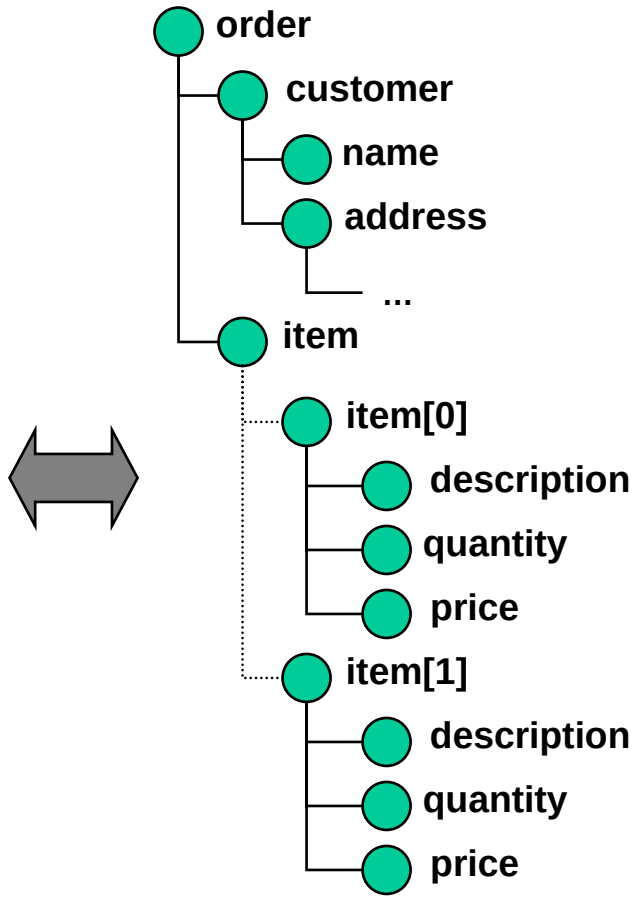
Mapping XML to Objects

```
<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

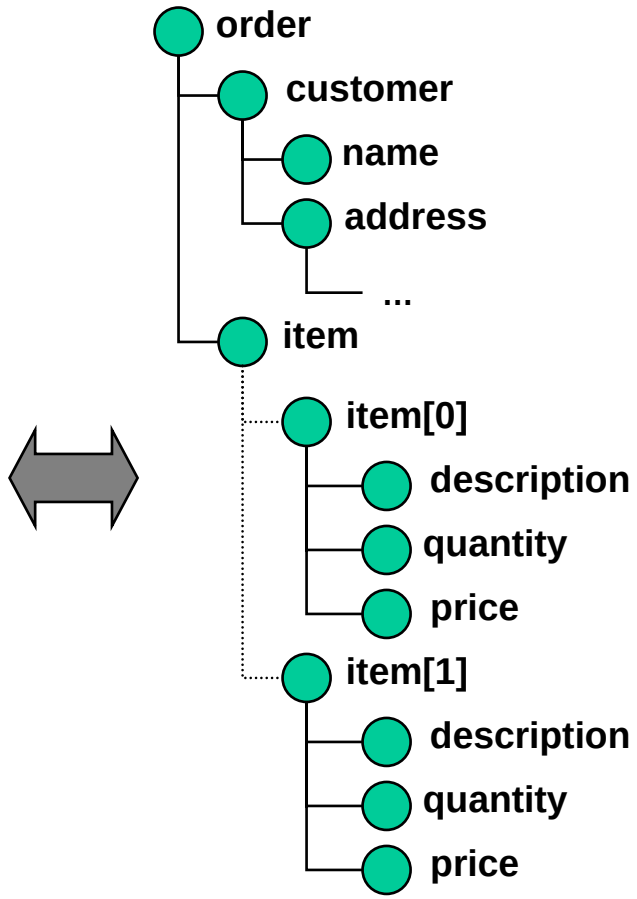
```
order = {
    customer: {
        name: “I. Wannabuy”,
        address: … ,
    },
    item [
        {
            description: “Small Rodent, Generic”,
            quantity: 35,
            price: 29.99
        },
        {
            description: “Catapult”,
            quantity: 1,
        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



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

Mapping XML to Objects

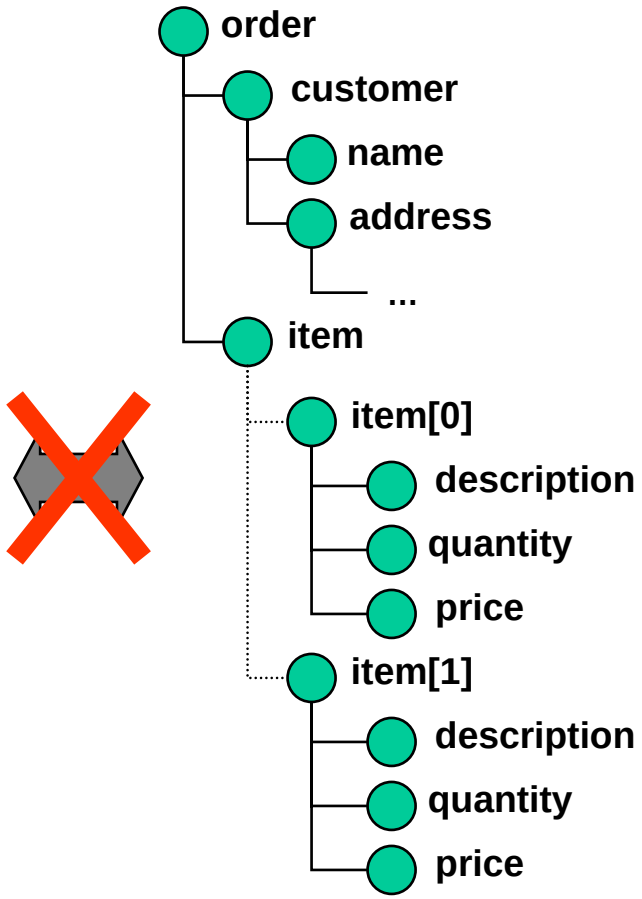
```
order = {
    item [
        {
            quantity: 35,
            price: 29.99
            description: “Small Rodent, Generic”,
        },
        {
            price: 149.95
            description: “Catapult”,
            quantity: 1,
        }
    ]
 customer: {
        name: “I. Wannabuy”,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



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

Mapping XML to Objects

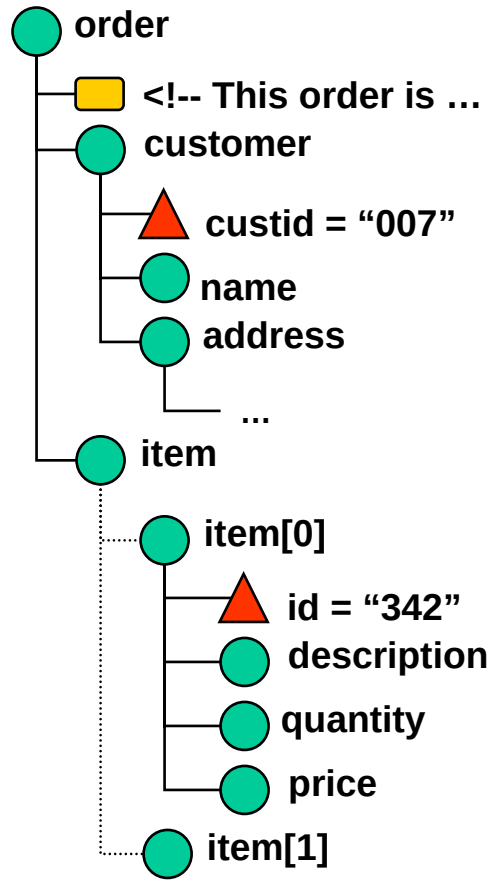
```
<order>
    <item>
        <quantity>35</quantity>
        <price>29.99</price>
        <description>Small Rodent, Generic</description>
    </item>
    <item>
        <price>149.95</price>
        <description>Catapult</description>
        <quantity>1</quantity>
    </item>
    <customer>
        <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
  - Namespaces, Mixed content



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

E4X Approach

- Add native XML data types to ECMAScript
  - An *ordered* collection of properties with a name, base-object (i.e, parent) and set of XML attributes
  - Properties represent elements, comments, PIs or text
  - Property names can be QNames in Namespaces
- 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)

Overview

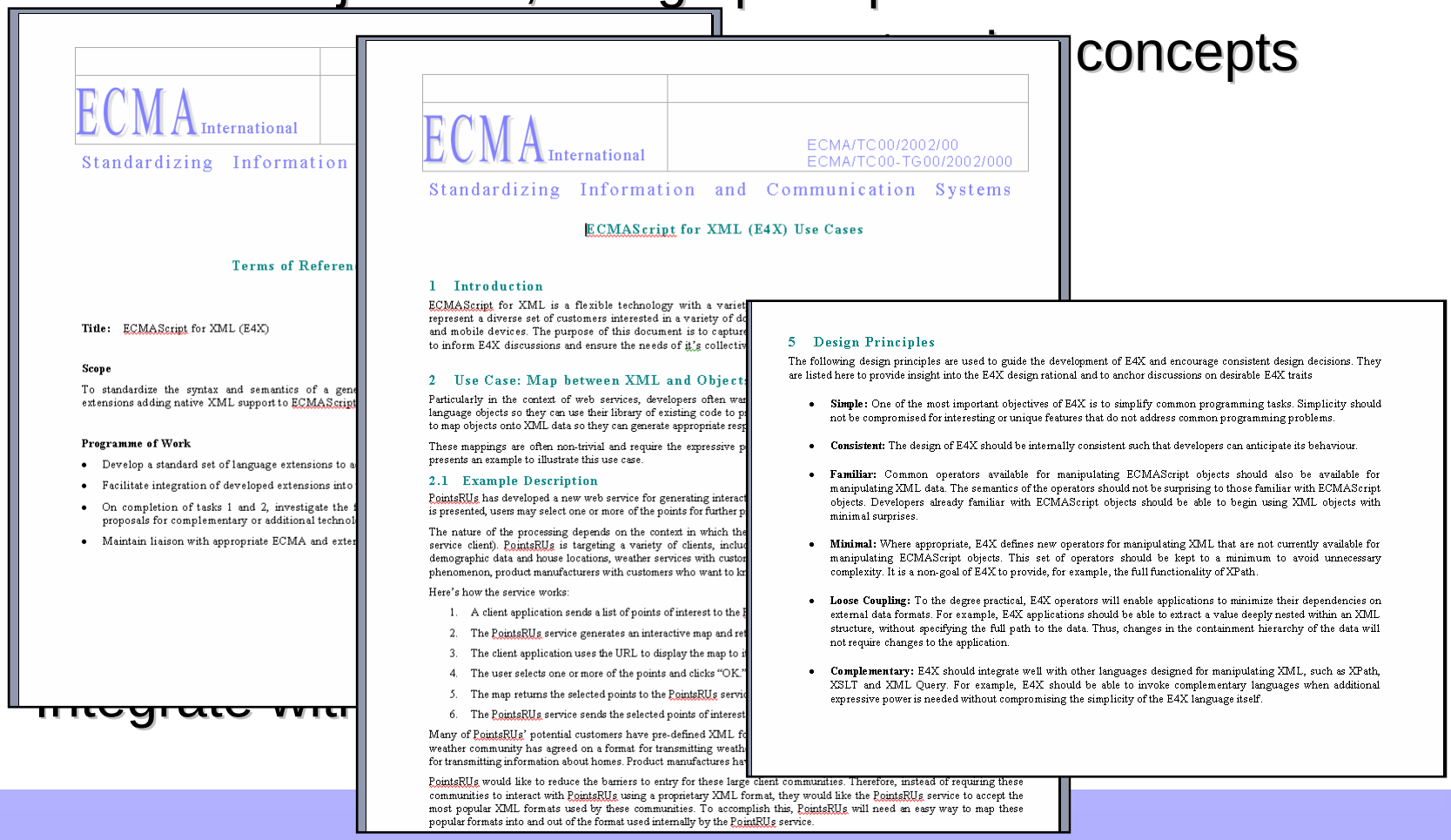
- The Problem (review)
- The E4X Solution
- Current Status
- Conclusions
- Recommendations

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

Task Breakdown

- ✓ Establish objectives, design principles and use cases
- concepts
- integrate with



ECMA International

Standardizing Information

Terms of Referen

Title: ECMAScript for XML (E4X)

Scope

To standardize the syntax and semantics of a gene
extensions adding native XML support to ECMAScript

Programme of Work

- Develop a standard set of language extensions to a
- Facilitate integration of developed extensions into
- On completion of tasks 1 and 2, investigate the f
proposals for complementary or additional technol
- Maintain liaison with appropriate ECMA and exter

ECMA International

ECMA/TC00/2002/00
ECMA/TC00-TG00/2002/000

Standardizing Information and Communication Systems

ECMAScript for XML (E4X) Use Cases

1 Introduction

ECMAScript for XML is a flexible technology with a variet
represent a diverse set of customers interested in a variety of do
and mobile devices. The purpose of this document is to capture
to inform E4X discussions and ensure the needs of it's collectiv

2 Use Case: Map between XML and Object

Particularly in the context of web services, developers often wan
language objects so they can use their library of existing code to p
to map objects onto XML data so they can generate appropriate resp

These mappings are often non-trivial and require the expressive p
presents an example to illustrate this use case.

2.1 Example Description

PointsRUs has developed a new web service for generating interact
is presented, users may select one or more of the points for further p

The nature of the processing depends on the context in which the
service client). PointsRUs is targeting a variety of clients, includ
demographic data and house locations, weather services with custo
phenomenon, product manufacturers with customers who want to kn

Here's how the service works:

1. A client application sends a list of points of interest to the P
2. The PointsRUs service generates an interactive map and ret
3. The client application uses the URL to display the map to it
4. The user selects one or more of the points and clicks "OK."
5. The map returns the selected points to the PointsRUs servic
6. The PointsRUs service sends the selected points of interest

Many of PointsRUs' potential customers have pre-defined XML fo
weather community has agreed on a format for transmitting weath
for transmitting information about homes. Product manufactures ha

PointsRUs would like to reduce the barriers to entry for these large client communities. Therefore, instead of requiring these communities to interact with PointsRUs using a proprietary XML format, they would like the PointsRUs service to accept the most popular XML formats used by these communities. To accomplish this, PointsRUs will need an easy way to map these popular formats into and out of the format used internally by the PointRUs service.

5 Design Principles

The following design principles are used to guide the development of E4X and encourage consistent design decisions. They are listed here to provide insight into the E4X design rational and to anchor discussions on desirable E4X traits

- **Simple:** One of the most important objectives of E4X is to simplify common programming tasks. Simplicity should not be compromised for interesting or unique features that do not address common programming problems.
- **Consistent:** The design of E4X should be internally consistent such that developers can anticipate its behaviour.
- **Familiar:** Common operators available for manipulating ECMAScript objects should also be available for manipulating XML data. The semantics of the operators should not be surprising to those familiar with ECMAScript objects. Developers already familiar with ECMAScript objects should be able to begin using XML objects with minimal surprises.
- **Minimal:** Where appropriate, E4X defines new operators for manipulating XML that are not currently available for manipulating ECMAScript objects. This set of operators should be kept to a minimum to avoid unnecessary complexity. It is a non-goal of E4X to provide, for example, the full functionality of XPath.
- **Loose Coupling:** To the degree practical, E4X operators will enable applications to minimize their dependencies on external data formats. For example, E4X applications should be able to extract a value deeply nested within an XML structure, without specifying the full path to the data. Thus, changes in the containment hierarchy of the data will not require changes to the application.
- **Complementary:** E4X should integrate well with other languages designed for manipulating XML, such as XPath, XSLT and XML Query. For example, E4X should be able to invoke complementary languages when additional expressive power is needed without compromising the simplicity of the E4X language itself.

Task Breakdown

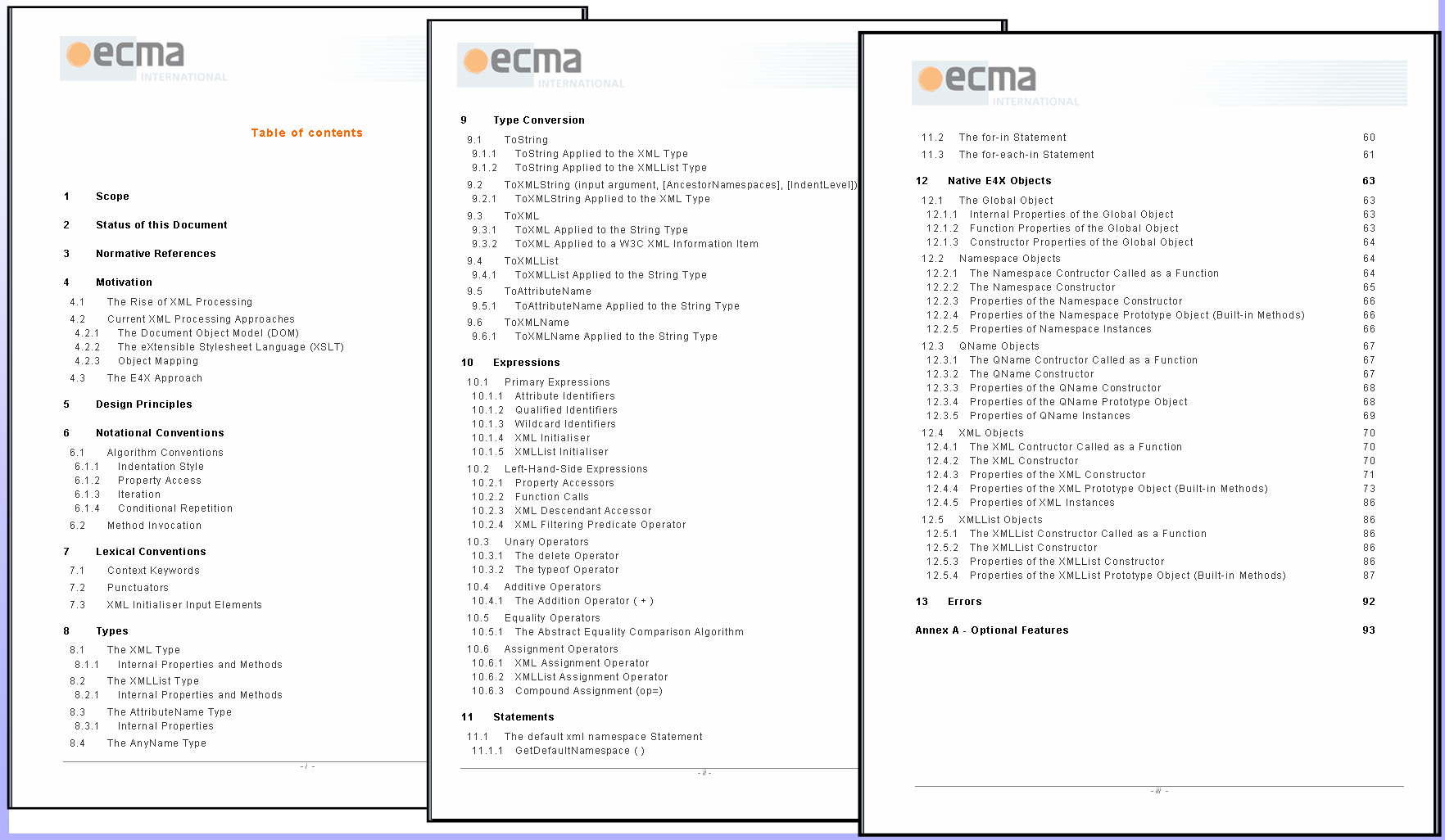
- ✓ Establish objectives, design principles and use cases
- ✓ Review and agree on language extension concepts
  - ✓ Data model and type extensions
  - ✓ Extended semantics for existing operators
  - ✓ New operators
  - ✓ Statements and directives
  - ✓ Built-in classes (properties and methods)

14 face-to-face meetings
5 teleconferences
E-mail discussions

- ✓ 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

17 working drafts
5 final draft candidates

Specification Review & Demo



ecma INTERNATIONAL

Table of contents

ecma INTERNATIONAL

ecma INTERNATIONAL

Implementations

- All active TG1 members have at least one implementation or plan to obtain one
- Several non-members have also indicated plans to support E4X
- Several mobile implementations in the works
- BEA plans to contribute an implementation to the Mozilla/Rhino open source project

Overview

- The Problem (review)
- The E4X Solution
- Current Status
- Conclusions
- Recommendations

Conclusions

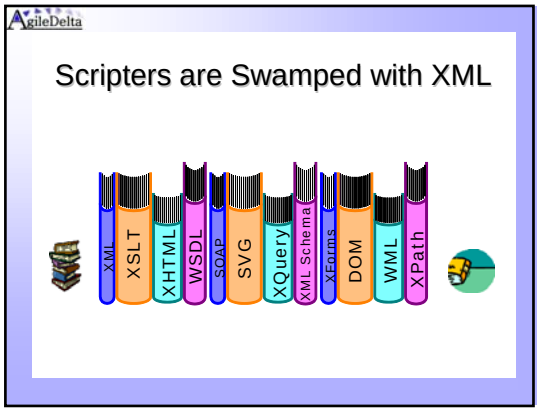
- Scripters are inundated with XML processing tasks
- Current XML techniques are complex and unfamiliar to the scripter
- ECMAScript for XML
  - Drastically simplifies creating, navigating and manipulating XML for one of the largest developer communities worldwide
    - Minimizes required knowledge, expertise, time and resources
    - Requires little or no additional knowledge
  - Is the first mainstream programming language with native support for XML and the only mainstream XML language to support XML updates
  - Reduces code complexity, time to market and revision cycles
  - Decreases XML footprint requirements
  - Enables looser coupling between code and external data sources
- Several companies have created independent implementations
- ECMA TC39/TG1 is very pleased with the final draft of the E4X specification and agrees it is ready for approval

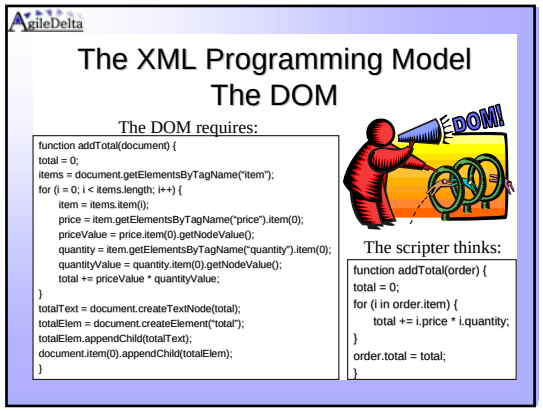
Overview

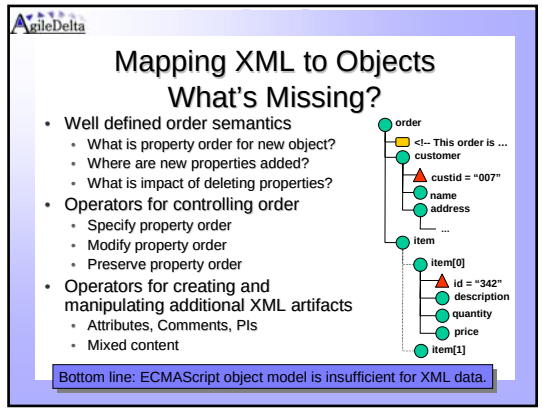
- The Problem (review)
- The E4X Solution
- Current Status
- Conclusions
- Recommendations

Recommendations

- ECMA TC39/TG1 recommends ECMA TC39
  - Acknowledge the importance of XML to scripters
  - Recognize the value of E4X for processing XML
  - Approve the final 02-26-04 draft of the E4X specification, as modified, for adoption by the ECMA GA as the first edition of the E4X standard



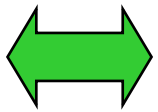




Questions and Discussion

The scripter thinks:

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



E4X enables:

```
function addTotal(order) {
total = 0;
for each (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

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

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