John Hosie– Product architect, IBM Integration – jhosie@uk.ibm.com

IBM Integration Bus

Transformation Options

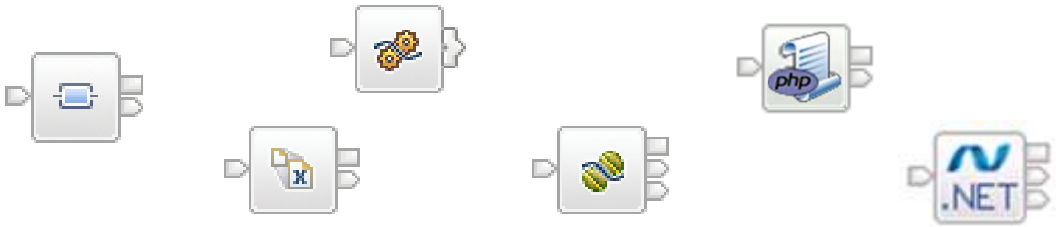
IBM Integration Bus
Version 9

Important Disclaimer

- THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.
- WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.
- IN ADDITION, THIS INFORMATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.
- IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.
- NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:
  - CREATING ANY WARRANTY OR REPRESENTATION FROM IBM (OR ITS AFFILIATES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS); OR
  - ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.

Objectives

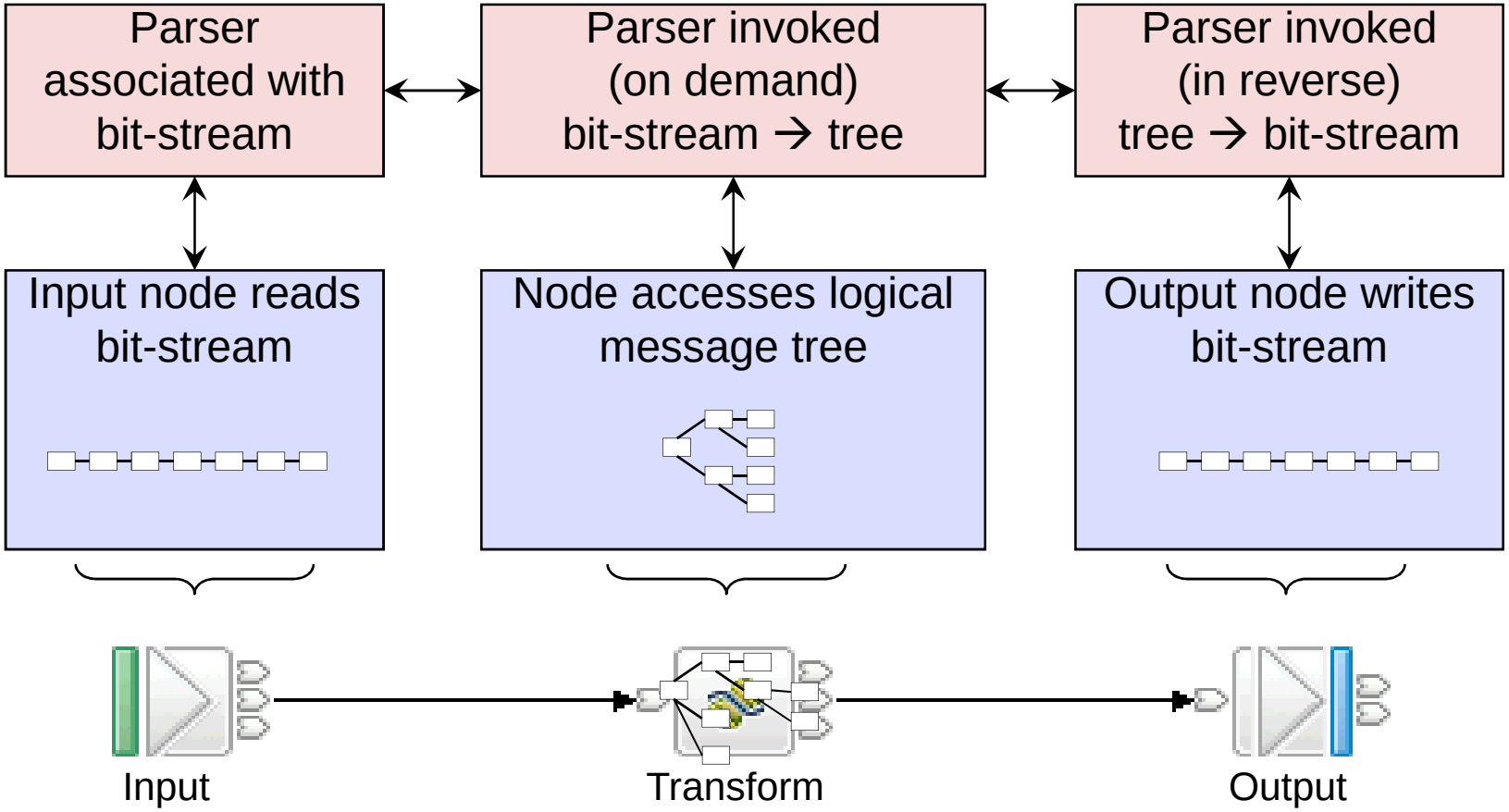
- **Discuss the major transformation technologies available in WebSphere Message Broker**
  - Mapping
  - XSLT
  - ESQL
  - Java
  - PHP
  - .NET



- **Give an overview of each technology and how to use them in the broker**
- **Highlight the key strengths and weaknesses of each technology**
- **We are going to keep a score card to assess the relative merits of each technology!**
  – Performance and scalability
  – Backend integration
  – Skill sets and learning curve
  – Developer usability
  – Portability and maintenance

- **There are no right answers in all situations for the choice of transformation!**

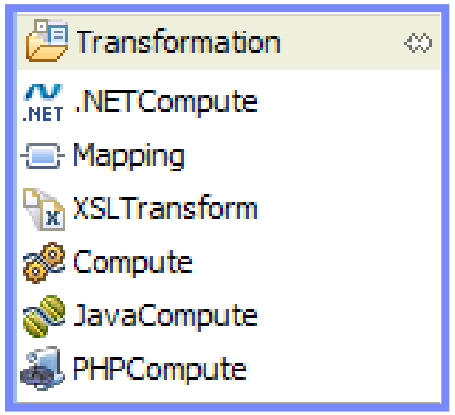
Broker Overview – Flows, Nodes and the Message Tree

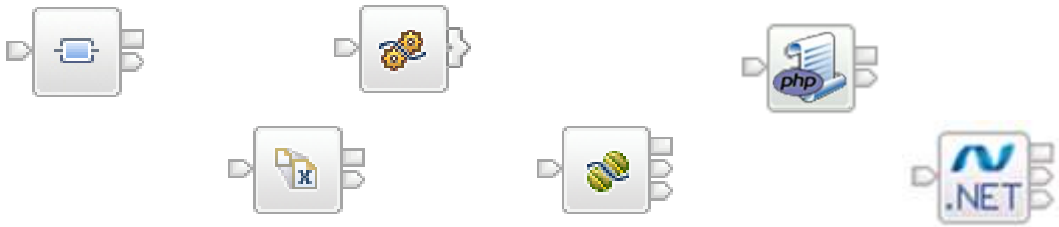


Why So Many Options?

- **Because that is what our customers asked for**
  - Desire to re-use assets
  - User skill set and background
- **Different tasks require different tools**
- **"But which should I choose?"**
- **"To a man with a hammer, everything looks like a nail."**

- Mark Twain





Mapping Node

Introduction

- **Message flows are a kind of 'graphical programming'**
  - Wires define message routing
- **Maps extend this to message transformation**
  - Lines drawn between structural elements of the message
- **Simple to use – drag and drop transformation**
- **Highly configurable using XPath 2.0**
  - Versatile expression language
  - Large function library
- **Ability to call user-defined functions/methods**
  - ESQL and Java (static methods)
- **Can map transport headers and** `LocalEnvironment`
  - Not **`Environment`** or **`ExceptionList`**
- **Can generate multiple output messages**
  - Split a document into different structures
  - Shred a large document with repeating elements
- **Database support**
  - Select, Insert, Update, Delete
  - Stored procedures

The Mapping Editor

Invoice > PurchaseList > Statement

Invoice

Invoice		<Anonymous>
<Click to filter...>		
Initial	[1..*]	string
Surname	[1..1]	string
Item	[1..*]	<Anonymous>
Balance	[1..1]	decimal
Currency	[1..1]	string

Assign
Assign
fn:string-join
Move
Move
For each
Custom XPath

Statement		<Anonymous>
<Click to filter...>		
Style	[0..1]	string
Type	[0..1]	string
Customer	[1..1]	<Anonymous>
Initials	[1..1]	string
Name	[1..1]	string
Balance2	[1..1]	string
Purchases	[1..1]	<Anonymous>
Article	[1..*]	<Anonymous>
Amount	[1..1]	AmountType
Currency	[0..1]	string
value	[0..1]	decimal

Properties | Problems | Deployment Log | Progress | Console | History | Plug-in Registry

Transform - string-join

General
Cardinality
Variables
Condition
Sort
Order
Documentation

Takes an input sequence of strings and a separator and returns a string created by concatenating the members of the input sequence using the separator as a delimiter. See XPath 2.0 Specification...

Parameters:

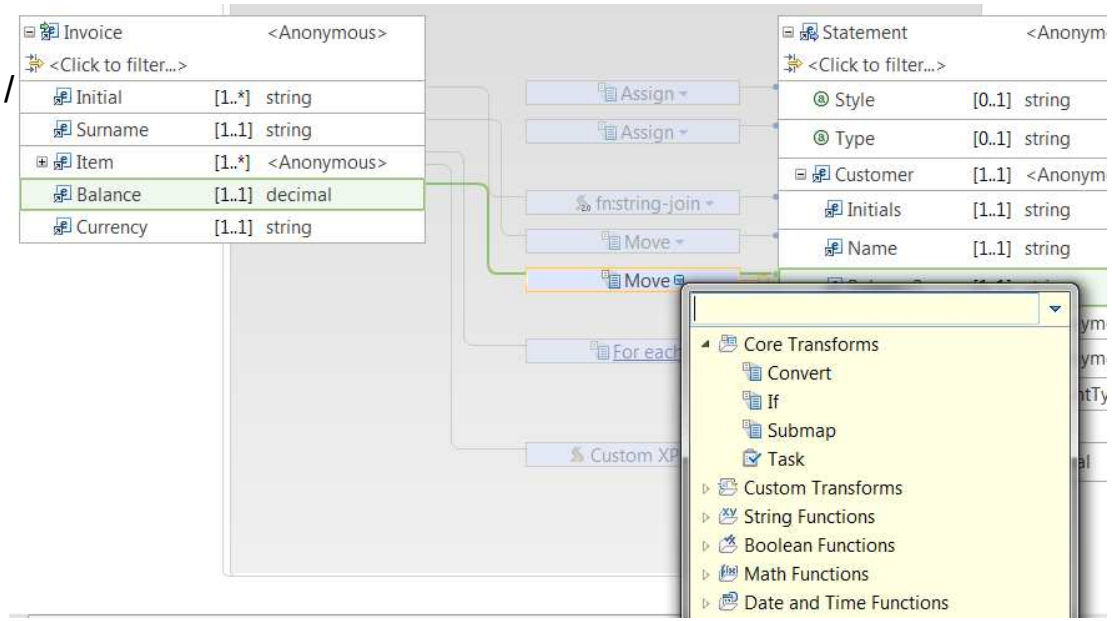
Name	Type	Value
strings	xs:string[ ]	$Initial
separator	xs:string	''

Add
Edit...
Remove

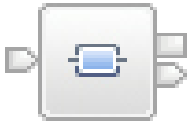
Creating Mappings

- **Drag and drop source to target (can also drag target to source)**
- **Choose a transform (how to map)**
  - Defaults to most commonly used transform appropriate to the source and target
- **Hierarchical mapping editor**
  - Create ‘nested’ maps to break a complex transformation into smaller units
  - ‘Structural’ transforms – ‘Local’, ‘For each’, ‘Join’, ‘Append’, ‘Submap’, ‘If / Else’
  - Drill down to create the mappings between the children of these elements
  - ‘Basic’ transforms – ‘Move’, ‘Assign’, ‘Convert’, ‘XPath’

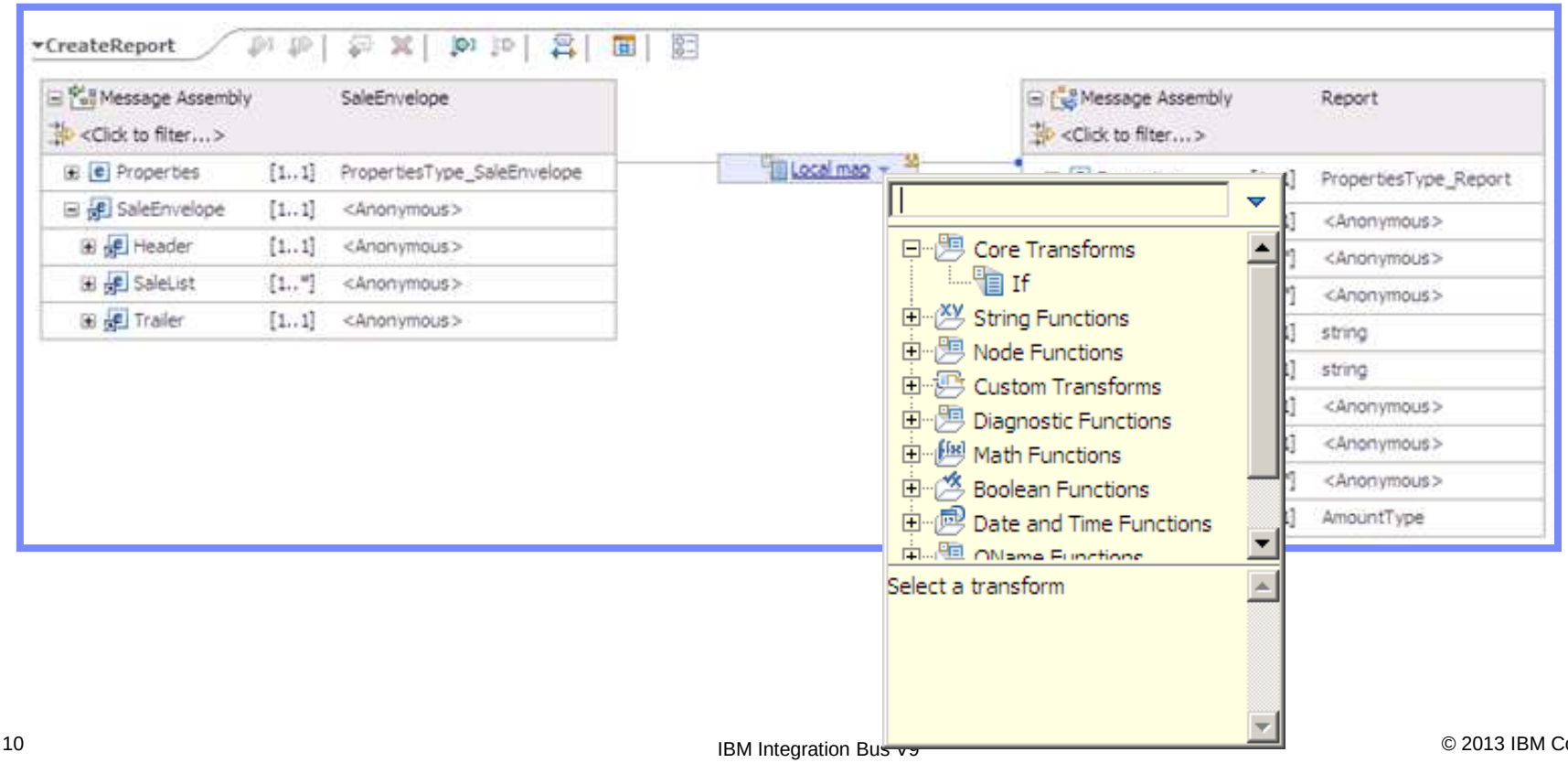
- **Configure the transform**
  - In the ‘Properties’ view
  - E.g. condition predicates, array filtering / ordering, etc.
  - Expressed in XPath
  - Content assist available



Function Transforms

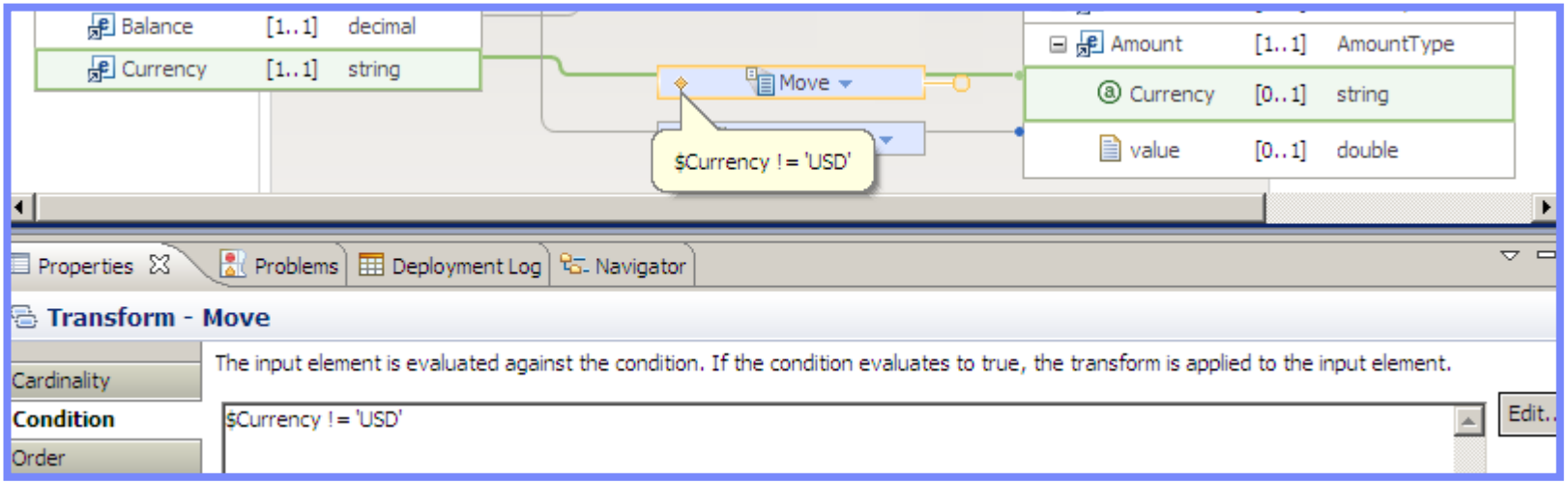


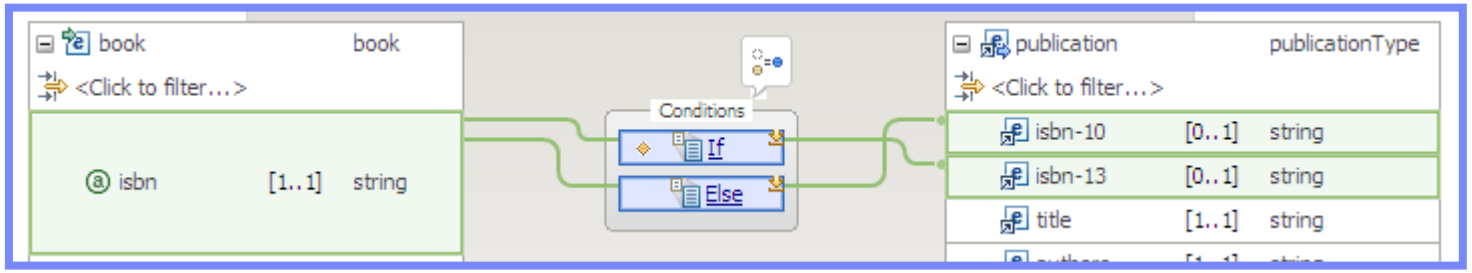
- **Target value can be computed by applying a function to one or more inputs**
  - Large function library inherited from XPath 2.0
  - String manipulation: concatenation, sub-string, matching, find/replace, regex
  - Numeric calculation: counting, summing, rounding, min/max
  - Date/time processing: creating time stamps, extracting components of dates and times



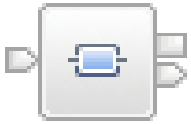
Conditional Mapping

- **Individual transforms can be configured to occur only if a condition is met**
  - User enters an XPath 2.0 predicate
  - Mapping is only performed if predicate evaluates to true
- **Can also create an `if else` condition for higher level control**
  - Each clause contains a nested map

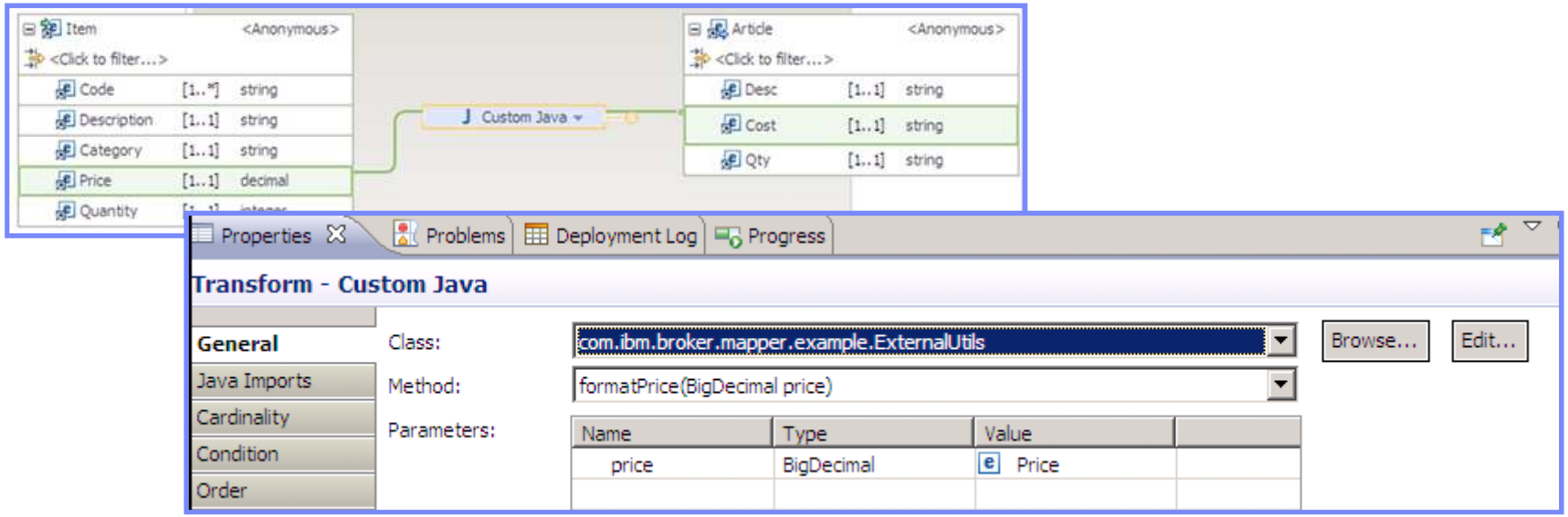




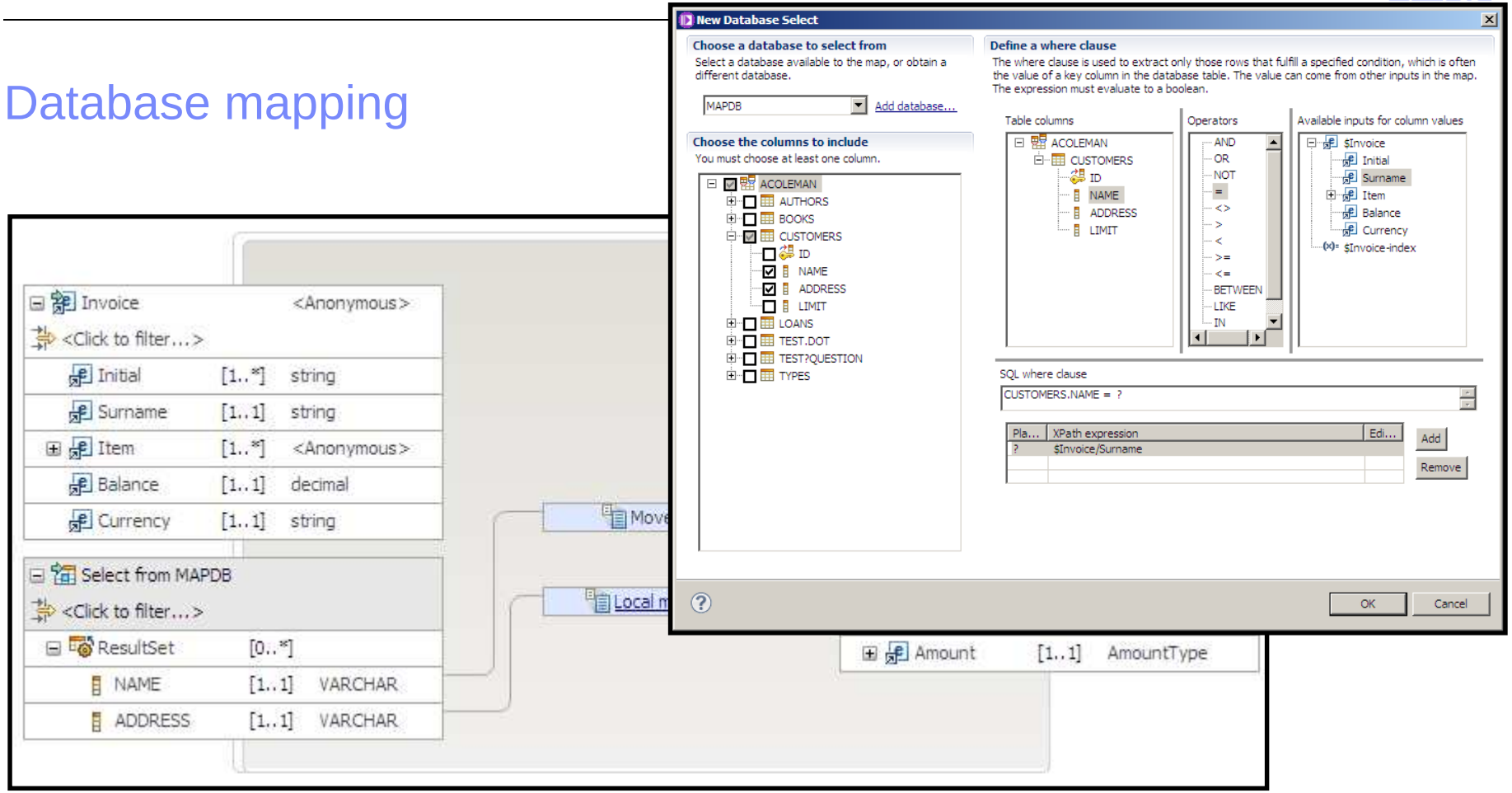
Custom Transformation Logic – XPath, ESQL and Java



- **Complex data manipulation logic can be written in XPath**
  - Use when complexity goes beyond built in transform types and function library
  - E.g. `1.6 * sum($Item/(Price * Quantity))`
- **Can also call out to user-defined functions written in Java and ESQL**
  - Using APIs familiar to existing broker users
  - User-defined functions extend the built-in library for use in XPath expressions



Database mapping



- **Database SELECT, INSERT, UPDATE & DELETE supported in mapper**
  - WHERE clause built in drag-and-drop editor
  - Result set appears as extra input tree in map editor
  - Insert/Update/Delete appears as a target
- **Stored procedure supported**
  - Map fields from input message to IN or INOUT parameters in stored procedure
  - Map OUT and INOUT parameters to fields in output message

Mapper Score Card

Performance and scalability	✓	**Maps are directly deployed as MSL files.** **High performing transformation engine executes map on runtime.**
Backend integration	?	**Mapper integrates with databases and Java code**
Skill sets and learning curve	✓	**Small learning curve and requires minimal developer skills.** **Standards based XPath 2.0 expression language.**
Developer usability	✓	**Simple to use, drag and drop. But must have schemas.**
Portability and maintenance	✓	**Visual metaphor makes maintenance relatively easy** **Tricky to merge changes in source code control**

XML Transformation Node

Introduction

- **W3C standard XSLT 1.0 transformation language**
- **XSLT is a functional programming language**
  - Functions cannot modify state – variables are not variable!
  - Can only transform XML documents
- **Uses XPath 1.0 to navigate the incoming XML document**
- **Functions are expressed as template rules**
  - Templates are ‘applied’ to elements of the XML document
  - Output document is defined within these templates
- **Ability to dynamically select style sheet from LocalEnvironment**
  - Cannot map transport headers or environment trees
- **No database support nor external functions**
- **XSLT debugger available in Message Broker Toolkit**
  - Not integrated with Flow Debugger

XPath 1.0 Overview

- **XPath is a language for addressing parts of a document**
- **Operates on the logical tree**
- **Location Paths**
  - Used to navigate the tree
  - Series of location steps separated by ‘/’
  - Each step selects a set of nodes relative to the previous step
- **Expressions**
  - Function library
  - Numeric and boolean operators
  - Variable references
- **W3C recommendation http://www.w3.org/TR/xpath**
- **Used by other Message Broker nodes, not just XSLT**

XPath Example

```
//book[@price < 10]/title
```

```
<bookshop>
  <stock>
    <book price=‘49.99’>
      <title>Learn WebSphere in 24 hours</title>
    </book>
    <book price=‘9.99’>
      <title>Unix in a Nutshell</title>
    </book>
    <book price=‘4.50’>
      <title>XPath quick reference</title>
    </book>
  </stock>
</bookshop>
```

XPath Example

```
//book[@price < 10]/title
```

```
<bookshop>
  <stock>
    <book price='49.99'>
      <title>Learn WebSphere in 24 hours</title>
    </book>
    <book price='9.99'>
      <title>Unix in a Nutshell</title>
    </book>
    <book price='4.50'>
      <title>XPath quick reference</title>
    </book>
  </stock>
</bookshop>
```

XPath Example

```
//book[@price < 10]/title
```

```
<bookshop>
  <stock>
    <book price='49.99'>
      <title>Learn WebSphere in 24 hours</title>
    </book>
    <book price='9.99'>
      <title>Unix in a Nutshell</title>
    </book>
    <book price='4.50'>
      <title>XPath quick reference</title>
    </book>
  </stock>
</bookshop>
```

XPath Example

```
//book[@price < 10]/title
```

```
<bookshop>
  <stock>
    <book price='49.99'>
      <title>Learn WebSphere in 24 hours</title>
    </book>
    <book price='9.99'>
      <title>Unix in a Nutshell</title>
    </book>
    <book price='4.50'>
      <title>XPath quick reference</title>
    </book>
  </stock>
</bookshop>
```

XSLT Template Rules

- **Each template rule defines part of the transformation**
  - ‘Pattern’ matched against the source tree (XPath)
  - ‘Template’ instantiated to form part of the result tree
- **Each template can instantiate other templates:**

```
<xsl:template match="/">
  <book-authors>
    <xsl:apply-templates select="/library/books/book"/>
  </book-authors>
</xsl:template>

<xsl:template match="book">
  <book title="{title}">
    <xsl:copy-of select="author"/>
  </book>
</xsl:template>
```

XSLT Score Card

Performance and scalability	?	**Easy to write inefficient style sheets by using poor XPath expressions (for example, using the // expression)** **The XSLT engine works with XML documents not trees!**
Backend integration	?	**No ability to invoke user-defined functions**
Skill sets and learning curve	✓	**Modest learning curve and requires minimal developer skills** **Standards based so there is a lot of material available**
Developer usability	✓	**Lots of excellent third party tools available**
Portability and maintenance	?	**Maintenance can be difficult for large transformations** **Portability of style sheets extends to third party applications** **Tricky to merge changes in source code control**

Compute Node

Introduction

- **Excellent for database interaction**
- **Syntactically similar to SQL**
- **Invoke static Java methods and database stored procedures**
- **Supports declarative and procedural programming styles**
- **Powerful SELECT statement can be applied to messages as well as database tables (or a mix of the two at the same time – can even be nested!)**
- **Access to all message domains**
- **Can address all message headers and environment trees**
- **Toolkit support with editor for syntax highlighting and context assist**
- **ESQL debugger integrated with Flow Debugger**

ESQL: SQL + Procedural Extensions

- **Typed user defined variables and constants**

```
DECLARE var1 CHARACTER 'Hello World';
DECLARE var1 CONSTANT CHAR 'Hello World';
```

  – If not initialized they are initialized to NULL for you
- **Data types**

```
CHARACTER DECIMAL FLOAT INT BIT BLOB BOOLEAN
```

- **Data and time**

```
DATE TIME TIMESTAMP INTERVAL GMTTIME GMTTIMESTAMP
```

- **Operators**
  – For manipulation and comparison of variables, etc

```
BETWEEN IN LIKE IS (NOT)
```

- **Conditional constructs**

```
IF, ELSEIF, ELSE, CASE, WHEN
```

- **Several looping constructs**

```
WHILE, REPEAT, LOOP, FOR
```

- **Functions**
  – Over 80 built-in functions

```
SUBSTRING LENGTH UPPER CONTAINS STARTSWITH RAND ROUND CEILING FLOOR
```

Functions and Procedures

- `Main()` – **the entry point for the Compute node**

```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
  ...
    SET OutputRoot.XMLNSC.Money.Amount = twice(myInt);
    CALL multiplyBy2(myInt);
  ...
  RETURN TRUE; --causes message propagation
END;
```

- **User-defined functions**

```
CREATE FUNCTION twice(IN p INTEGER) RETURNS INTEGER
  BEGIN RETURN p * 2; END;

CREATE PROCEDURE multiplyBy2(INOUT p INTEGER)
   BEGIN SET p = p * 2; END;
```

Working with Messages – Path Extensions

- **Field References**
  - Path syntax to address the tree elements
  - Starts with ‘correlation name’ to identify root of tree

```
SET OutputRoot = InputRoot; -- copies the whole message

SET OutputRoot.MQMD = NULL; -- removes the MQMD header

SET OutputRoot.XML.doc.title =    -- will generate the output
      InputBody.session[4].title; -- tree if it doesn’t exist


SET OutputRoot.XML.Library.Publication[] =
      InputBody.library.books.book[]    -- copies all elements
                                           (deep copy)
```

Transformation using SELECT

```
SET OutputRoot.XML.Library.Publication[] =
          SELECT BOOK.title    AS BookTitle,
                 BOOK.author[] AS Authors.Name[],
                 BOOK.isbn     AS ISBN,
                 BOOK.price    AS Price
          FROM InputBody.library.books.book[] AS BOOK;
```

- **The SELECT implicitly loops over the repeating 'book' element in the input message**
  - A 'Publication' element is created in the output for each one
- **The children elements of 'book' are mapped**
  - Element names are changed ('title' -> 'BookTitle', etc)
  - Values are copied (deep copy)
  - Arrays and structures are built (author[] -> Authors.Name[])
  - Note that nested repeating structures can be transformed with nested SELECTS

Mixing Declarative and Functional styles

```
SET OutputRoot.XML.Library.Publication[] =
          SELECT BOOK.title           AS BookTitle,
                 BOOK.author[]        AS Authors.Name[],
                 ToIsbn13(BOOK.isbn)  AS ISBN13,
                 BOOK.price * 1.6     AS Price
          FROM InputBody.library.books.book[] AS BOOK;
```

```
CREATE FUNCTION ToIsbn13(IN oldIsbn CHAR) RETURNS CHAR
  BEGIN
      IF(LENGTH(oldIsbn) = 10) THEN
          RETURN '978' || oldIsbn;
      ELSE
          RETURN oldIsbn;
      END IF;
  END;
```

Database Access

- `SELECT` **statement**
  - Creates an entire message tree from a database query

```
SET OutputRoot.XMLNSC.Response.Services.Service[] =
  (SELECT P.SVCCODE AS Code, P.SVCDESC AS Description
    FROM Database.SERVICES as P);
```

- `INSERT` **statement**
  - Allows you to add a row to a database table

```
INSERT INTO Database.Prices(ITEM, ITEMPRICE)
  VALUES (Body.Msg.Item, Body.Msg.ItemPrice);
```

- `UPDATE` **statement**
  - Changes one or more existing rows in a database table

```
UPDATE Database.Prices AS P
  SET ITEMPRICE = Body.Msg.ItemPrice
  WHERE P.ITEM = Body.Msg.Item;
```

- `DELETE` **statement**
  - Removes one or more existing rows in a database table

```
DELETE FROM Database.{DSN}.{Schema}.Prices AS P
  WHERE P.ITEM = Body.Msg.Item;
```

ESQL Score Card

Performance and scalability	✓	**Excellent performance - ESQL language run time is tightly coupled with message tree structures**
Backend integration	?	**Integrates with databases, Java and .NET code**
Skill sets and learning curve	?	**Good - many similarities to the SQL language** **Language extensions make it domain specific**
Developer usability	✓	**Excellent toolkit integration for edit and debug** **Straightforward to build re-usable code libraries**
Portability and maintenance	✓	**Merge utilities work as expected for source code control** **ESQL is deployed as source to Broker runtime**

Java Compute Node

Introduction

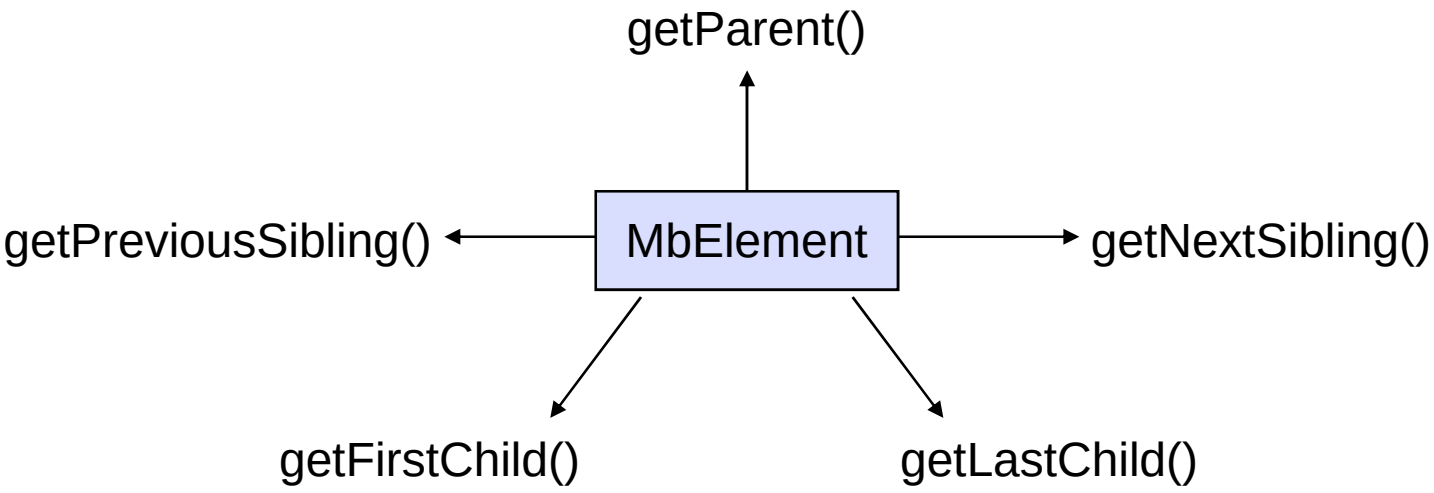
- **General purpose programmable node**
- **Java 7**
- **API to work with messages and interact with broker**
- **JAXB support for creating portable transformation logic**
- **Full XPath 1.0 support for message navigation**
- **Two general purpose output terminals for message routing**
- **Access to external resources**
  - For example file system, SMS, POJOs
- **JDBC (XA) Database support**
- **Supports all message domains, headers and environment messages**
- **Full IDE support of Eclipse Java Development Tools (JDT)**
  - Java debugger integrated with Flow Debugger

Getting Started

- **Wizard driven**
  - Skeleton Java code is created
- **User codes logic in `evaluate()` method**
  - Called by the broker once for each message processed
- **The incoming message is passed into `evaluate()` as part of a message assembly**
  - MbMessageAssembly encapsulates four MbMessage objects
    - Message - the incoming message
    - Local environment - local scratch pad work area
    - Global environment - global scratch pad work area
    - Exception list - the current exception list
- **Message assembly is propagated to an output terminal**

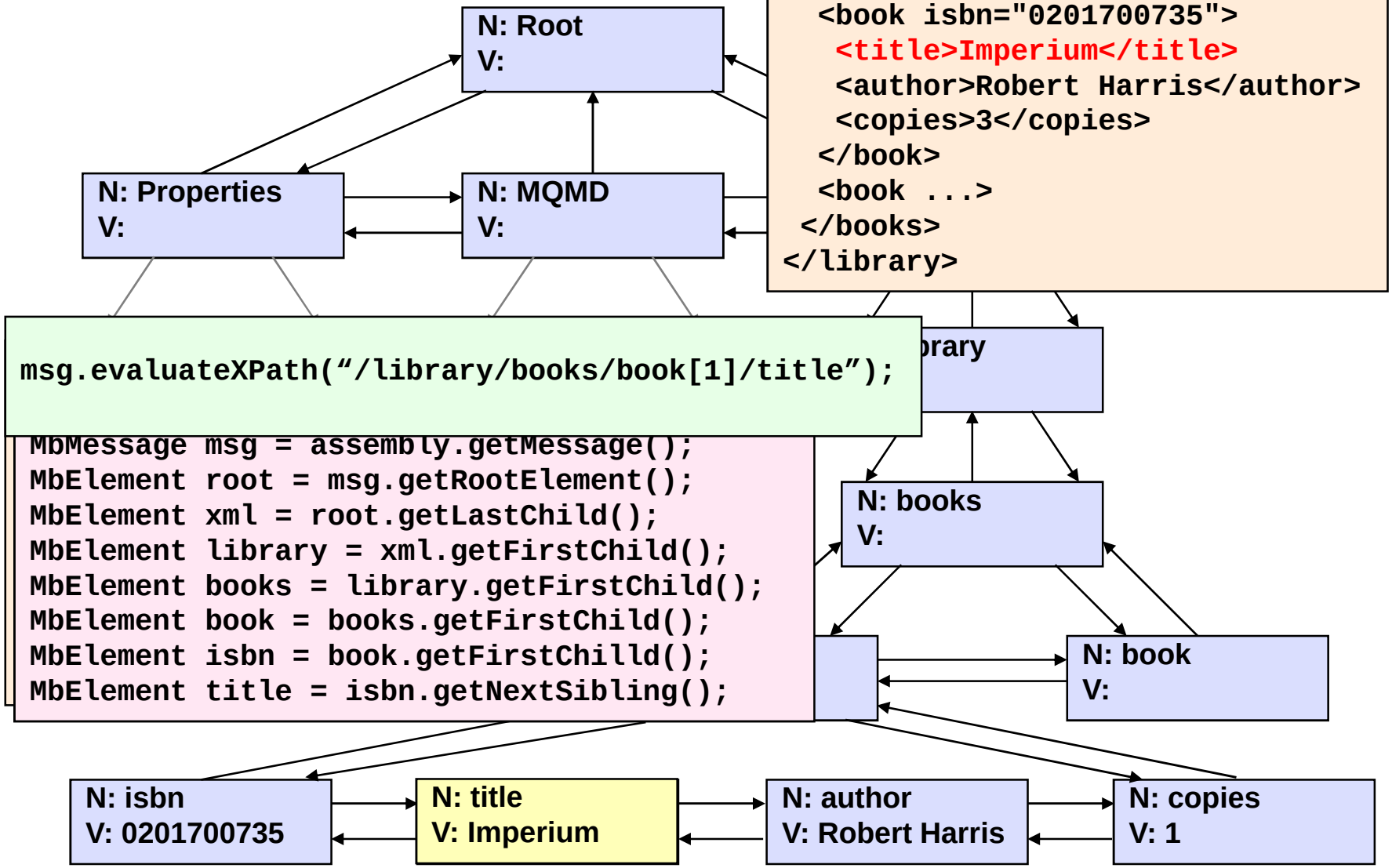
Navigating the Message Tree

- **Each element in the tree is represented by an** `MbElement` **object**
- `MbMessage.getRootElement()` **returns the root of the tree**
- **Methods to access an element –** `getType(), getValue()` **and** `getName()`
- `MbElement` **contains methods for traversing the message tree**



Navigating the Message Tree

```
<library>
 <books>
  <book isbn="0201700735">
   <title>Imperium</title>
   <author>Robert Harris</author>
   <copies>3</copies>
  </book>
  <book ...>
 </books>
</library>
```




```
msg.evaluateXPath("/library/books/book[1]/title");
```

```
MbMessage msg = assembly.getMessage();
MbElement root = msg.getRootElement();
MbElement xml = root.getLastChild();
MbElement library = xml.getFirstChild();
MbElement books = library.getFirstChild();
MbElement book = books.getFirstChild();
MbElement isbn = book.getFirstChilld();
MbElement title = isbn.getNextSibling();
```

Modifying a Message

- **The message passed to the node is read-only**
- **Must take a copy of it to modify**

```
MbMessage outMessage = new MbMessage(inMessage);
```

- **Methods for setting name and value of** `MbElement` **object**

```
setName(), setValue()
```

- **Methods for creating new** `MbElement` **instances in the tree**

```
createElementAsFirstChild()
createElementAsLastChild()
createElementBefore()
createElementAfter()
```

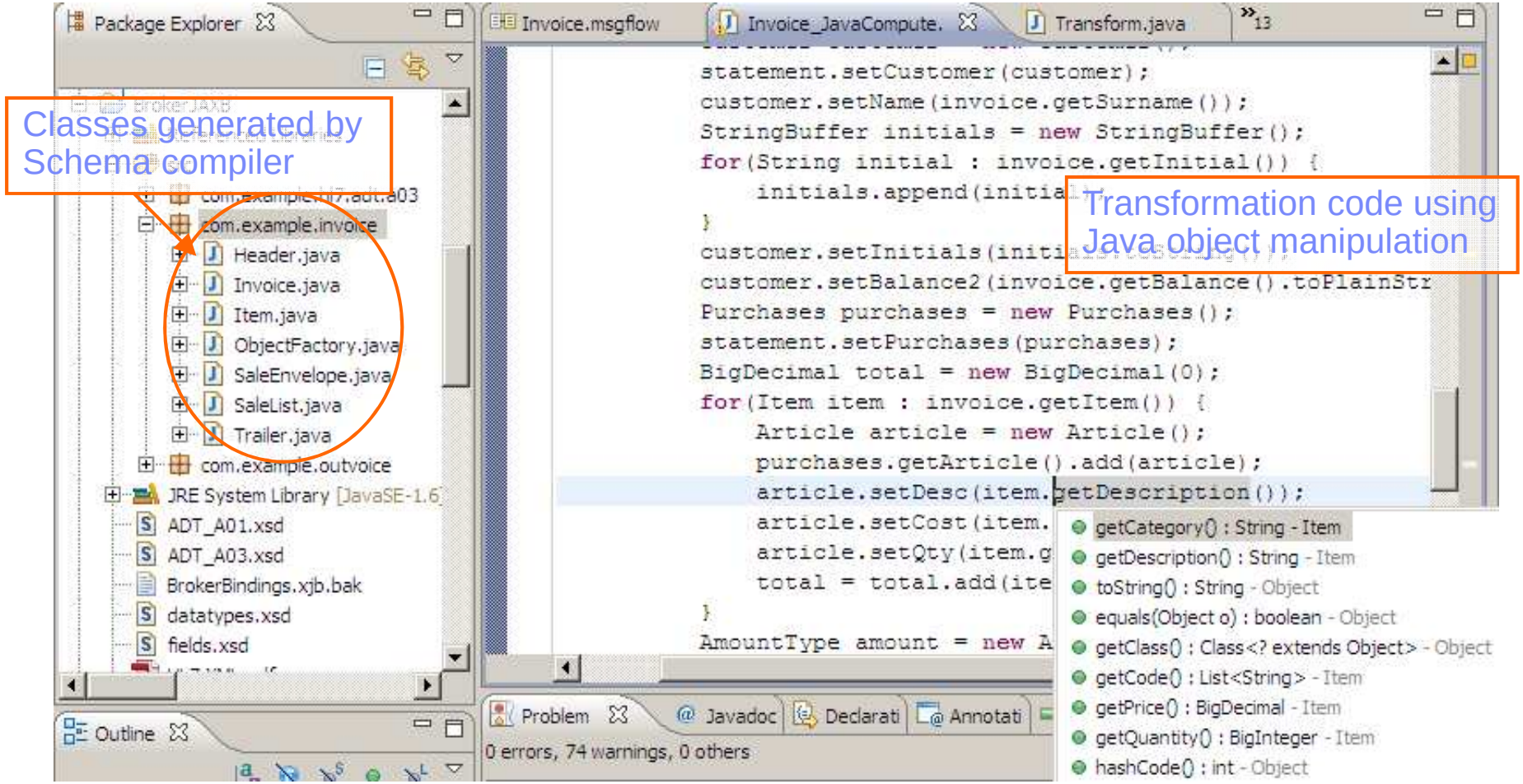
- **Methods for copying and moving sub-trees**
- **Use XPath to query data from the input message**

Introducing support for JAXB

New for 8.0.0.1

- **Java Architecture for XML Binding**
- **Allows Java developers to write transformation code using a Java object representation of the data**
  - JavaBean like representation – getter and setter methods are used to traverse, modify and build messages
  - Transformation logic can be coded without using MbElement API
    - Useful for writing code for use across multiple products, or migrating existing code into WMB
  - Full generation wizard and content assist support within WMB toolkit
- **JAXB comprises several components**
  - Schema compiler – generates a set of Java classes from an XSD
  - Schema generator – generates an XSD from Java classes
  - Binding runtime framework – for converting data between the two representations
    - Unmarsalling – converts data in the WMB logical tree into a set of Java objects
    - Marshalling – converts the Java objects back into a WMB tree

JAXB in a Java Compute node



Pros and Cons of JAXB versus MbElement API

	JAXB	MbElement API
+	▪ **Standard – built into J2SE JDK (from v6)** ▪ **Schema driven** – Enables content assist – Output marshalled tree always correct shape ▪ **Self-contained and portable across JAXB implementations** ▪ **Suitable for migrating code between products**	▪ **Best performance** ▪ **No input or output schema required**
–	▪ **The whole tree gets unmarshalled up front** ▪ **No benefit from parse on demand capability** ▪ **However, JAXB binder allows parts of the tree to be unmarshalled at a time**	▪ **Proprietary API** – Although XPath 1.0 is standard ▪ **User has to build output tree of correct shape** – Elements must be in correct order

Java Score Card

Performance and scalability	?	**Slight 'JNI' overhead when accessing string data in tree.** **Excellent performance for custom logic – highly scalable, but be aware of thread synchronisation issues**
Backend integration	✓	**Excellent integration with external systems and libraries**
Skill sets and learning curve	✓	**Most popular programming language worldwide**
Developer usability	✓	**Excellent tooling available for editing and debugging Java** **Language syntax is verbose for manipulating message trees** **Straightforward to build re-usable code libraries**
Portability and maintenance	✓	**JAXB ensures transformation logic can be portable across products**

PHP Compute Node

PHP

- **PHP is a dynamic scripting language used to implement web sites**
- **Easy to use – gentle learning curve**
- **Efficient syntax and library have evolved in open source**
  - Community driven to get more done in less time
  - Impressive results with little code
  - Extensive library support
  - Language suited to rapid incremental prototyping
- http://www.php.net
- **More than 3 million developers worldwide**
- **Predicted to grow to 5.5M developers**
  - 60% corporate by 2013
- **4th most popular language (after Java/C/VB)**
- **Customer demand for scripting support in the broker**
  - Allowing rapid development of message processing logic

Introduction

- **New general purpose programmable node**
- **Embeds the IBM Runtime for PHP**
  - Java implementation, fully compliant with PHP version 5.2
- **Message tree navigation syntax integrated into PHP language**
  - Inspired by ESQL path navigation and SimpleXML PHP extension
- **XPath 1.0 support**
- **Two PHP script styles are supported**
  - Declare a class with an **`evaluate()`** method
    - Gets invoked for each message
    - Annotations control message copying and routing behaviour
  - Plain script
    - No class declaration required
    - Users have to write more code to support message copying and routing
- **No state is retained by the node between messages**
  - Inherently thread-safe
- **Multiple dynamic output terminals for message routing**

PHP Example


```
<?php

class Hello {
    /**
     * @MessageBrokerSimpleTransform
     */
    function evaluate ($output, $input) {
        $output->XMLNSC->doc->greeting = 'Hello';
    }
}

?>
```

```
<doc>
  <greeting>Hello</greeting>
</doc>
```

Navigating the Message Tree

- **Each element in the tree is represented by an `MbsElement` object**
- **The element tree can be navigated in two ways:**
  - Path syntax

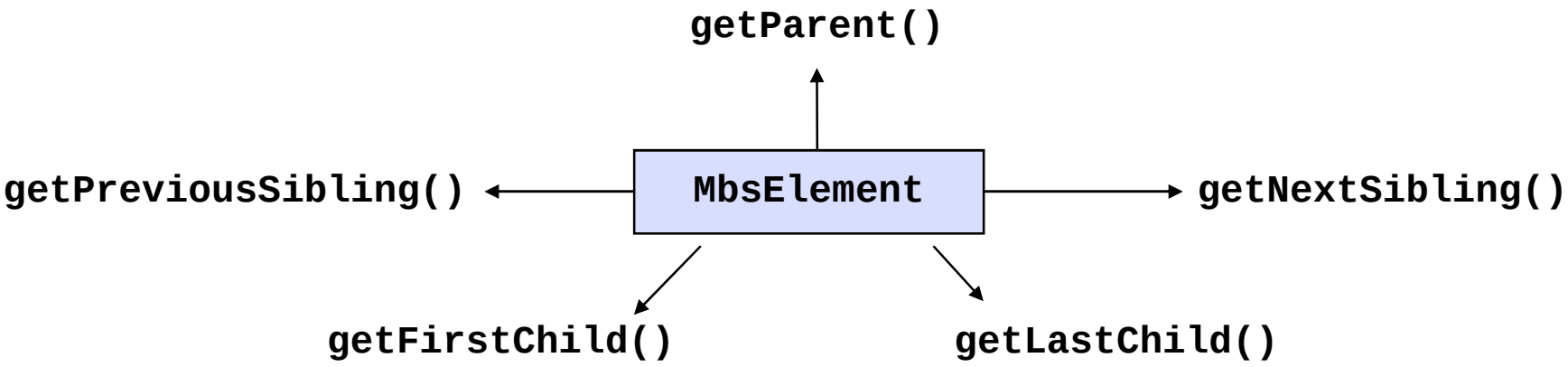
```
$output->MRM->bin->item = $input->XMLNSC->doc->ref->item;
```

    - Performs deep tree copy from RHS to LHS
    - Elements on LHS are created if they don't already exist
    - Navigation is not namespace-aware
  - API methods

```
$value = $items->getFirstChild()->getValue();
$catalog->addElement('Item', $value);
```

Element Navigation API

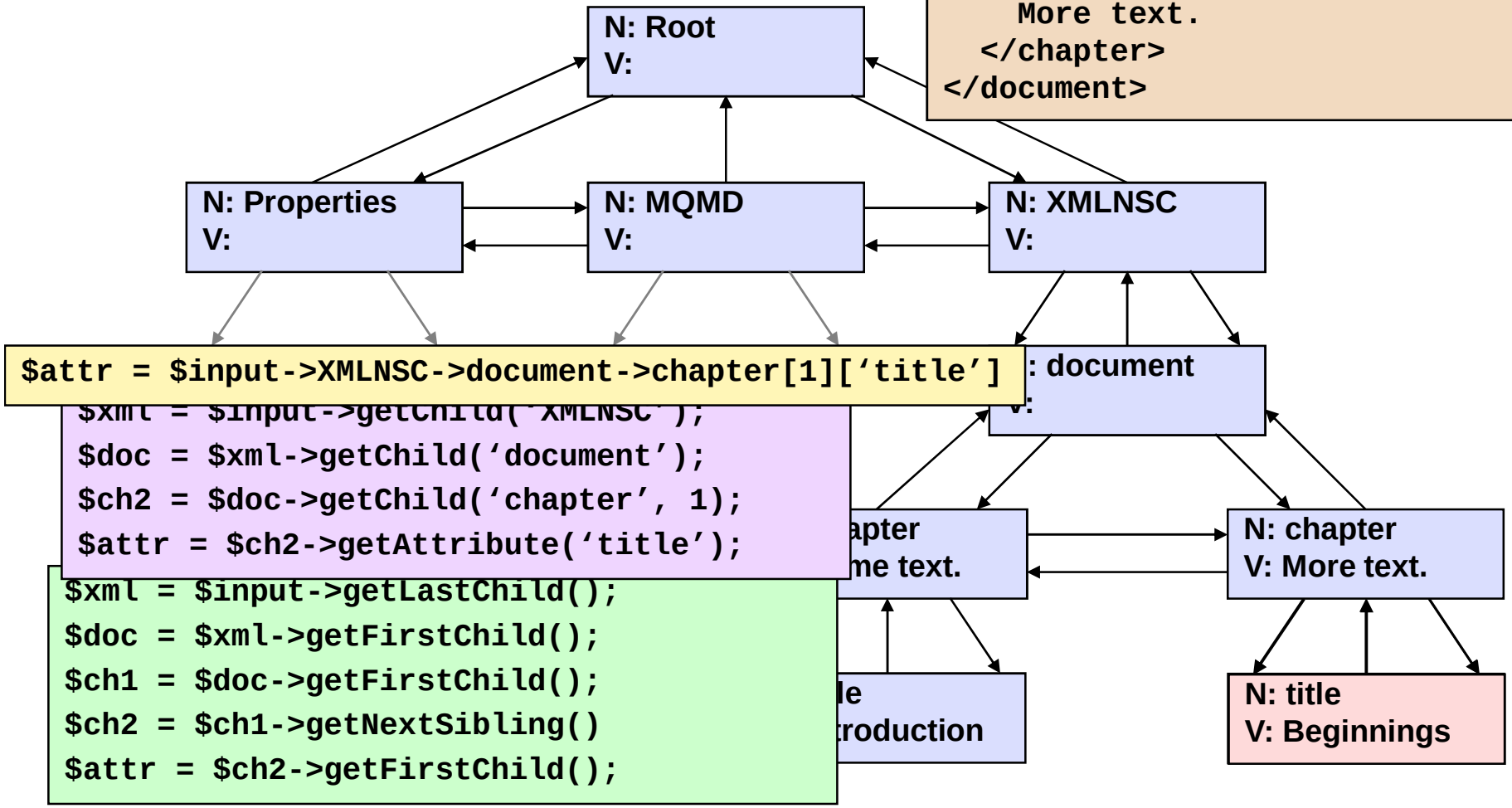
- `MbsElement` **has methods for traversing the message tree**



- **Get a child element by name (optional occurrence)**
  - `getChild(string $name [, int occurrence])`
- **Get an array of all children (optional namespace)**
  - `getChildren([string $namespace])`

Navigating the Message Tree

```
<document>
  <chapter title='Introduction'>
    Some text.
  </chapter>
  <chapter title='Beginnings'>
    More text.
  </chapter>
</document>
```




```
$attr = $input->XMLNSC->document->chapter[1]['title']
```

```
$xml = $input->getChild('XMLNSC');
$doc = $xml->getChild('document');
$ch2 = $doc->getChild('chapter', 1);
$attr = $ch2->getAttribute('title');
```

```
$xml = $input->getLastChild();
$doc = $xml->getFirstChild();
$ch1 = $doc->getFirstChild();
$ch2 = $ch1->getNextSibling()
$attr = $ch2->getFirstChild();
```

MbsElement – Repeating Structures

- **MbsElement supports the array operator `[]` to access repeating elements**

```
$second = $input->XMLNSC->doc->item[1];
```

- **It also supports the creation of repeating elements**

```
$output->XMLNSC->doc->item[] = 'foo';
```

  – This creates the **`XMLNSC`** and **`doc`** folders, if they don't exist
  – It creates a new **`item`** element regardless of whether one already exists

- **Can iterate over a repeating element:**

```
foreach($input->XMLNSC->doc->item as $item) {
    print $item;
}
```

- **An array can be used to create a repeating structure**

```
$list = array('ein', 'zwei', 'drei');
$output->XMLNSC->doc->number[] = $list;

<doc>
    <number>ein</number>
    <number>zwei</number>
    <number>drei</number>
</doc>
```

PHP Score Card

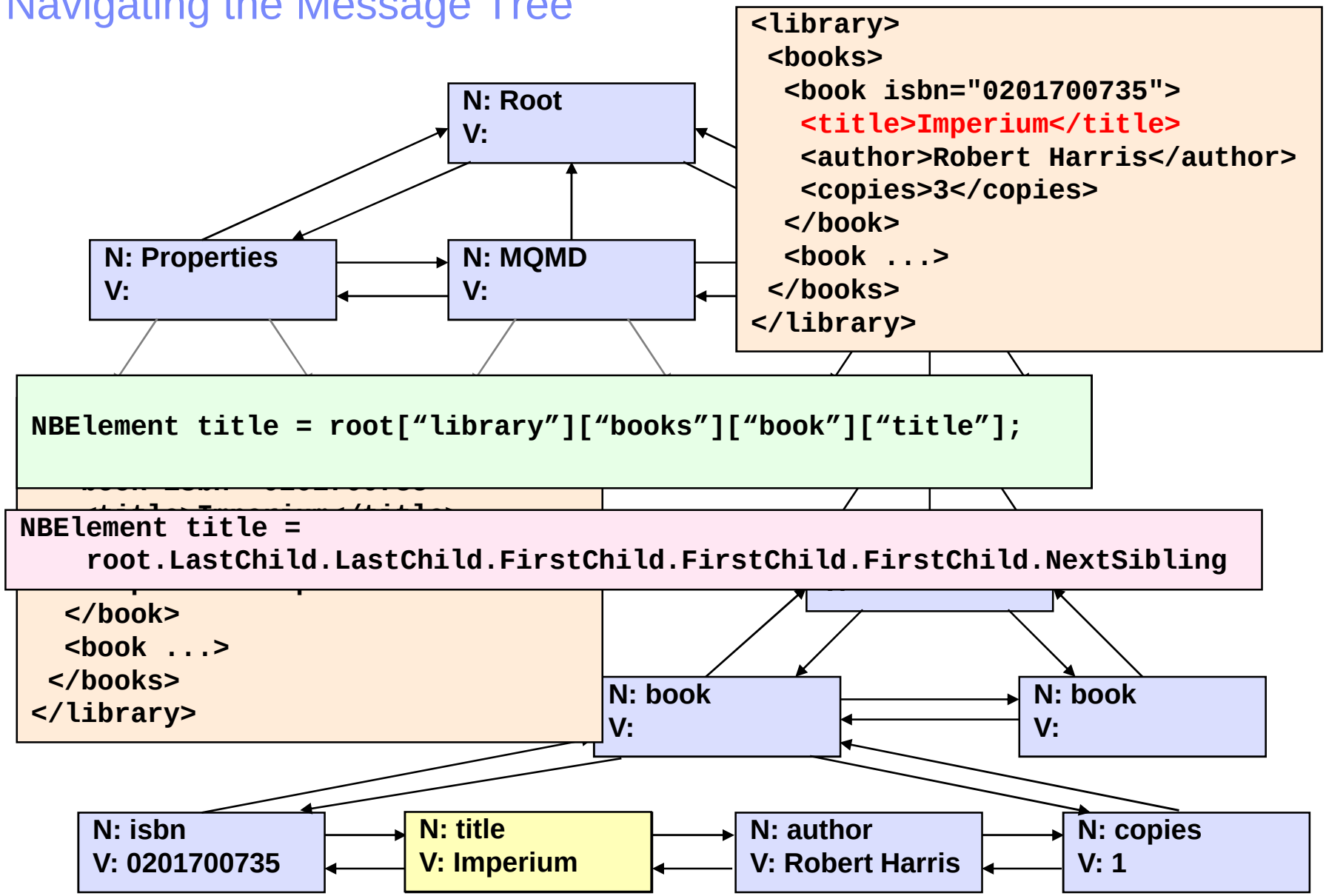
Performance and scalability	?	**Trades performance for developer productivity** **Layers on top of the Java node API**
Backend integration	✓	**Databases supported, integrates well with Java through a built-in Java bridge**
Skill sets and learning curve	✓	**Very easy to use and low barrier to entry** **Hugely popular scripting language**
Developer usability	?	**Very productive scripting language - supports a quick edit and test cycle without a re-deployment of the PHP scripts** **Excellent integration with broker – in particular the short hand notation for navigating message trees** **No integrated debug or editor support in the WMB toolkit**
Portability and maintenance	✓	**Merge utilities work as expected for source code control**

.NET Compute Node

Introduction

- **General purpose programmable node**
- **CLR v4 hosted inside the Execution Group**
- **Supports all CLR languages (e.g. C#, VB.NET, JScript, F#, etc.)**
- **API to work with messages and interact with broker**
- **Multiple dynamic output terminals for message routing**
- **Supports all message domains, headers and environment messages**
- **Full IDE support of Visual Studio**
  - Launch Visual Studio from Eclipse
  - Plug-ins to provide fast node creation
  - Content assist for easy access to the API
  - Debug your nodes using Visual Studio

Navigating the Message Tree



.NET Score Card

Performance and scalability	✓	**Excellent performance – comparable with ESQL and Java.** **CLR integrated tightly with broker internals**
Backend integration	✓	**Excellent integration with external systems and libraries**
Skill sets and learning curve	✓	**Very widely used programming languages**
Developer usability	✓	**Excellent tooling available for editing and debugging .NET code** **Straightforward to build re-usable code libraries**
Portability and maintenance	?	**Merge utilities work as expected for source code control** **Limited to supported Microsoft .NET platforms (Windows)**

Questions?