Vikram Jamwal
(Ph.D. Student)
KR School of IT
IIT Bombay

BoBs: Breakable Objects

Sridhar Iyer
(Associate Professor)
KR School of IT
IIT Bombay

Abstract

Direct redeployment of an application from one scenario to the another through straightforward refactoring is difficult. Application objects need to be in a form amenable to partitioning.

We propose Brekable Objects (BoBs) as a solution. We show how BoBs may be used (BoB Driven Architecture) in an application and how BoBs are favourable to splitting and redeployment.

Background: *Why are we doing this ?*

- Distributed systems have grown from nodes with uniform *computing* and *communication* capabilities to nodes with widely varying capabilities. Application designed for one scenario may not be amenable to direct redeployment in another scenario.
- Application translation is difficult mainly due to:
  1. Functionality partitioning problem
  2. Environmental heterogeneity problem [5]
  3. Distribution of components problem [1]
- (2) and (3) active areas of research for long and many solutions exist; (1) still requires adequate attention.

What exactly is functionality partitioning problem?

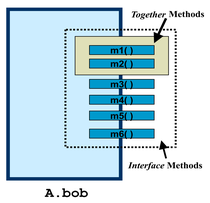
"Apportioning application functionality into - deployment specific component sub-sets."

Hard to achieve in practice as we cannot draw clean lines of separation:

- Some functionality may span across multiple classes.
- A single class may contain multiple functionality.

Need to reformulate basic components of an application so that they can be easily partitioned into sub-components. Hence ***Breakable Objects – BoBs***.

Fig. 1: BoB Features



BoB Definition

A BoB is an entity* in a program that can be *readily split* into sub-entities.

- Sub-entities should be formed in a manner:
  - they can replace the original BoB
  - maintain original semantics (operational) of the program
- Supports:
  - ***Interface*** - set of public methods
  - Construct - `together` - to denote methods designated inseparable by the BoB designer.

** BoB can be a class, object, component, or even a web-service.*

Figure 2: BoB Driven Architecture – Process Stages

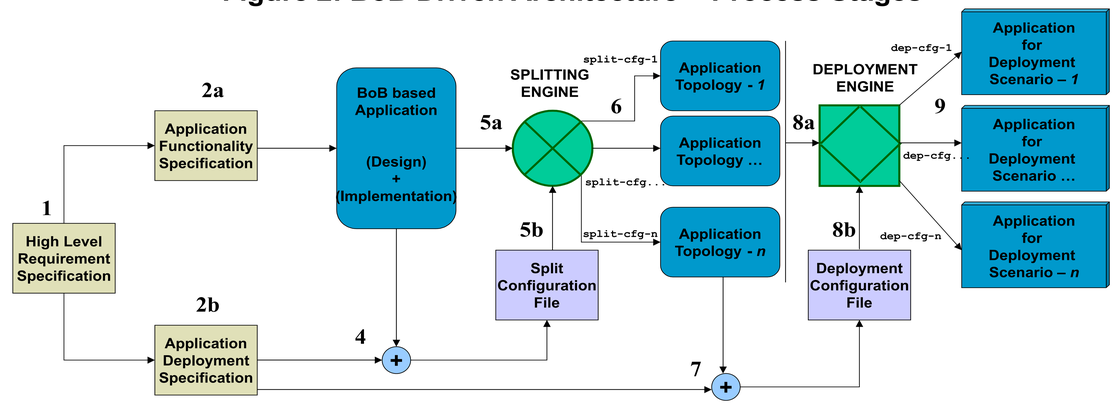
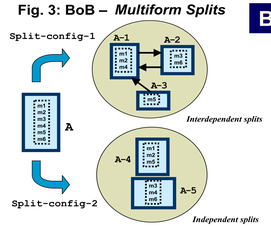


Fig. 3: BoB – *Multiform Splits*

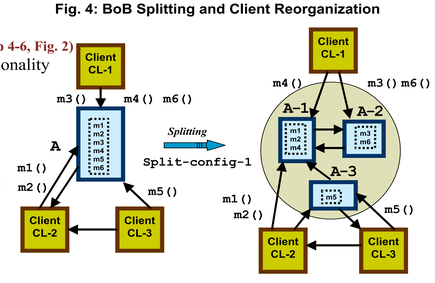


BoB Methodology

1. Design and Implementation (Step 1-3, Fig.2)

- *Deployment Independent* version of application
- Similar to normal o-o programs, plus following:
  - Application functionality ➡ set of (Objects + BoBs)
  - For each BoB specify interface + `together` methods

Fig. 4: BoB Splitting and Client Reorganization

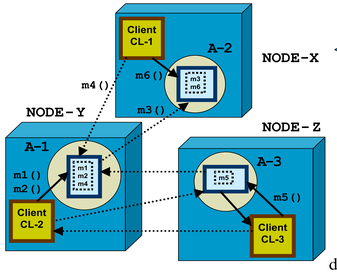


2. Splitting and Reorganization (Step 4-6, Fig. 2)

- For a deployment scenario, functionality mapped onto
  - Normal objects in the system
  - Interface methods of BoBs

  (Split configurations generation)
- Automated BoB splitting
- Automated client transformations: original refs to split-BoB refs

Fig. 5: BoB Redeployment



3. Redeployment (Step7-9, Fig. 2)

- Each application object is mapped to a node of application deployment setup. *(Deployment configurations* generation)
- Components are prepared for new distributed environments by doing source/binary level transformations on them.
- Resulting application components are distributed and deployed across the specified nodes.

JAVA$_{BoB}$ – *A Simplified Class Model*

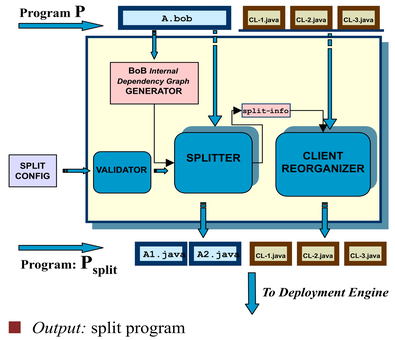
- Though simple, programming model is functionally comprehensive
- Only one additional programming/pre-processor construct – `together`
- No modifications to JVM™ required

Construct in JAVA	Status in JAVA$_{BoB}$
Class declarations	
public	Allowed
Abstract	Not Allowed
final	Allowed (default)
class *Name of Class*	Allowed
extends *Super*	Not Allowed
implements Interface	Allowed
Field Declarations	
public	Not Allowed
private	Allowed
protected	Not Allowed
package	Not Allowed
static	Allowed
final	Allowed
transient	Not Allowed
volatile	Allowed
Method Declarations	
public	Allowed
private	Allowed
protected	Not Allowed
package	Not Allowed
static	Allowed
abstract	Not Allowed
final	Allowed
native	Not Allowed
synchronized	Allowed
Miscellaneous	
Constructors	Allowed
Exceptions	Allowed
Threads	Not Allowed
Nested Class / Inner Class	Not Allowed

- Java$_{BoB}$ class differs from Java class principally in the following ways:
  - No *public* fields. Use *getters* and *setters*.
  - No *inheritance*. Use *aggregation + delegation*.
  - BoB cannot be an *active* object. However, can be accessed by other application threads.

Splitting and Deployment Engines

Fig. 6: Splitting Engine Mechanism



❖ **Splitting Engine:**

- *Input:*
  - BoB Classes
  - Java Classes
  - Configuration file
- *Output:* split program
  - Java Classes
- Multiple BoBs: iterate

❖ **Deployment Engine:**

- Presently uses mechanisms by J-orchestra [2], Pangea [3] for BoB deployment.
- A more specialized *BoB specific* version is under development.

Case Study – *Multimode* Distributed e-mail Client

- We may access e-mail in variety of scenarios - e.g. using a desktop on LAN or a PDA on 3GA network.
- We may want different versions:
  - Online, Offline, Disconnected
  - `Store` class on server
  - `Folder` class partitioned between user and server
- Our experience with icemail, jwma, pooka, popmail, etc.† showed that direct refactoring is not feasible.

† `java-source.net/open-source/mail-clients`

Fig. 7: An e-mail implementation model

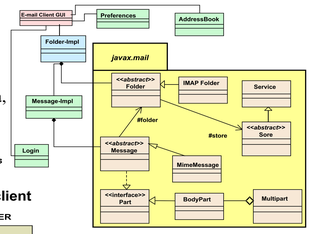
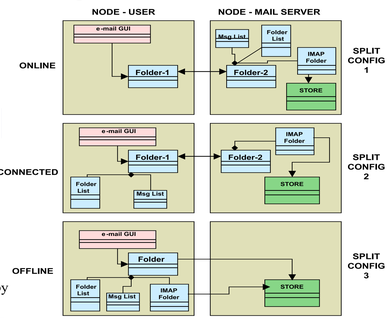


Fig. 8: Different versions of e-mail client



- We identified two objects for BoB based implementation:
  - `BoBFolder`
  - `BoBMessage`
- `split` and `depl` configs specified manually
- We obtain a flexible multi-mode e-mail client application *through BODA process*.

Conclusions and Future Work

- Need for structuring applications in a way that direct refactoring for different scenarios is practicable and easy.
- A BoB based architecture greatly facilitates *refactoring* for *redeployment*.
- Our ongoing and future work includes:
  - A specialized framework for BoB refactoring and redeployment.
  - Techniques for *mutation* and *merging* of BoBs.

References

1. **Galen Hunt**, "Automatic Distributed Partitioning of Component-Based Applications", Ph.D. Dissertation, Computer Science Department. University of Rochester, Rochester, NY, July 1998.
2. **Eli Tilevich**, Yannis Smaragdakis, "J-Orchestra: Automatic Java Application Partitioning", *European Conference on Object-Oriented Programming* (ECOOP), Malaga, June 2002.
3. **André Spiegel**, "Automatic Distribution of Object-Oriented Programs", PhD Thesis. FU Berlin, FB Mathematik und Informatik, December 2002.
4. **V. Jamwal and S.Iyer**, "BoB Based Programming and Formalizations", Technical Report KReSIT, IIT Bombay, 2005. Available at: `www.it.iitb.ac.in/~vikram/bob-formal.pdf`
5. **Marija Mikic-Rakic**, "Software Architectural Support for Disconnected Operation in Distributed Environments", PhD Thesis. University of Southern California, 2004.

www.it.iitb.ac.in/~vikram

e-mail: vikram@it.iitb.ac.in