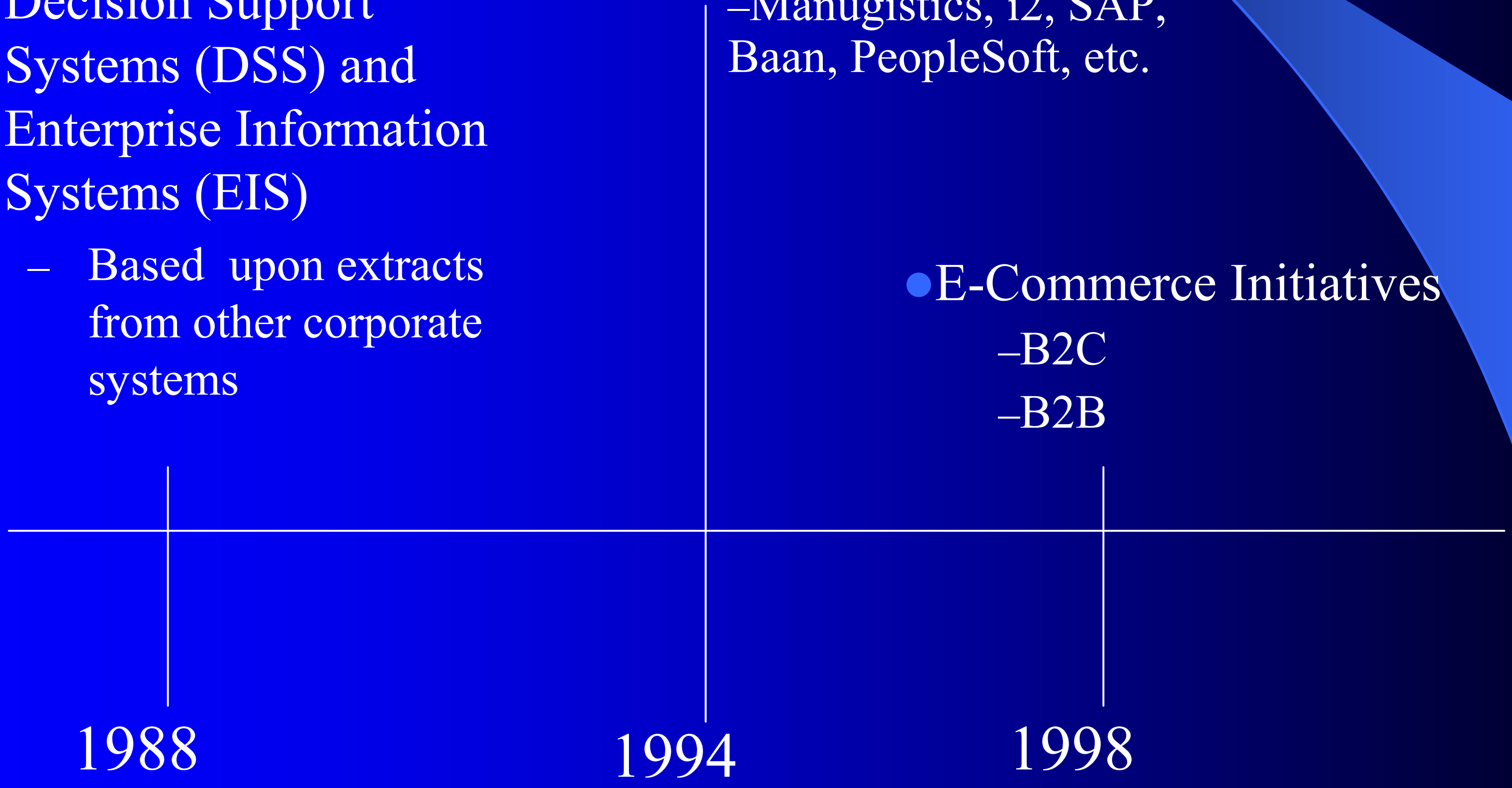
Enterprise Application Integration Tutorial

OMG EAI Workshop
February 7-9, 2000
Lake Buena Vista, FL

JP Morgenthal
Chief Technology Officer
XMLSolutions, Inc.
jpm@xmls.com

Introduction to EAI

Origins of EAI



- Decision Support Systems (DSS) and Enterprise Information Systems (EIS)
  - Based upon extracts from other corporate systems

1988

- Emergence of tools to support enterprise planning and management
  - Manugistics, i2, SAP, Baan, PeopleSoft, etc.

1994

- E-Commerce Initiatives
  - B2C
  - B2B

1998

State of Integration Between Enterprise Systems

- Multiple departmental and divisional systems
- Diverse hardware and operating systems
- Diverse data formats
- Inconsistent usage of industry terms

What is the Goal of EAI?

“The seamless integration of business processes for the purposes of conducting business electronically.”

“The sharing and/or exchange of data between systems for the purpose of providing a unified interface.”

Steps to Mastering EAI

1. Establish common semantics across communicating entities
2. Establish published formats for delivery of semantic information
3. Establish well-defined pathways for delivering information to other systems
4. Establish the rules of engagement

Establishing Common Semantics Across the Enterprise

First Rule of EAI

“Usage defines meaning and must be consistent, otherwise the meaning is broken”

ASSUME

EAI Problem Analogy

- a + b = c
  - If c is to be a number, a and b must be a number
  - If c is to be a string, a and b must be strings
  - If a is a number and b is string we break the meaning of the addition (+) operator

EAI in the Real World

- Companies define their business entities by the processes that they are part of
- When processes must be integrated, the same business entities have multiple meanings
- EAI is the reconciliation of the meanings of business entities

EAI Doesn’t Change Meaning

- Integration bridges the gap between two meanings
- Integration does not force meanings to change to fit the requirements of integration
- Thus, we map from one meaning to another

Framing The Problem

Data trapped in disparate applications and formats	Systems developed departmentally with no focus on the enterprise
Meaning of data has been skewed throughout the company	There are no “silver bullet” solutions

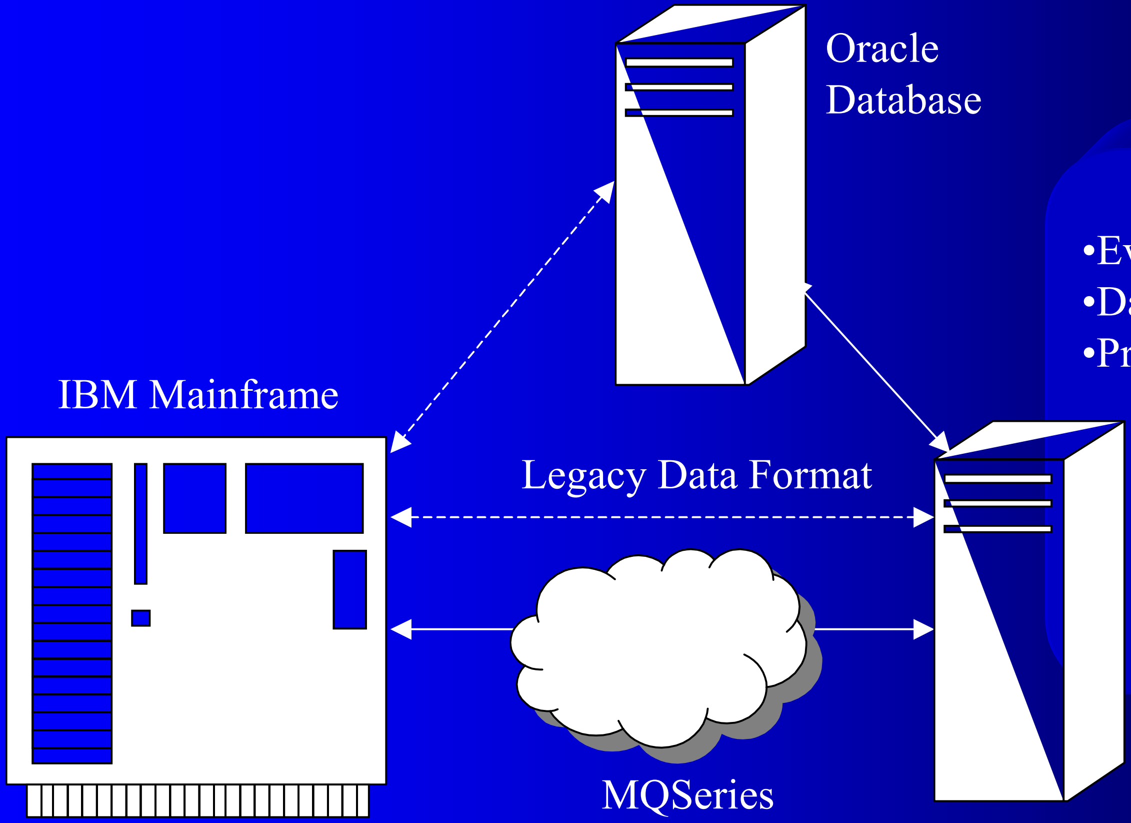
Goals

- Create a transparent flow of data through business processes
- The seamless integration of business processes for the purposes of conducting business electronically
- Identify patterns that impact business processes and compensate immediately

Step 1: Specify Business Objectives

- Step 1 requires the organization to identify specific business objectives
  - Lower the cost of printing
  - Increase throughput in production
- Analyze the flow of data relative to these objectives
  - e.g. track events related to generating a printed document

Step 2: Correct Data Flow



- Events Processing
- Data Transformation
- Process Flows

Handle Events

- Data enters the system in an asynchronous manner
- Multiple inputs
- Events bind inputs to processing
  - Associates behavior with data
  - Abstracts the processing from the data
- Events generate multiple new events
- Capture event invocation in audit log

Transform Data

- Web Information Portals
  - Identify
    - Capture metadata on need-to-use basis
    - Saves time and money
  - Extract
  - Bi-directional transforms
  - Update

Develop Process Flows

- A process flow is the implementation of a business operation
- Comprises multiple processes
- Initiated by an event
- Example:
  - Purchase generates billing event (invoice)
  - Invoice generates accounting event

Step 3: Analyze Event Log

- The events captured in Step 2 will identify the path that is being followed
- Step 3 requires human intervention to examine the logs and identify the events that are leading away from the business objective
  - e.g. Lead time for print jobs are less than a week

Step 4: Manage Events

- Based upon the analysis in Step 3, the company can now institute new events to change the process flow
  - e.g. Force lead times for jobs to be at least two weeks without executive signature
- Repeat Steps 2-4 until business objective is met

Declarative Vs. Procedural

- Declarative
  - Environment interprets for itself meaning
  - Facilitates openness and flexibility
  - Easier to change
- Procedural
  - Environment is told what it is being given
  - Requires agreements between sending and receiving parties
  - Broken agreements lead to exception handling

Establishing Published Formats for Delivery of Semantic Content

Introduction to XML

- Syntax and Semantics
- XML 1.0 Grammar
- Document Type Definitions
- Processing XML

Syntax and Semantics

Syntax

- Defines the grammar used to describe the physical representation of information
  - Syntax allows us to build parsers that automate the extraction of information from a formatted document
- Defines the tokens that make up a legitimate sentence in a grammar
  - Tokens are defined by an allowable sequence of other tokens or characters

Semantics

“The study of relationships between signs and symbols and what they represent” — Webster’s Dictionary

- Semantics assign meaning to the tokens defined by syntax

The Importance of Syntax and Semantics to EAI

- Syntax is used to define the format that will be used to exchange/share information between two applications
- Semantics defines the fields and values that will be captured within the data that is shared/exchanged.

XML 1.0 Grammar

Well-formed & Valid

- Well-formed
  - Follows the rules for creating an XML document as defined by the XML specification
- Valid
  - Is well-formed and also adheres to the rules defined by the document’s corresponding Document Type Definition (DTD)

Well-Formed XML

- Contains a single XML prolog followed by at least one element, known as the ‘root’ element

```
<?xml version=‘1.0’?>
<ROOTELEMENT>
</ROOTELEMENT>
```

XML Prolog

```
<?xml version=“1.0”?>
```

- Options
  - EncodingDecl – defines the language encoding

    ```
    <?xml version=“1.0” encoding=“UTF-16”?>
    ```

  - SDDecl – states that there are no markup declarations external to the document

    ```
    <?xml version=“1.0” standalone=‘yes’?>
    ```

Elements

- Two types
  - Empty
  - With Content
- Empty element

```
<ANELEMENT/>
```

- With Content

```
<ANELEMENT>Some Content</ANELEMENT>
```

Elements and Tags

- An element is defined by a start and end tag

```
Start Tag: <ANELEMENT>
content
End Tag: </ANELEMENT>
```

- Content is defined as zero or more of the following XML types:
  - Element, PCDATA, Reference, CDATA, PI, Comment

Tag Name Examples

<Document>This is an example of a well-formed XML document</Document>

<Document>This is an example of an erroneous XML Document</document>

Attributes

- Start tags can contain a set of attributes
- Attributes are used to provide metadata about the content an element contains
- Attributes use the syntax key=‘value’
- Example:

<StartTag attribute1=‘yes’ attribute2=“yes”/>

- Notice that we can use the single or double quote character for attributes.
  - This is consistent throughout the XML specification where ever quotes are used.

Attribute Example

```
<PurchaseOrder>
  <AmbiguousAmount>
    12.34
  </AmbiguousAmount>
  <ClearAmount currency=“US”>
    12.34
  </ClearAmount>
</PurchaseOrder>
```

Character Data

- Elements can contain character data between the start tag and end tag
- Two types:
  - Parsed character data (PCDATA)
    - All PCDATA is considered for overall document well-formedness
  - Raw character data (CDATA)
    - A specially marked section representing text that will be ignored when evaluating the document’s well-formedness and validity

Processing Instructions

- Processing instructions are not considered part of the document's character data, but must be passed onto the processing application

- Format:

```
<?TargetName any sequence of characters?>
```

- TargetName may not be XML in any combination of upper or lower case letters

Comments

- Comments may appear anywhere within the XML document
- They are not considered part of the document’s character data, but may be passed onto the the processing application
- Format:

```
<!-- Any sequence of characters -->
```

Document Type Definitions

Valid XML Document

- In addition to being well-formed, a document can also be valid
- We ensure validity by providing a DTD that the parser uses to ensure that the elements in the document follow a prescribed order

What Is A DTD?

- A list of the elements, attributes, and entities that will be found in a single XML document
- Describe the acceptable semantics of an XML document
- Define possible hierarchies
- Used to validate XML documents

Sample DTD

```
<!ELEMENT Policy  (PolicyHolder )>
<!ATTLIST Policy  number   CDATA   #REQUIRED
                  location ENTITY  #REQUIRED >
<!ELEMENT PolicyHolder  (Name , Phone , Address ,
  FAX? , Type )>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Phone (#PCDATA)>
<!ELEMENT FAX (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
```

Uses for DTDs

- Authoring
  - Allows tools to assist the author in creating a valid XML document
- Processing
  - Allows us to identify in advance of processing that the document matches a prescribed schema
- Definition
  - Provides a way to define the agreement for the XML schema that will be used

Internal Vs. External DTDs

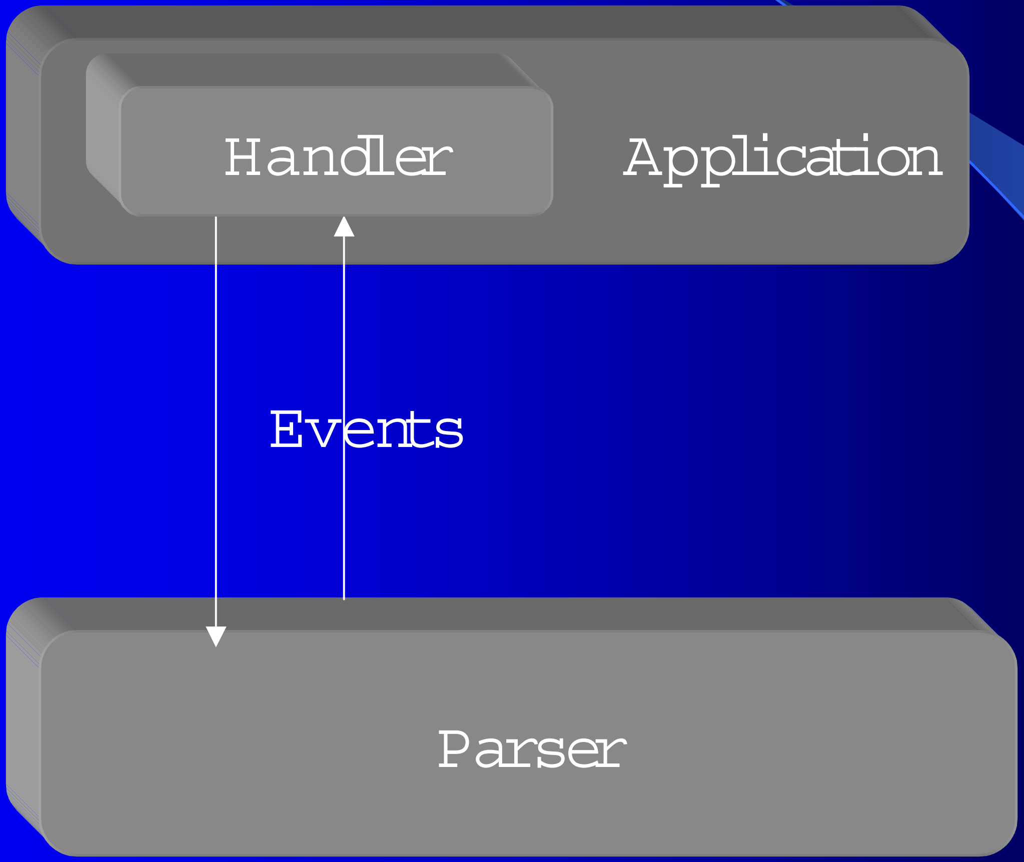
- XML Documents can declare their DTD inline or point to an external document that has the document type definition

Processing XML

What Is SAX?

- Simple API for XML
  - Developed by XML-Dev mailing list community
  - Earliest implementation by Microstar
- Event-based handling of XML Parsing
  - Framework for interfacing with a raw XML parser

SAX Architecture



Handler

Application

Events

Parser

SAX Interfaces For Parser Developers

- Parser developer implements these interfaces:
  - *AttributeList*
  - *Locator*
  - *Parser*
- these classes
  - *HandlerBase*
  - *InputSource*
- and these exceptions
  - *SAXException*
  - *SAXParseException*

SAX Interfaces For Application Developers

- Application developer implements these interfaces:
  - *DTDHandler*
  - *DocumentHandler*
  - *EntityResolver*
  - *ErrorHandler*

Parser Interface

- Constructors
  - *InputSource*
  - *URI*
- Handler connections
  - *setDocumentHandler*
  - *setDTDHandler*
  - *setEntityResolver*
  - *setErrorHandler*
  - *setLocale*

DocumentHandler Events

- Start Document
- End Document
- Start Tag
- End Tag
- Processing Instruction
- Characters
- White Space
- Locator

AttributeList

- Presents the interface for retrieving attributes of an element during the processing of a start element event
- Methods
  - getLength
  - getName
  - getType
  - getValue
    - Works over position or name

Handling Text

- Generates a *character* event
- Text is defined as the set of characters between a start tag and an end tag, but preceding another start tag
- Parser will pick up ‘n’ number of characters and deliver them to the DocumentHandler’s *characters* an array of type char

Handling Text

```
<TAG>text
  <TAG2>text</TAG2>
  text continued
</TAG>
```

- Minimum of three character events
  - text
  - text
  - text continued
- Note: only two elements
  - Processing requires state to be maintained

Processing Instructions

- <?xml?> is not a processing instruction but instead is identified as the XML Declaration
- It will not generate an PI event
- “XML” & “xml” are reserved target names

DTDHandler Interface

- Triggers events upon identification of declarations of type ENTITY, or ENTITIES within the DTD
- Application must store this information for use further on in the parsing

EntityResolver Interface

- Allows application to resolve external entities
- Single method *ResolveEntity*
- Application responds by supplying an InputSource back to the parser
- InputSource implements a character stream reader that will supply the resolved entity

ErrorHandler Interface

- SAX errors broken down into three categories
  - Recoverable
    - Application may continue to pass parsed character data after detection
  - Non-recoverable
    - Application must not continue to pass parsed character data to the application in the standard manner
  - Warnings

Locator

- Passed into the document handler on the *setLocator* event before ever any other document handling events are triggered
- Can be used by the application to find out the position within a XML document that triggered an event
- Not mandatory to implement

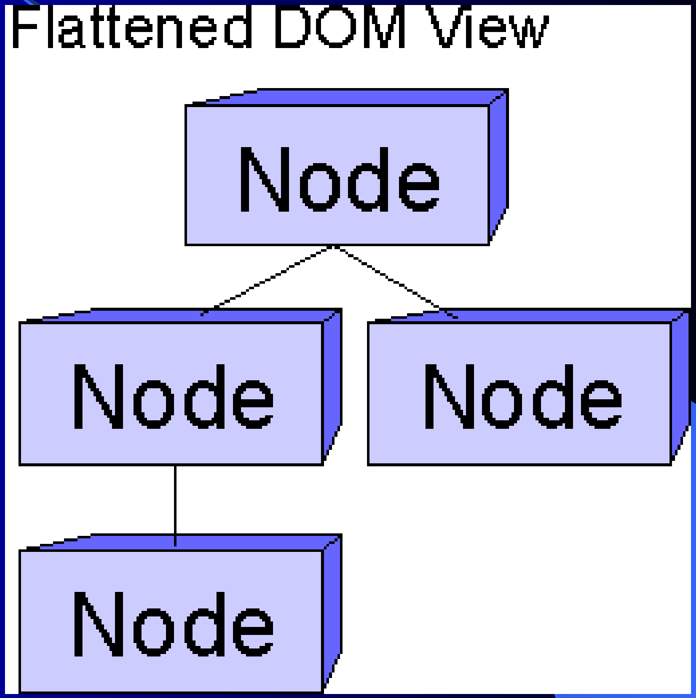
W3C Document Object Model (DOM)

W3C Document Object Model (DOM)

- Attempt to provide a standard programmatic interface to XML and HTML documents
- Provides two complementary views of the parse tree
  - Flat model
  - Object-Oriented Model

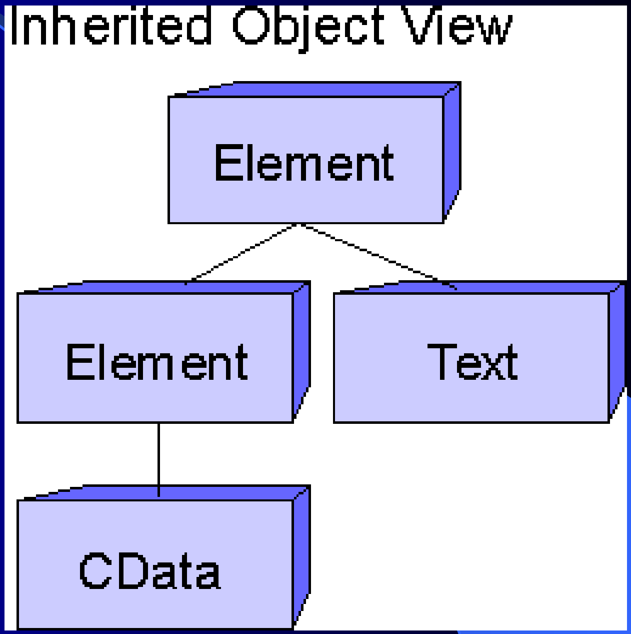
Flat View

- The flat view was developed to augment the Object-Oriented model (viewed as the default model) to support environments and programming languages that do not support data typing

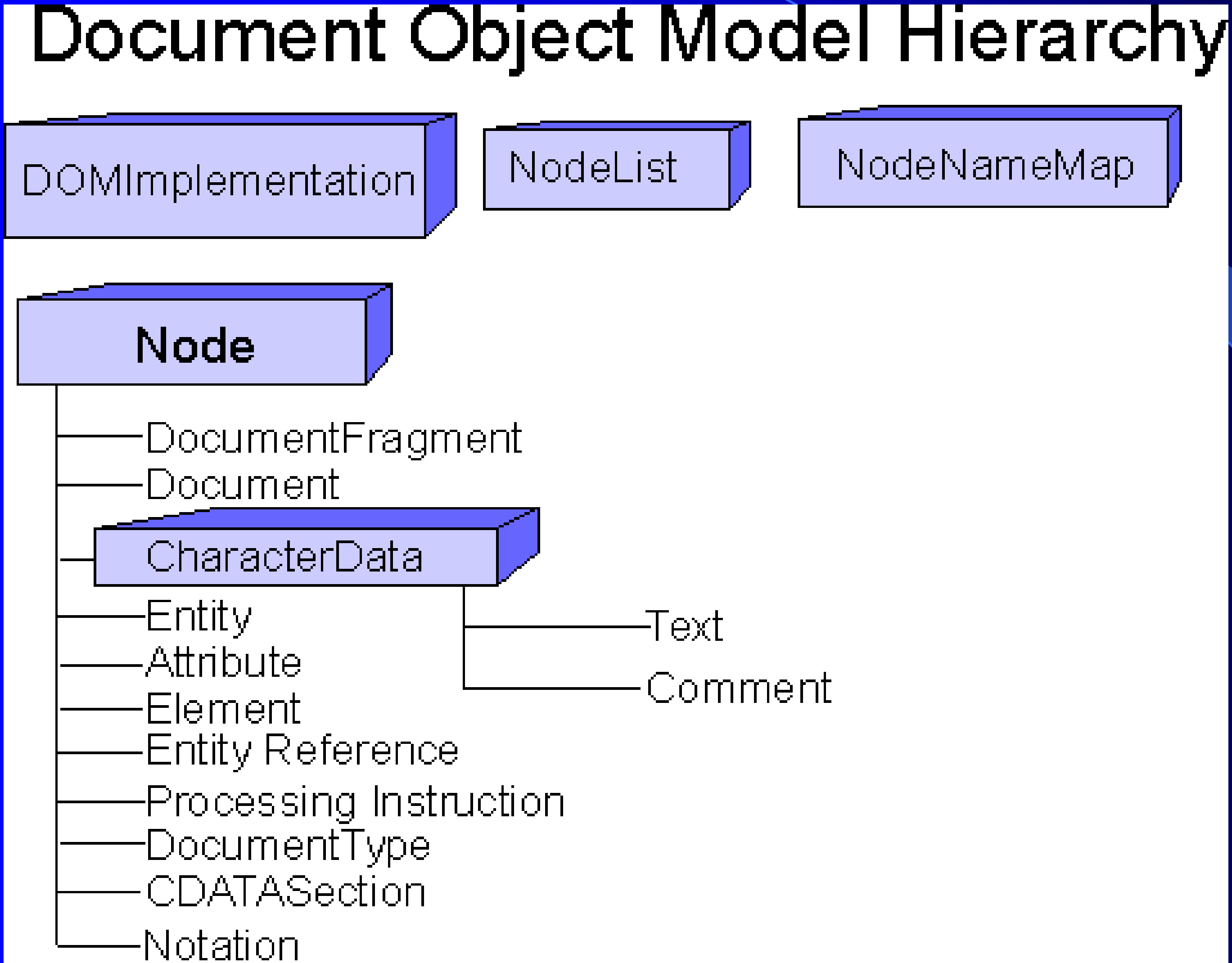


Object-Oriented View

- OO view provides a hierarchical typed view of CDATA, ELEMENTS, ENTITIES, COMMENTS, & TEXT within the containment of a DOCUMENTFRAGMENT



DOM Object Model



Document Object Model Hierarchy

DOMImplementation

NodeList

NodeNameMap

Node
- DocumentFragment
- Document
- CharacterData
  - Text
  - Comment
- Entity
- Attribute
- Element
- Entity Reference
- Processing Instruction
- DocumentType
- CDATASection
- Notation

DOMImplementation

- Allows developers to check to see if a particular feature was implemented
- Currently used to check for XML and HTML features and the versions supported

Document

- Virtual representation of the HTML or XML document
- Used to retrieve elements from the document
- Used to create new document components which can later be inserted into the document
- Holds the root element

DocumentFragment

- Lightweight implementation of the Document object
- Basically a reference holder for a sub-tree that will be inserted into a larger document
- Don’t require a root element
- Can have zero children

Node

- Represents monolithic flat view of all elements
- Also represents the base class for all document objects
- Allows direct manipulation of the underlying document
- *removeChild*, removes from tree, but object is still active

NodeLists

- NodeLists are ordered in a manner consistent with a preorder traversal of the parser tree
  - Root, Leftmost Branch to Right Branch

NamedNodeMap

- Collection of Node objects
- Access by name
- Does not maintain preorder traversal ordering
- Used by Node object to return set of Attributes

CharacterData

- Abstract base class for Text and Comment objects

Text

- Represents the characters after a start tag, but before a corresponding end tag, or another start tag
- *splitText* breaks the text node into two text nodes at a specified offset
  - Both text nodes continue to exist in the sub-tree
- Text is stored in objects named #text as a child of the element it is contained in

Element

- Manages its attributes directly instead of as children
- Interface looks awkward if not familiar with inheritance, functionality to add children and retrieve the list of attributes is actually in the Node object
- Combines adjacent Text nodes into a single Text node

Attribute

- Inherit functionality from Node, but are not considered to be part of the document
- Attributes are associated with the Elements that they are declared with
- Being of type Node allows implementations to more easily represent a combination of Text and EntityReferences

Comment

- Holds the text between:

  <!-- & -->

ProcessingInstruction

- Represents the target and associated data for a PI

CDATASection

- Unparsed text that is identified between:

  <![CDATA

  and

  ]]>

DocumentType

- Contains a list of all entities defined in a document
- Separates entities and notations into separate NamedNodeMap objects

Notation & Entity

- Representations of Notation or Entity declarations defined in the XML document

EntityReference

- A holder for aliased parsed text
- Allows for resolution on use, instead of at parse time

SAX or DOM

- When to use SAX
  - Going after a small-memory footprint
  - Read only what you need
  - Want to expose DOM or other API
- When to use DOM
  - Multiple queries against the document
  - To build a document in memory from scratch, or to change an existing one
  - Compare document sections

XML Stylesheet Language (XSL)

- Broken down into two components
  - Formatting objects
  - Transformation language (XSLT)
- XSLT
  - XML vocabulary for transforming one XML vocabulary into another
  - Works by establishing transformation rules for each type of element in the original XML document

XSLT (cont'd)

- Positives
  - Simplifies creation of new XML documents from existing XML
  - Provides some conditional processing
  - Can be used by anyone that knows XML
- Negatives
  - Path expressions used for extraction of elements from original document cannot account for subjective relationship
    - e.g. Administrative contact information available if parent element represents a Vice-President

XPointer

- An XML facility for extraction of elements from within an XML document by describing a unique path from the root element
- Described as a URL facilitating ease of using from within Internet applications

XML Path Language (XPath)

- Used to provide a standard path expression language for XPointer and XSLT

```
//Line[position()=1]/attribute::num
```

Matching Node: num/Matching Attribute Value: 1

```
//Line[position()=2]/attribute::num
```

Matching Node: num/Matching Attribute Value: 2

```
//Line[position()=last()]/attribute::num
```

Matching Node: num/Matching Attribute Value: 3

XML Query Language (XQL)

- Currently a newly engaged working group within the W3C
- Attempting to create a language for extraction and updating of XML documents through the use of relational algebraic expressions
- Will allow easier manipulation of complex relationships within an XML document

XML Application Integration (XAI)

XAI Requirements

- Meta-models
- The Integration Document

Meta-models

Meta-Models

- Meta-models are models that used to define other models
- They define types of information that would be found in the model instead of instances of the information itself
- Example:
  - A meta-model for e-commerce may include:
    - Price, currency, consumer, producer, destination, account, etc.
  - From these types a model of a purchase order may be devised

XML and Meta-models

- The semantic representation of any information is best served by being represented in XML using a meta-model
- Example:
  - Instead of defining a purchase order vocabulary, which would be extremely rigid, use constructs defined in the e-commerce meta-model

Benefits of Using Meta-models

- The abstract definitions are defined outside of the scope of a single document. This leads to the following result:
  - Changes in logic to process the model will apply to multiple document types simultaneously

Problems of Using Meta-models

- Requires significant investment up-front to study emerging patterns from existing data
- Is an iterative design process that requires significant investment in time and money resources

Final Note on Meta-models

- A single one-off application integration does not require a meta-model since the benefits of applying to multiple documents does not exist
- A multi-document, multi-application integration without a meta-model will cost significantly more to develop than one with a meta-model

The Integration Document

Integrating Applications using XML

- XAI stipulates that applications are exchanging data—not sharing data!
- It is not a requirement for standardization of vocabularies in order to exchange data
  - It does, however, introduce the need for multiple levels of translation
  - The impact of this mainly affects performance

XAI @ Work

- RosettaNet
  - Designed to permit the electronics manufacturers to directly communicate with their customers
  - Defines process flows, messaging protocols, document structures, and document entities
- BizTalk
  - Designed to provide an XML-based messaging infrastructure
  - Defines messaging protocol

Establishing Well-Defined Pathways for Delivering Data To Other Systems

Message-based Integration

- Message-Oriented Middleware (MOM)
- Message Brokering
- Workflow

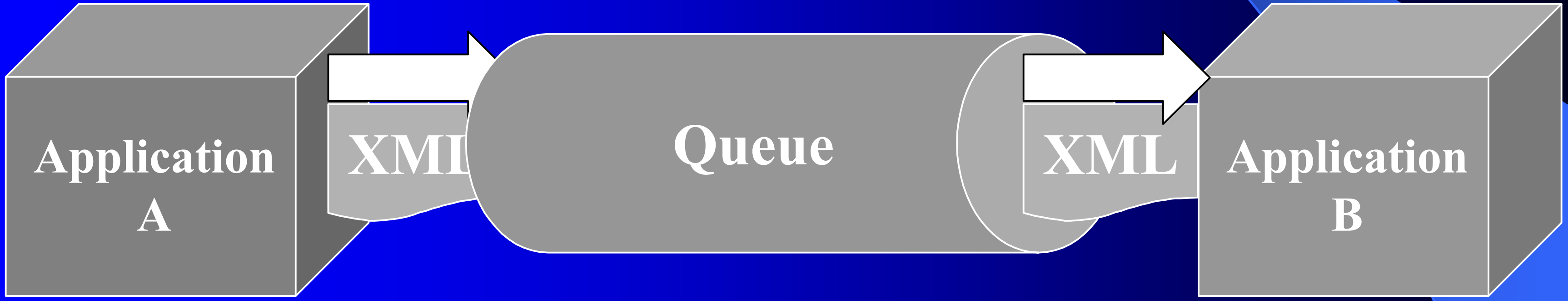
MOM

- Message-Oriented Middleware provides an infrastructure for the guaranteed delivery of data in an asynchronous manner
- Two primary methods
  - Queuing
  - Publish & Subscribe

Queuing

- Queuing typically provides a single pipe between two applications
  - The sending application puts data into the pipe
  - The receiving application pulls data out of the pipe
  - These two operations do not happen synchronously
- Can provide strong security since receiver must first authenticate themselves to the queue

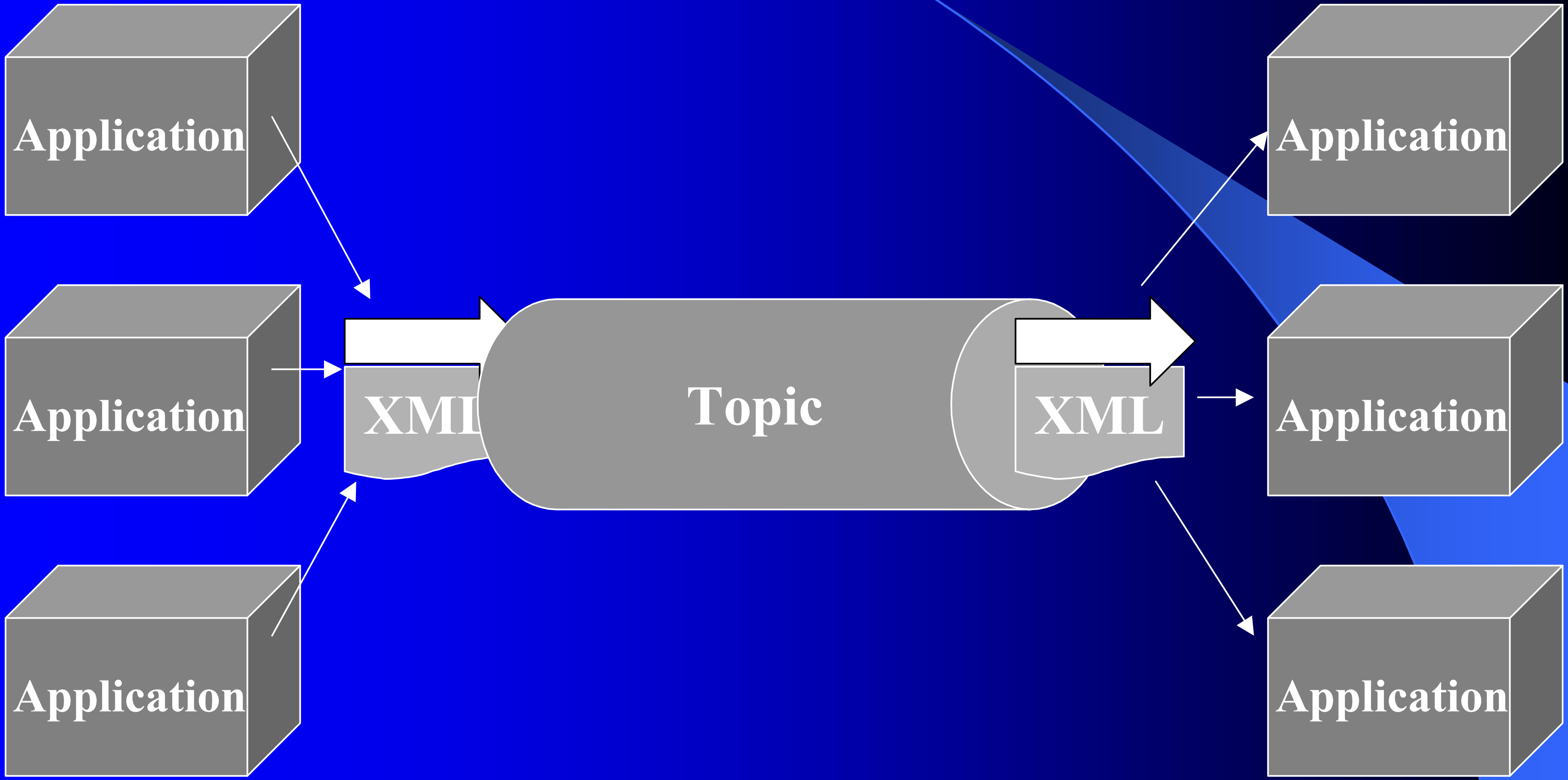
Queuing Example



Publish & Subscribe

- Pub/Sub abstracts the sender from the receiver over a topic
  - Receiver registers with middleware to receive updates over a particular topic
  - Sender issues message over a topic or set of topics
- Relationship over a topic can be many to many
  - Queues are typically 1:1
- More cumbersome to enforce security in a pub/sub scenario

Pub/Sub Example



Message Brokering

- Message brokering is a fancy term for document dispatching
- MBs examine some portion of the document's content or envelope and selects a receiver dynamically
  - Receivers may be a process or person
- MBs require a knowledge-engineer to create a map from the incoming document stream to a particular document type
  - Requires semantic and syntactic analysis of data stream

Workflow

- Workflow is the highest level component in a MOM solution
- Workflow uses MOM and MBs to provide a foundation for transmission and analysis of documents
- Workflow allows users to define and manage routing of documents based upon state and content

MOM and EAI

- MOM-based solutions have been most popular for EAI to date
- Reasons:
  - EAI traditionally has been an intranet operation allowing selection of a single messaging platform
  - EAI usually requires communication between systems across a departmental or divisional boundary

MOM and B2B

- Business-to-Business application integration will rely on Web-based protocols for messaging
- Reasons:
  - MOM requires applications to conform to a particular application programming interface
  - MOM implementations are not consistent and and ubiquitous

Defining Interfaces Between Applications

Interface Is An Overloaded Term

- There are many ways to define an interface between two applications
- The term “interface” does not represent a single instance of a type of interface, but an entire class

What is an “interface”?

An interface is a clearly-defined method of moving data into or extracting data from another system

3 Key Integration Interface Classes

- Tables
  - DBMS
- Application Programming Interfaces
  - Direct API
  - XML-RPC
  - Java RMI, COM, CORBA
- Documents
  - Comma-delimited, XML, all the rest…

Establish the Rules of Engagement

Rules-based Environments And EAI

- Rules-based environments provide the following benefits for EAI:
  - Flexible
  - Dynamically configurable
  - Responsive to multiple input types

The Components of A Rules-based Environment

- Integration Engines
- Process Modeling
- Metadata Brokering

Integration Engines

- Integration engine responsibilities:
  - Data format transformation
  - Execution of process model
  - Aggregate data from multiple sources
    - API Integration
    - Document Integration
    - Communications

Process Modeling Runtimes

- Responsible for:
  - Creating a new process flows by connecting together a group of smaller process flows with simple logic routing
  - Providing a facility for associating names with process flows
  - Allow external applications to execute defined flows

Metadata Brokering

- The process flows and the transformation maps combine to create the metadata for a single integration
- Metadata brokering allows alternate metadata sources to be matched up with the existing process flows and transformation maps
  - Lowers costs of integration
  - Allows other data sources to provide input to existing systems

Summary, Q&A