BoBs: Breakable Objects

Vikram Jamwal
KR School of IT,
IIT Bombay, INDIA
vikram@it.iitb.ac.in

Sridhar Iyer
KR School of IT,
IIT Bombay, INDIA
sri@it.iitb.ac.in

ABSTRACT

Direct redeployment of an application from one scenario to another through straightforward refactoring is difficult. Application objects need to be in a form amenable to partitioning. We propose *Breakable Objects - BoBs*, as a solution. We show how BoBs may be used(*BoB Driven Architecture*) in an application and how BoBs are favorable to splitting and redeployment.

Categories and Subject Descriptors: D.1.5 [**Programming Techniques**]: - Object-Oriented Programming; D3.3 [**Programming Languages**]: - Constructs and Features

General Terms: Design, Languages, Theory

Keywords: Objects, Refactoring, Application Partitioning

1. BACKGROUND

Distributed systems have grown from having nodes with uniform computing and communication capabilities, to having nodes with widely varying capabilities. The underlying communication networks also have become more complex and heterogeneous in nature. A given application may need to be run in these different *deployment scenarios*. For example, we may access an e-mail in a variety of situations e.g. using a desktop on LAN or a PDA on 3GA network. Ideally, given an application designed for one scenario, one should be able to generate applications for a new scenario through an automated refactoring process[4].

However, this is extremely difficult in practice for the following reasons: **(i) Functionality partitioning problem:** This involves apportioning application functionality into component sub-sets *suitable* for redeployment in new scenarios. **(ii) Distribution problem:** This involves distributing an application's component across different nodes and making them work as distributed components. This might involve modifying application's source code, application's binaries prior to execution components, or manipulating application's execution through run-time interventions [2]. **(iii) Environmental heterogeneity problem:** This involves dealing with differences in environments encountered due to hardware and software constraints on the target environment, e.g., CPU speed, link characteristics, battery power, available system software, operating systems, run time libraries etc. [5].

Of these, component distribution and environmental heterogeneity (points (ii) and (iii)) have been areas of active research for quite some time. *Functionality partitioning of application* (point (i)), however, remains an important problem, which still requires much attention. In order to automate (i), the main requirement is: application functionality should cleanly separate in terms of discrete components that can be grouped into deployment-specific subsets. This is hard to achieve in practice as we may not be able to draw clean lines of separation through an application - *some functionality may span across multiple components, or a single component may include parts of multiple functionality.*

For example, in an e-mail application, `Folder` class may have its functionality partitioned between client and server for *online* and *disconnected* modes. For the *offline* mode, it may be required only on client.

Hence, we need application component (e.g. `Folder`) in a form such that its functionality can be easily partitioned into sub-entities. Due to complexity of various features, traditional objects are not always suitable for such partitioning.

We propose the concept of *Breakable Object - BoB*. A BoB is similar to an object in a class based object-oriented system, but has a simplified structure to allow easy refactoring of its functionality. We have found that BoB-based application design greatly facilitates automated refactoring of the application for various redeployment scenarios.

2. BOB OVERVIEW

An informal definition of BoB is as follows: A BoB is an entity (class/object) in a program that can be *readily split into sub-entities*. Sub entities should be so formed that they can replace the BoB while retaining the semantics (operational) of the original program.

Additionally, BoB supports an interface and has an added construct - **together** - to denote the groupings of its interface methods which are designated inseparable by the designer of the BoB.

3. BOB PROGRAMMING MODEL

Our programming model for **BoBs**, called **Java**$_{BoB}$ is based on the object oriented language Java. Though simple, the programming model is functionally comprehensive.

BoB class resembles a Java class except the restrictions that are placed on certain features. There is an additional programming language construct in Java$_{BoB}$, viz., *together*, which is used to specify inseparable methods. A preprocessor generates Java class definition files (.java files) from the BoB class definition files (.bob files). Thus Java BoBs can be used with existing Java Virtual Machines(JVMs) with-

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

out any modification to the latter. Due to lack of space, we omit the formal description of a BoB Class here; we provide it in [3].

The Java$_{BoB}$ classes differs from those in Java principally in the following ways: **1. Public fields:** No public fields are exported by the BoB. The designer provides *getter* and *setter* methods for accessing the fields if required. **2. Inheritance:** The class that needs to be split cannot be a derived class. The only class that a BoB class implicitly inherits from is `object`, the root Java object class. Same applies to the interface inheritance. Also each BoB is a *final* class. We propose the use of aggregation and delegation, as the preferable composition mechanisms for BoBs. **3. Threading:** BoB is not an active object; that is, it cannot run as separate threads on its own. However, BoB methods can be accessed by different threads in a program and we can specify the methods as `synchronized`.

4. BOB DRIVEN ARCHITECTURE

We explain here briefly the stages involved in a BoB based programming process:

1. Program Design and Implementation The program designer proceeds in a manner similar to object-oriented analysis and design, and uses requirement guidelines to divide the application functionality into a set of objects and BoBs. In this stage a deployment independent version of the application is prepared.

2. Splitting and Reorganization The *split-configurations* for a given scenario are specified for all the relevant BoBs. A *Splitting Engine* prepares class-definitions for the new set of BoB-splits that will replace the original BoB in the program. The rest of program is reorganized to convert the references to original BoBs into references to their splits. In this stage application functionality is partitioned into *node-specific subsets* of objects.

3. Redeployment Each application component is first mapped to a node of the application deployment setup *(deployment configuration specifications)*. Components are prepared for these new distributed environments by doing source /binary level transformations on them. These application components are finally redistributed across the specified nodes of the network.

In our work so far, we have provided the guidelines for BoB based program design and implementation, and evolved the algorithms for BoB splitting and reorganizations. These algorithms have been verified for correctness using Abstract State Machines models for Java$_{BoB}$ [3]. The redeployment Engine is still under development, and currently uses mechanisms developed by J-orchestra [8], Pangaea [7] for BoB deployments.

Case Study: We have applied BoB-based programming and restructuring process to a real-world application - a distributed e-mail application client. We are able to obtain three configurations for this application - offline, disconnected and online, through automated refactorings. We illustrate this case-study in the poster.

5. RELATED WORK

We discuss here the related work in application partitioning and object refactoring areas.

Application Partitioning: Application partitioning systems like J-orchestra [8], Pangaea [7] , try to automate application partitioning of arbitrary Java programs, Coign [2] does partitioning of COM based applications. Work on the application partitioning has so far focused mainly on finding optimal ways to partition an application among different nodes, and component conversions into distributed components. Our focus is: (i) to define an entity which is more suitable for such partitioning (ii) automate the partitioning process, given an external specification of deployment configuration. This makes our approach a *purely declarative way* of application partitioning. Additionally, BoBs support a finer granularity level of partitioning than these systems.

Class Refactoring: Different methods of refactoring have been proposed in literature [1] [4][6]. Class refactoring methods like *extract class*, *extract interface* etc. [1] provide means to refactor classes for improving designs of the existing code. However, no comprehensive techniques exist to provide refactoring for redeployment, as proposed in this paper.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have motivated the need for structuring programs in such a way that they can be easily refactored for deployment in various scenarios. Toward this end, we have developed the notion of BoBs as an entity in a programming language and also presented the methodology for BoB-based programming architecture. Although, in this paper we concern ourselves with *class-based* programming language models only, the definition of BoB is generic and is applicable to an object in object based systems, a component in component driven systems, or a service in service oriented systems. We are in the process of developing a specialized BoB deployment engine. In future, we also plan to investigate mechanisms by which two BoBs can be merged.

7. REFERENCES

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: improving the design of existing code*. Object Technology Series. Addison-Wesley, 1999.

[2] G. C. Hunt. *Automatic distributed partitioning of component-based applications*. PhD thesis, University of Rochester. Dept. of Computer Science, 1998.

[3] V. Jamwal and S. Iyer. BoB Based Programming and Formalizations. Technical Report KReSIT, IIT Bombay, India, 2005. Available at: `www.it.iitb.ac.in/~vikram/bob-formal.pdf`.

[4] T. Mens and T. Tourwe. A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, Feb. 2004.

[5] M. Mikic-Rakic. *Software Architectural Support for Disconnected Operation in Distributed Environments*. PhD thesis.

[6] W. F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois, Urbana-Champaign , IL , USA, 1992.

[7] A. Spiegel. Pangaea: An automatic distribution front-end for java. In J. D. P. R. et. al., editor, *IPPS/SPDP Workshops*, pages 93–99, 1999.

[8] E. Tilevich and Y. Smaragdakis. J-orchestra: Automatic java application partitioning. In B. Magnusson, editor, *ECOOP*, volume 2374 of *Lecture Notes in Computer Science*, pages 178–204. Springer, 2002.