Reusability Analysis for Shipbuilding Components Modeled in XML and Java

Prof. Steven A. Demurjian, Sr.
Jeffrey R. Ellis
Computer Science & Engineering Department
The University of Connecticut

Prof. Donald M. Needham
Computer Science Department
U.S. Naval Academy

steve@engr.uconn.edu
JRE95001@mail.snet.net
needham@usna.edu
http://www.engr.uconn.edu/~steve
(860) 486 – 4818

Background

Part of the Integrated Shipbuilding Environments Consortium, Phase II Efforts

- **Focus:** Interoperability of Design, Versions, and Schema for Shipbuilding CASE Tools
- **Domain:** Design and Co-Production between Shipyards/Suppliers Developing a Submarine
- Facilitate the Exchange of Common Information Between Major Systems

- **Information Exchange Might Be:**
  - Early Binding: Application-to-Application (maximized use of XML Attributes)
  - Early Binding: Business-to-Business (maximized use of XML Elements)
  - Containment Oriented
Example

A Shipbuilding Parts Library with 1 Parts Manufacturer, 2 Shipyards (Different Companies)

- Parts Library Data in Different Forms at Each Shipyard, Req: Inter/Intra Shipyard Exchange
- Likely: Asynchronous XML-based Message System for Product Data (SOAP Protocol)
- Language Bindings Might Include Express to XML Schemas (meta data) and/or Java/C/C++
- Information Models:
  - Express
  - Java
  - XML DTDs

Together CC UML

Current Model

Currently have STEP AP227, an Express Model for Shipbuilding (e.g. CATIA 227)
- Anticipate Utilizing XML Schemas/Instances
  - Provide Common Form Across Enterprise
  - Allow Access to Tools to Handle the Import & Export of the Shipbuilding Models
- Current STEP Model Yields Impractical Results.
  - Recent Design Iteration has 200+ Classes
  - Need to Evaluate Restructuring the Design to Increase the Reuse Potential for the Domain
Electric Boat’s EDO

- EB Developed Enterprise Data Objects (EDO), a subset of STEP AP227
  - Subsetted by Considering only Detailed Design (3D Layout)
  - Focus is on Flow Analysis
  - Reduces Step AP227 to a Subset of 30 Classes from the 200+ Original Classes

- Goal: Have EDO Become a Java Entity Bean

EB Proposed Model - Dynamic View

- EXPRESS → XML Schema
- XML Schema → Java Classes
- Java Classes → UML Model
- EXPRESS → Java Objects
- Java Objects → XML Instances
Focus on Reuse

Consideration Being Given for Modeling Design in UML, with Translations to XML and Java
Want the Resulting XML and Java to be Reusable Within the Domain

Main Issue: How is Reuse Measured within the Domain?

- Will Domain (Shipbuilding) Influence the Definition of Reuse and of Dependencies?
- What Does it Mean to have Reusable XML Schemas? XML Instances?
- How is Reuse Measured in Java with its Classes, Components, Packages, Interfaces?

Reuse in the Shipbuilding Domain

Potential Modeling Translations

Express

XML

Java

UML
UConn/USNA Goals and Objectives

- Evolve UConn’s Design Reusability Evaluation (DRE) Tool for Use in the Shipbuilding Domain
  - Current DRE Evaluates C++ and Java Code
  - Focus on Java Capabilities of DRE
- Goal: Evaluating Reuse for XML and Java
  - Short-Range Goal: Investigate Reuse in Java with an Emphasis on Interfaces and Packages
  - Mid-Range Goal: Investigate Reuse in XML with an Emphasis on Schemas and Instances
  - Aside: Examine Usage of Together Control Center as Part of Research Process
Effort of PIs S. Demurjian/D. Needham
June 1 to August 31, 2000

S. Demurjian

- Background Work on XML
- Reading Schemas, Structures, & Datatypes
- Preparation and Planning for Fall 2000
- New Integrated Reuse Lecture
  [www.engr.uconn.edu/~steve/Cse298300/finallooreuse.ppt](www.engr.uconn.edu/~steve/Cse298300/finallooreuse.ppt)

D. Needham

- One-Week Course on XML
- Review Java-specific Capabilities of DRE
- Examination of Reuse as Related to Packages and Interfaces within Java
- Evaluating the Impact on Reuse Measurement for Java Packages and Interfaces

Planned Work
September 1 to December 31, 2000

Research on Reusability in Java

- Examine Reuse: Packages/~`Component’`
- Assess Impact on Reusability Framework

Research on Reusability in XML

- What are Reusable Components in XML?
- What are Dependencies in XML?

Work on DRE:

- Fine Tune DRE’s Java Parser
- Refine and Rework the GUI
- Planning for Including Levels of Generality
  
  Current DRE Supports General and Specific
  Expand for Levels of Generality
Other Planned Related Work

- DRE Project: CSE300 Distrib. Object Computing
  - Together/CC
  - Jeff Ellis Team Leader, 3-4 Team Members
  - Research on Reuse and Upgrading DRE
  - Expand Prior Work
    - See Web Site - Component-Based Reuse
      - www.engr.uconn.edu/~steve/Cse298300/Spring99Projs/sp99projs.html

- DRE in CT Insurance Department Project
  - Developing Java GUI (Swing, JDBC, RMI, …)
  - Examine Reusability of GUI to Transition Code from One Division to Another
  - Provides Test Bed for Tool and its Java Reusability Capabilities

Overview of Reuse Issues (DN)

- Designing/Developing Reusable Software Must Balance
  - The Structuring of Software to Capture Necessary Functionality
  - A Focus on Increasing the Reuse Potential of the Components, Packages, Classes, etc.

- Increasing Reuse Potential
  - Analysis of Reusability During Iterative Design and Development Process
  - Designer Benefits from Reuse-Specific Feedback during Design and Development
Reuse Perspective

Reuse Focus has Been on Consumer (Reuser) Instead of Producer Perspective

- Studies Measure Amount of Reuse a Project has Achieved (or Effort Saved by Reuse)
- Little Attention Given to Analyzing Reusability During Design/Development in Order to Produce Reusable Products

Our Focus …

- As Current Application is Developed, Can we Identify those “Components” Most Likely to be Reused in Future Systems
- Employ Objective Measurement Techniques

Three Classes of Software in a Typical Software Application

- Domain-Independent: 20%
- Domain-Specific: 65%
- Application-Specific: 15%

Two-Fold Goal

- Elevate Reuse to Equal Partner Starting with Design
- Focus on Domain-and-Organization Specific Reuse
Domain & Organization-Specific Reuse

- Consider Reusable Components from the Producer’s Perspective
  - Reuse Should Center on the Application's Domain
  - Specific Reuse Should Focus on an Organization's Future Systems

- Need Reusability Metrics That
  - Applied Iteratively During Design and Implementation of Software Life-cycle
  - Guide the Production and Identification of Reusable Components, Packages, Classes, etc.

Focus of Prior Reusability Work (SD)

- **General Class (G)**
  - Those Application Classes that Facilitate Domain-and-Organization Specific Reuse
  - Different Levels of Generality

- **Specific Class (S)**
  - Those Application Classes that are Limited to use in a Single Application

- Purposes
  - Determine Classes with Highest **Reuse Potential** for Organization’s Future Systems
  - Dependencies from General to Specific are both Non-Reusable and Hinder Reuse
General/Specific Class Characterization

- **General Class (G)**
  - Expected to be Reused in Future Applications
  - Abstract Classes/Root Classes/Non-Leaf Classes in Inheritance Hierarchies
  - Domain Independent/Domain Specific
  - What are Some Examples?

- **Specific Class (S)**
  - Only Applicable in Current Applications
  - Unlikely to be Reused in Future Applications
  - Classes that Retrieve from Company Database
  - Application Specific
  - What are Some Examples?

---

Related Characterization in Levels of Components - HTSS

- Item \((G^0)\)
- NonPerishItem \((G^1)\)
- PerishItem \((G^1)\)
- Environ \((G^1)\)
- ProduceItem
- DairyItem
- DeliItem
- SubZero
- RTemp

- Does \(R\) from Environ to PerishItem Make Sense?
- Should \(R\) be from PerishItem to Environ?
Related Characterizations in Levels of Components - HTSS

- Root classes for Items, ItemDB, etc., which are Most General.
  \( (G^3) \)

- Inventory Control/Other Components.
  \( (G^2) \)

- Classes Specific to Grocery Store Domain.
  \( (G^3) \)

Specific Applications for Big Y or Shaw’s or Stop/Shop (S)

Where do Changes for Other Domains Occur?

Reusability in HTSS Domain

- Root classes for Items, ItemDB, etc., which are Most General.
  \( (G^3) \)

- Inventory Control/Other Components.
  \( (G^2) \)

- Classes Specific to Grocery Store Domain.
  \( (G^3) \)

- Classes for Large Supermarket

- Classes for Specialty Supermarket

- Classes for 24 Hour Convenience

Specific Applications for Big Y or Shaw’s or Stop/Shop (S)
What are Dependencies Among Classes?

Object Inclusion: Class Contains a Instance of Another Object

Attribute Definition: Class Contains Attribute that is the Type of Another Object

Method Invocation: Class Invokes a Method Defined on Another Object

Goals: Classify and Understand Dependencies

EB Focus: Invariance Instead of Method Invocation

“Bad” Dependency Example

Assess “Good” vs. “Bad” Dependencies

Change “Bad” to “Good” by
- Changing Class from S to G or G to S
- Moving Code and/or Method Calls
- Splitting a Class into Two Classes
- Merging Two Classes

Diagram:

```
        G
       / \    
      G   S   G
     /     \  /
    S       G
```

Dependency
Facilitating Domain & Organization Specific Reuse (DN)

- Need Techniques for Distinguishing Levels of Components, Packages, Classes
  - Components May Overlap but Still be Considered Distinct Reusable Components

- Need Reuse Metrics To Measure Dependencies Within/Among Various Modeling Abstractions

- Need Guidelines Towards Improving Reusability
  - Guidelines Apply to Source and Destination of Dependencies Between Abstractions
  - Help to Determine Impact of Design Decision on Reusability of Other Abstractions

Domain/Organization-Specific Reuse

- Will Be Reused in an Army Tank
  - Attack Periscope (G)

- Will Be Reused in a Water Park Submarine
  - Navigation Periscope (G)

- Bearing To Contact
  - Ohio Class Navigation Periscope (S)
Upgrading Reusability Concepts (SD)

Prior Work Included Reusability Assessment for

- Classes (Regular and Abstract)
- Components (Subsets of Inheritance Hierarchy)
- Dependencies Among Classes/Components

Object-Oriented Design/Development Involves Other Abstractions, e.g., as Realized by Java

- Classes and Inheritance
- Packages
- Components (Java Beans, EJB, Designer-Specific Java Beans)

We Must Upgrade Our Paradigm and Framework!

Overview of Upgraded Design and Modeling Vocabulary (SD)

What are Available Abstractions?

- Classes
- Hierarchies
- Packages
- Components
- XML External Links
  - Link Base
  - Instance
  - Relationships

What are Different Dependencies Among Abstractions?

- Versions
- Aggregations
- Functional
- Inheritance
- Association
- Composition
What are Different Abstractions?

- Classes and Hierarchies
  - Building Blocks of Application
  - Independent Classes of All Types
  - Generalization and Specialization
  - Can Class Reference Interface?
- Interfaces/Abstract Classes
  - “Specification” of Behavior
  - Never Instantiated
  - Inconsistent Across Programming Lang.

Packages
- Organize Classes and/or Hierarchies into Named Conceptual Units
- Can Package Contain Interfaces?

Components
- Application-Oriented
- Logical and Physical Associations of Classes, Packages
- Purpose: Represent a “Meaningful” Chunk of Functionality

How Do Abstractions Relate?
How are Applications Conceptualized?

- Classes Defined/Grouped into Hierarchies
- Interfaces for Shared Behavioral Specification
- Packages as Collections of Classes and Interfaces
- Components Reuse Packages, Classes, Interfaces to Target Specific Application Function or “Chunk”

How are Abstractions Utilized?

- Use Individual Classes, Package or Subset of Package, Interface, Component, etc.
- Tools Use at Most a “Few” Select Classes, Packages and/or Components
- Tools that Span “Too Much” of Application Classes Represent Poorly Designed Software
- Should Tools Only Utilize “Components”?

Where is Reusability in Abstractions?

- Classes According to our Prior General/Specific Paradigm
  - Which Classes are General and Most Likely Reused in Future Applications?
  - Which Classes are Application Specific?
- Reuse of Interface Behavioral Specification by Multiple Classes
  - Are Some Interfaces Specific to Domain?
  - How do we Measure Interface Dependencies?
Where is Reusability in Abstractions?

- Packages Reuse Classes and Interfaces
  - General vs. Specific vs. Mixed Packages?
  - What are Package Dependencies?
- Can Packages be “Reused” in Other Packages?
- Components Reuse Classes, Interfaces, Packages, and Components!
  - General vs. Specific vs. Mixed Components?
  - What are Component Dependencies?

Current DRE Features (JE)

- Java/C++ Source Code Parsed
  - Classes Enumerated
  - Dependencies Recognized
- User Inputs System Knowledge
  - What classes are General? Specific?
  - What pairs of classes are Related?
- Coupling Counts Measured to Determine Reuse Potential
  - Counts of Specific Types of Dependencies
  - May Be Good for Reuse, Bad for Reuse, or Unaffecting Reuse Measurement
Current DRE Features

- Suggestions For Better Reuse Presented
  - Move Dependency To Subclass
  - Move Dependency To Superclass
  - Change General/Specific Definitions
  - Make Classes Related
  - Remove Coupling

Simulation Tool
- Allow Selection of Suggestions to Simulate
- Rerun Calculations “As-If” Code Had Been Re-written According to the Selected Suggestion
- Definitions Saved in Characterization File

DRE Components

- Source Code/Language Parser
- Metrics Scheme & Calculator
- Simulation Tool
- Graphical User Interface
- Help System (Common Classes)
### DRE’s Source Code/Language Parser

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shore Up Java Support</td>
</tr>
<tr>
<td>Add Java-Specific Constructs</td>
</tr>
<tr>
<td>Modularize Parser</td>
</tr>
<tr>
<td>New Common Data Structures</td>
</tr>
</tbody>
</table>

- Current Parser is C++ Upgrade
- Interface
- Package
- Java Bean (Component)
- Inner Class (Invisible to Outside World)
- Allow For Future Replacement
- Provide Communication Ability Across Modules

### Metrics Scheme & Calculation

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation to New Data Structures</td>
</tr>
<tr>
<td>Remove All GUI Traces</td>
</tr>
<tr>
<td>Incorporate Multiple Levels of Generality</td>
</tr>
<tr>
<td>Incorporate Package/Interface/Component</td>
</tr>
</tbody>
</table>

- Original Research Presented General vs. Specific
- Later Research Expanded to Spectrum (G0, G1, … Gx/Specific)
- Extension of Price Research Required
### Problem Domain Layering

<table>
<thead>
<tr>
<th>Layer</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>Person</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Part</td>
</tr>
<tr>
<td>Shipbuilding</td>
<td>Piping-Part</td>
</tr>
<tr>
<td>Company-Specific</td>
<td>Sub-Piping-Part</td>
</tr>
</tbody>
</table>

- Not OK
- OK

### DRE’s Simulation Tool

- Present Final Report of Selected Changes
- Incorporate Interface Idea into Simulation
- Presented Expanded Code Listing for Coupling Information
- Remove Simulation “ERROR 2”
Graphical User Interface Upgrades

- Professional-Looking Design
  - Previous GUI Was Experiment in Swing
  - Stability, Appearance, Modular
- Allow Flexibility in User Selections
  - Workspace to Allow Full Source Code Measurement
  - Selectively Choose Java Files From Multiple Directories
  - Coupling Analysis Granularity
- Integrated Help System
  - Intuitive Interface
  - Detailed Instructions
  - Theory Explanation
  - Interpretation of Metrics Discussion
- Ease of Use

New Class Abstractions

- Rationale
  - Allow for Modularization of Components
  - Allow Full Functionality of Java Structures
  - Allow Full Analysis of Couplings
  - Maintain Ease for Simulation
  - Allow Extension of Future Ideas
- Examples
  - Generality Class – Altered to Express Generality on Sliding Scale
  - Coupling Class – Alterable to Incorporate New Coupling Theory
  - Interface Class – Extension of New Data Type
Together Control Center

- An Interactive Software Design Tool
- Can Create UML Diagrams from Java Source Code
  - Java Changes Propagated to UML Diagrams
- Numerous Standard Metrics/Measurements Included
  - Coupling Counts
  - Ability to Develop Own Metrics
    - How Well Does this Work?
    - How Complex Can “Own” Metrics Get?
- Research – Can We Incorporate DRE Measurements as Part of TCC?

DRE’s New Visible Features

- Specification of Workspace
  - Flexible Definition of Included Classes
  - Strict Definition of Classes Excluded from Reuse Potential Measurement
  - Identification of Non-Workspace Dependencies
    - Including Java API Classes or Other Classes
- Allow Multiple Generality Levels
  - Ability to Fine-Tune Reusability Ideas
- Full Coupling Analysis
  - Full Dependency Granularity – to Variable and Method level
- Java Concepts of Interface, Package, (and Component?) Incorporated into Reuse Potential
DRE Planned Improved Features

- Java Parser Upgraded to Full-Strength
- User Help Improved
- User Friendliness Becomes Key Concern
- Expandable Code for Future Extensions
- GUI-Independence of All Data
- Simulator Bugs Removed
- New Ideas in Reuse Incorporated into System
- Java 1.3 Implemented
- Removal of “Multiple Cooks” Syndrome
- Professional-Strength Application Developed

Issues Involved in Upgrade

- (Initially) Drop Support for C++
  - Take Advantage of Java Constructs
  - Multiple Inheritance Prevented
  - Later Incorporation Will Require New Data Structures, Based on C++ Constructs

- Modular Design
  - Each Component Can Be Replaced Independently
  - Particularly Useful for GUI, Parser
  - Only Data Structures Must Remain
## Schedule

<table>
<thead>
<tr>
<th>Course</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structures</td>
<td>JE &amp; CSE300</td>
</tr>
<tr>
<td></td>
<td>September-October 2000</td>
</tr>
<tr>
<td>Java Parser</td>
<td>JE</td>
</tr>
<tr>
<td></td>
<td>October- November 2000</td>
</tr>
<tr>
<td>Metrics Scheme &amp; Calculator</td>
<td>JE</td>
</tr>
<tr>
<td></td>
<td>November-December 2000</td>
</tr>
<tr>
<td>GUI</td>
<td>CSE300</td>
</tr>
<tr>
<td></td>
<td>October-December 2000</td>
</tr>
</tbody>
</table>

## Schedule

<table>
<thead>
<tr>
<th>Course</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help System</td>
<td>CSE300</td>
</tr>
<tr>
<td></td>
<td>November-December 2000</td>
</tr>
<tr>
<td>Simulation Tool</td>
<td>JE</td>
</tr>
<tr>
<td></td>
<td>December 2000-February 2001</td>
</tr>
<tr>
<td>Together Control Center Application</td>
<td>CSE300 &amp; JE</td>
</tr>
<tr>
<td></td>
<td>September-December 2000</td>
</tr>
</tbody>
</table>
Reuse in XML (DN)

- A Typical XML Document is a Tree-Structured Logical Representation of Data
  - The Logical Rep of XML Data is Independent of the Data’s Presentation Structure
    - Data is Converted for Presentation Depending on the Type of the Client
    - Document Type Definitions (DTDs) Act as the Specification of How XML Data is Presented
  - Analysis of DTDs Allows Creation of Data Structures That Can be Manipulated by Application Programs, e.g., Java Objects

Investigating Reuse in XML Schemas

- Schemas Allow Boundary and Error Checking So XML Records can beParsed and Checked for Well-Formedness and Internal Validity
  - Example: XML Info of any Length and any Language can be Entered into a CDATA Field
  - However, a Database Might Accept Only 20 ASCII Characters in the Internal Field to which the CDATA Field Corresponds
- SOX (Schema for Object-Oriented XML) is Emerging. Seeks to Extend XML into Full Object Orientation.
  - http://www.w3.org/TR/NOTE-SOX/
The Role of SOX

SOX an Alternative to XML DTDs, Can Define the Same Class of Document Types

SOX Surpasses DTDs by Supporting:
1. An Extensible Set of Datatypes
2. Inheritance Among Element Types
3. Namespaces
4. Polymorphic Content
5. Embedded Documentation
6. Features for Robust Distributed Schema Management
7. All of the Above are Supported With Strong Type-Checking and Validation

The Role of SOX (Continued)

SOX Decreases the Complexity of Interoperation Among Heterogenous Applications
- Facilitates Software Mapping of XML Data Structures, Explicitly Expressing Domain Abstractions and Common Relationships
- Enables Reuse at the Document Design and the Application Programming Levels
- Supports the Generation of Common Application Components
Shorter Term XML Issues to Address

Items to Address in Research on Reuse in XML

- What are the Abstractions in XML?
- What are Dependencies Among Abstractions in XML?
- What Level of Reusability can be Attained from Each of the Different Abstractions?
- How can XML Dependencies be Restructured to Improve Domain & Organization-Specific Reuse of XML Components?
- Should we Examine Role of SOX in Increasing the Reuse Potential of XML Components?

Longer Term XML Issues to Address
Concluding Remarks/Looking Ahead