BoBs: Breakable Objects

Building blocks for flexible application architectures

Ph.D. Viva-Voce
July 23rd 2007

Vikram Jamwal
Advisor: Prof. Sridhar Iyer
IIT Bombay, Mumbai
Contents

- Problem and Motivations
- Breakable Object (BoB) basics
- BoB for application partitioning
- BoB as elements of reuse
- Related work comparisons
- Discussion and conclusions
Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions
Given

- Computing devices with various capabilities
- Networks which are diverse
- Applications need to adapt
  - to different deployment scenarios
Furthermore...

- Applications need to evolve
  - Change in user requirements
  - Different sets of user requirements
  - Restructuring for
    - better maintenance
    - better performance
Application adaptation

- A
- A^σ
- A'

Redesign
Refactor
Adapt
Adapt
Broader Problem

- Design and implement an application
- Such that
  - Given the **software** for one scenario, we can generate a version of application for a new scenario
    - Through refactoring (and/or) adaptation
    - Easily
    - Automatically?
Problems:

1. Environmental Heterogeneity
2. Distribution of Components
3. Functionality Partitioning

- (1) and (2) active areas of research, many solutions exist
- (3) still requires adequate attention
Motivating Example

- E-mail Application
- Different devices
  - PC, Web-Client, Mobile Device
- Different Modes / Scenarios
  - On-line
  - Disconnected
  - Off-line
Functionality Partitioning

- Apportioning application functionality into deployment specific component sub-sets
- Difficult to achieve in practice because we cannot draw clean lines of separation
  - Some functionality may span across multiple classes
  - A single class may contain multiple functionality
E-mail Internals (jwma)
E-mail-design

- Different partitioning for different versions
- Store class on server
- Folder class partitioned between user and server
More specific problem:

- Design and implement an OO application such that
  - functionality of constituent objects is factorable

- The part-units
  - are of desired granularity
  - can be easily extracted
  - are reusable

- How?
**Jwma Classes**

- **JwmaStoreInfo**
  - `JwmaFolder`

- **JwmaFolderImpl**
  - `m_Store`
  - `m_Folders`
  - `m_ActualMessage`

- **JwmaFolderList**
  - `m_Subfolder`
  - `List(from util)`
  - `m_Folders`

- **JwmaMessageInfoListImpl**
  - `m_MessageInfos`
  - MessageInfoListImpl is used to keep handy information about messages in a folder for sorting etc.

- **JwmaMessageInfo**
  - `m_ActualMessage`

- **JwmaMessage**
  - `m_ActualMessage`

- **JwmaTrashInfo**

- **JwmaInboxInfo**

- **JwmaMessagePart**
  - `-m_MessageParts[]`

- **JwmaMessagePartImpl**

- **JwmaDisplayMessage**
  - `-m_RepliedMessage`

- **JwmaStoreImpl**
  - `m_Store`
  - `m_Folders`

- **ComposeMessage**

- **MessageSorting(from util)**
Our solution:

- Reformulate basic component of the application so that it can be readily partitioned into sub-components

- Hence Breakable Objects (BoBs)

Main Advantage:

- Designing and implementing applications using BoBs makes them more flexible; specifically, more amenable to partitioning.
Other issues:
Problems with inheritance and composibility

- Decomposition
  - Duplicated Features
  - Inappropriate hierarchies
  - Duplicated wrappers

- Composition
  - Conflicting Features
  - Fragile Hierarchies
Other issues:

- Inheritance based composition mechanisms
- Problem of software contraction
- Large software sizes
  - Heavier and more complex versions
Implementation

+AUTHMODE_PRIVATE: int = 0
+AUTHMODE_ANY: int = 4
+AUTHMODE_ANY_RO: int = 8
+DB_INIT: byte[*] = {    }
-SIGNATURE_LENGTH: int = 8
-DB_RECORD_HEADER_LENGTH: int = 16
-DB_BLOCK_SIZE: int = 16
-DB_COMPACTBUFFER_SIZE: int = 64
-dbCache: java.util.Vector = new java.util.Vector(3)
-dbCacheLock: Object = new Object()
-recordStoreName: String
-uniquePath: String
-opencount: int
-draf: RecordStoreFile
~rsLock: Object
-recordStoreName: String
-uniquePath: String
-recordStoreName: String
-dbNextRecordID: int = 1
-dbVersion: int
-dbAuthMode: int
-dbNumLiveRecords: int
-dbFirstRecordOffset: int
-dbDataStart: int = 48
-dbFirstFreeBlockOffset: int
-dbLastModified: long
-dbState: byte[*] = new byte[DB_INIT.length]
-RS_SIGNATURE: int = 0
-RS_NUM_LIVE: int = 8
-RS_AUTHMODE: int = 12
-RS_VERSION: int = 16
-RS_NEXT_ID: int = 20
-RS_FREE_START: int = 24
-RS_LAST_MODIFIED: long
-RS_DATA_END: int = 44

<<create>> RecordStore()
<<create>> RecordStore(uidPath: String, recordStoreName: String, create: boolean)
+deleteRecordStore(recordStoreName: String)
+openRecordStore(recordStoreName: String, createIfNecessary: boolean): RecordStore
+openRecordStore(recordStoreName: String, createIfNecessary: boolean, authmode: int, writable: boolean): RecordStore
+openRecordStore(recordStoreName: String, vendorName: String, suiteName: String): RecordStore
+setMode(authmode: int, writable: boolean)
+closeRecordStore()
+listRecordStores(): String
+getName(): String
+getVersion(): int
+getNumRecords(): int
+getSize(): int
+getSizeAvailable(): int
+getLastModified(): long
+addRecordListener(listener: RecordListener)
+removeRecordListener(listener: RecordListener)
+getNextRecordID(): int
+addRecord(data: byte, offset: int, numBytes: int): int
+deleteRecord(recordId: int)
+getRecordSize(recordId: int): int
+getRecord(recordId: int): byte
+setRecord(recordId: int, newData: byte, offset: int, numBytes: int)
+enumerateRecords(filter: RecordFilter, comparator: RecordComparator, keepUpdated: boolean): RecordEnumeration

~recordStoreName: String
~uniquePath: String
~deleteRecordStore(recordStoreName: String)
~openRecordStore(recordStoreName: String, create: boolean)
~ dbNextRecordID: int
~ dbVersion: int
~ dbAuthMode: int
~ dbNumLiveRecords: int
~ dbFirstRecordOffset: int
~ dbDataStart: int = 48
~ dbLastModified: long
~ dbState: byte[*] = new byte[DB_INIT.length]
~ RS_SIGNATURE: int
~ RS_NUM_LIVE: int
~ RS_AUTHMODE: int
~ RS_VERSION: int
~ RS_NEXT_ID: int
~ RS_FREE_START: int
~ RS_LAST_MODIFIED: long
~ RS_DATA_END: int = 44

~getInt(data: byte, offset: int): int
~getLong(data: byte, offset: int): long
~putInt(i: int, data: byte, offset: int): int
~putLong(l: long, data: byte, offset: int): int
~getRecordIDs(): int
~compactRecords()
Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions
BoB Splits

split-config-1

A

split-config-2

A-1

A-2

A-3

A-4

A-5

m1
m2
m3
m4
m5
m6

Interdependent splits

Independent splits

split-config-2

m1
m2
m3
m4
m5
m6
Denotes *is-split-of* and *is-principal-of* relationships. The thick head lies towards the principal class’s side.
Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions
Program Reorganization

After Splitting

Client CL-1

Client CL-2

Client CL-3

A

m1() m2() m3() m4() m5() m6()
Redeployment
BODA: Breakable Object Driven Architecture
Online-Disconnected-Offline

NODE - USER

E-mail GUI

Folder-1

Folder-2

Msg List

Folder List

NODE - MAIL SERVER

IMAP

Folder

Msg List

Folder List

IMAP Folder

STORE

SPLIT CONFIG 1

SPLIT CONFIG 2

SPLIT CONFIG 3
# Programming Model

<table>
<thead>
<tr>
<th>Construct in JAVA</th>
<th>Status in JAVABoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class declarations</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>Allowed</td>
</tr>
<tr>
<td>abstract</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>final</td>
<td>Allowed (default)</td>
</tr>
<tr>
<td>class <em>Name of Class</em></td>
<td>Allowed</td>
</tr>
<tr>
<td>extends <em>Super</em></td>
<td>Not Allowed</td>
</tr>
<tr>
<td>implements Interface</td>
<td>Allowed</td>
</tr>
<tr>
<td>Field Declarations</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>private</td>
<td>Allowed</td>
</tr>
<tr>
<td>protected</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>package</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>static</td>
<td>Allowed</td>
</tr>
<tr>
<td>final</td>
<td>Allowed</td>
</tr>
<tr>
<td>transient</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>volatile</td>
<td>Allowed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Construct in JAVA</th>
<th>Status in JAVABoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method Declarations</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>Allowed</td>
</tr>
<tr>
<td>private</td>
<td>Allowed</td>
</tr>
<tr>
<td>protected</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>package</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>static</td>
<td>Allowed</td>
</tr>
<tr>
<td>abstract</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>final</td>
<td>Allowed</td>
</tr>
<tr>
<td>native</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>synchronized</td>
<td>Allowed</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
</tr>
<tr>
<td>Constructors</td>
<td>Allowed</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Allowed</td>
</tr>
<tr>
<td>Threads</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>Nested Class / Inner Class</td>
<td>Not Allowed</td>
</tr>
</tbody>
</table>
Inheritance and BoBs

Split Config-1

\[ S1 = M1, M2, M3 \]

\[ S2 = M4, M5 \]
Inheritance and BoBs

Split Config-2

\[ S1 = M1, M3, M4 \]
\[ S2 = M2, M5 \]
Inheritance and BoBs

A a = new A;
A b = new B; ?
A c = new C; ?

Each child can split A in a unique way.
Retaining old type?

- Case 1, A remains along with A1, A2
- Case 2, only A1 and A2 remain
- We prefer case 2, and
  - allow only interface inheritance with the condition that
    - all the methods of an interface are designated together

- **Recommended**
  - Aggregation or delegation as the principal composition mechanisms for BoBs.
    - Neater design, reduces complexity
We do class-level partitioning

If we allow object-level partitioning
  ● i.e. we allow a BoB to be split in more than one way
  ● For assignment, need to know the <type> of object on RHS and then convert it to the type being assigned (LHS)
    ● A ax = new A();
    ● A ay = new A();
    ● ax split ax1 = m1, m2, split ax2 = m3
    ● ay split ay1 = m2, split ay2 = m1, m3;
    ● ax = ay?

Such a support is not available in the present languages

Class level BoB partitioning - sufficient for meaningful applications
Splitting Engine

Program $P$

$P_{split}$ (to Deployment Engine)
Split Config

Format

Number of BoBs = n;
BoB 1
BoB Name {
  No of splits = k;
  Split 1 = (‘;’ separated list of methods specified as
           MethodName (list ArgumentTypes))
  ...
  Split k = ...
}
BoB n
BoB Name {
  ...
}

Properties

- Only public methods are specified.
- Every public method in each BoB has to be specified as part of some split.
- A method cannot belong to more than one split.
- Clubbed methods (identified by the *together* construct) cannot be split.
Splitting Engine Details

- Algorithms
- Program Equivalence
  - split and non-split program versions
- Equivalency Proofs
- Details
Deployment

■ Presently uses mechanisms by
  ● J-orchestra,
  ● Pangaea for BoB deployments

■ Both use RMI as the underlying distribution mechanism
Centralized Program (100% Java)

Distribution Requirements

Analyser

Backend-Adapter

for (CORBA) for (DOORASTHA) for (Java Party)

Distributed Program (backend-specific)

DOORASTHA

Executable Program
Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions
SLICE MANIPULATION LAYER

- Add BoB Slice
- Remove BoB Slice
- Modify BoB Slice
- Merge BoB Slice
- Delete BoB Slice

SOURCE CODE LAYER

A1 B1 C1 D1 E1
A2 B2 C2 D2 E2

Split

A

B

C

D

E
Problems with inheritance and composability

- Decomposition
  - Duplicated Features
  - Inappropriate hierarchies
  - Duplicated wrappers

- Composition
  - Conflicting Features
  - Fragile Hierarchies
BoB Operators

- Split, Merge
- Extract, Remove
- Addition, Subtraction
  - Overwriting
  - Hierarchical
- Replace
Operation: **split**

Selection Set: \( A, \{\{m1,m2\},\{m3\},\{m4,m5\},\{m6\}\} \)
Operation: merge
Selection Set: \{A_1, A_{23}, A_4\}
Operation: \textit{extract}

Selection Sets:
1. A, \{m1,m2\}
2. A, \{m6\}

Case (1)

Case (2)
Operation: **remove**

Selection Sets: \( A, \{m1,m2\} \)
Operation: **fragment addition – hierarchical**
Operation: **fragment subtraction** - *hierarchical*
Operation: **fragment subtraction - overwriting**

A (-) B
Operation: **fragment addition – overwriting**

$A (+) B$
Operation: **fragment replacement**

B # D
Example: Graph Product Line

- Family of classical graph applications
- Typical of product lines
  - No two applications will have same set of features
- Features of GPL
  - (Un)Directed, Weight, Search, Algorithm

\[
\text{GPL} := \text{Gtp} \text{ Wgt} \text{ Src} \text{ Alg}^+;
\]
\[
\text{Gtp} := \text{Directed} | \text{Undirected};
\]
\[
\text{Wgt} := \text{Weighted} | \text{Unweighted};
\]
\[
\text{Src} := \text{DFS} | \text{BFS} | \text{None};
\]
\[
\text{Alg} := \text{Number} | \text{Connected} | \text{StronglyConnected}
  | \text{Cycle} | \text{MST Prim} | \text{MST Kruskal} | \text{Shortest};
\]
Graph-Weighted
+void addAnEdge(Vertex start, Vertex end, int weight)

Edge-Weighted
+int weight
+EdgeConstructor(Vertex the_start, Vertex the_end, int the_weight)
+void adjustAdorns(Edge the_edge)
+void display()

Vertex-DFS
+boolean visited
+void VertexConstructor()
+void init_vertex(WorkSpace w)
+void dftNodeSearch(WorkSpace w)
+void display()

Workspace
+void init_vertex(Vertex v)
+void preVisitAction(Vertex v)
+void postVisitAction(Vertex v)
+void nextRegionAction(Vertex v)
+void checkNeighborAction(Vertex vsource, Vertex vtarget)

Vertex-BFS
+boolean visited
+void VertexConstructor()
+void init_vertex(WorkSpace w)
+void bftNodeSearch(WorkSpace w)
+void display()

Workspace
+void init_vertex(Vertex v)
+void preVisitAction(Vertex v)
+void postVisitAction(Vertex v)
+void nextRegionAction(Vertex v)
+void checkNeighborAction(Vertex vsource, Vertex vtarget)

GlobalVarsWrapper
+static LinkedList Queue = new LinkedList()
public BoBClass GraphOperations
{
  
  public static void main (args[]) {

    BoBClass Graph_UW = Graph_Core + Graph_Undirected + Graph_Weighted;
    Graph_UW g_uw = new Graph_UW();
    g_uw.display();

    BoBClass Graph_DW = Graph_UW - Graph_Undirected + Graph_Directed;
    Graph_DW g_dw = new Graph_Dw();
    g_dw.display();

    BoBClass Graph_DuW = Graph_DW - Graph_Weighted + Graph_UnWeighted;
    Graph_DuW g_duw = new Graph_DuW();
    g_duw.display();

  }
}
Contents

Problem and Motivations
BoB basics
BoBs for application partitioning
BoBs as elements of reuse
Related work comparisons
Discussion and conclusions
Related Work/Comparison

- Traditional Objects
- Related Composition Mechanisms
  - Traits, Jigsaw
- Fragmented Objects
- Fragmentation in Databases
- Application Partitioning
- Class Refactoring
- Multi Dimension Separation of Concerns
- Distributed Design Paradigms
Objects

- Objects can be considered an extreme case of BoB where:
  - all the interface methods are designated as *together*

- Other differences come from the programming model chosen for BoBs.
Operators in JIGSAW

[GiladBracha92]

- **Merge**
  - Yields the concatenation of two modules. Only non-confliction attributes are considered.

- **Modification**
  - Overriding \((M_1 \text{ override } M_2)\)

- **Name conflict resolution**
  - \(M \text{ rename } a \text{ to } b\)

- **Select**
  - Returns the value of attribute named \(a\) in \(r\)

- **Restrict**
  - Removes the attribute \(a\) from the module

- **Project**
  - Retain particular attribute(s) definition(s)

- **Freeze**
  - Statically binding an attribute \(a\)
  - Dual operation
    - \texttt{freeze\_all\_except } \(a\)

- **Attribute Visibility**
  - \(M \text{ hide } a, \ M \text{ show } a\)

- **Rename: Access to overridden definitions**
  - \texttt{copy } \(a\) as \(b\)
Operators in Traits [Toplas06]

- Composing Classes
  - Class = Superclass + State + Traits + Glue methods

- Trait Composition Operators
  - Sum (+)
  - Aliasing (->)
  - Exclusion (-)
Fragmented Objects (FOs)

- FO is a
  - Distributed shared object
  - Clients in different address space
  - Internally a set of fragments
  - Fragments communicate through communication channels

- Comparison
  - Very different concepts; distributed shared object v/s breakable objects
  - BoBs don’t carry the notion of distribution per say
  - BoBs can be used to build FOs
    - Possible to achieve dynamic FOs – form and the location of fragments keeps on changing
  - BoBs don’t retain a single identity after splitting
Fragmentation in Databases

- Horizontal and Vertical Fragmentation in Database systems: Class is an ordered relation: $C = (K, A, M, I)$
  - Horizontal Fragmentation: $C_h = (K, A, M, I')$, where, $(I' \subseteq I)$
  - Vertical Fragmentation: $C_v = (K, A', M', I)$, where, $(A' \subseteq A)$ and $(M' \subseteq M)$
  - Hybrid Fragmentation combination of the above two

- BoBs and Fragmentation in Databases
  - horizontal fragmentation and BoB splitting are not related at all
  - vertical fragmentation employs some similar lines, but focus is to find fragments techniques to optimize query.
  - simplified models as no shared fields considered in the latter case
  - BoBs look at splitting from a programming perspective
  - Configuration Files (CFs) make the process of finding the lines of splits external to BoBs
Application Partitioning

- J-orchestra, Pangaea, Coign, Addistant
- So far focused mainly on
  - finding optimal ways to partition an application among different nodes, and
  - component conversions into distributed components.
- Our focus is:
  - to have an entity which is more suitable for such partitioning
  - declarative approach to application partitioning.
- Granularity level of partitioning
  - objects or components,
  - our case, granularity level
    - finer
    - related to the methods of a BoB.
Class Refactoring

- Different refactoring methods have been proposed
- Class refactoring method
  - Extract Class
  - Extract Interface  [Fowler:catalogue]

- provides a means to create new class by moving the relevant fields *Move Field* and methods *Move Method* from the old class into a new class.
- The main intent here is to improve the code design by splitting bloated classes and creating new crisper classes.
- No comprehensive techniques exist to provide refactoring of application classes for functional partitioning
# Distributed Design Paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>SA-initial</th>
<th>SB-initial</th>
<th>SA-later</th>
<th>SB-later</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-Server</td>
<td>A</td>
<td>know-how resource B</td>
<td>A</td>
<td>know-how resource B</td>
</tr>
<tr>
<td>Remote Evaluation</td>
<td>know-how A</td>
<td>resource B</td>
<td>A</td>
<td>know-how resource B</td>
</tr>
<tr>
<td>Code on Demand</td>
<td>resource A</td>
<td>know-how B</td>
<td>resource</td>
<td>B</td>
</tr>
<tr>
<td>Mobile Agent</td>
<td>know-how A</td>
<td>resource</td>
<td>-</td>
<td>know-how resource A</td>
</tr>
<tr>
<td>Breakable Object</td>
<td>know-how A</td>
<td>resource</td>
<td>(A)-part-1, (knowhow)-part-1 resource</td>
<td>(A)-part-1, (know-how)-part-2 resource</td>
</tr>
</tbody>
</table>
Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions
Contributions of Work

- The concept of **Breakable Object (or BoB)**
- BoB Programming Model
  - Structure
- Application Partitioning
  - Splitting, Merging
  - Program Equivalence
    - Proof of equivalence
- BoB Driven Architecture (BODA)
- BoB Composition Mechanisms
Publications (related to this work)


3. **BoBs: Breakable Objects** (Poster Paper) 20th Object-Oriented Programming, Systems, Languages And Applications (OOPSLA) October 16-20, 2005, San Diego, California, USA Authors: Vikram Jamwal and Sridhar Iyer

4. **Mobile Agent based Realization of a Distance Evaluation System** 2003 International Symposium on Application and the Internet (SAINT 2003), Orlando, Florida, USA, Jan 27-31, 2003 Authors: Vikram Jamwal and Sridhar Iyer

5. **Mobile Agents for effective structuring of large-scale distributed applications** Workshop on Software Engineering and Mobility, ICSE 2001 at Toronto, Canada Authors: Vikram Jamwal and Sridhar Iyer
Thank you