Dynamic Software Updates: 
A VM-centric Approach

Suriya Subramanian\textsuperscript{1}  Michael Hicks\textsuperscript{2}  
Kathryn S. McKinley\textsuperscript{1}

\textsuperscript{1}Department of Computer Sciences  
The University of Texas at Austin 

\textsuperscript{2}Department of Computer Science  
University of Maryland, College Park

June 16, 2009  
ACM SIGPLAN PLDI 2009
Motivation

- Software applications change all the time
- Deployed systems must be updated with bug fixes, new features
- Updating typically involves: stop, apply patch, restart
- Not desirable
  - Safety concerns
  - Revenue loss
  - Inconvenience
Dynamic software updating

Version 1 process

Code
bytecodes
machine codes
Heap

Stacks

Version 2 process

Code
bytecodes
machine codes
Heap

Update Code

Update Data

Update Stacks

Check update safety
Dynamic updating systems

- Special-purpose architectures, application-specific solutions exist
- General-purpose solutions gaining strength
  - K42, Ksplice for OS updates
  - Polus, Ginseng for C applications
- Not for managed languages
Our solution

- **Jvolve** - a Java Virtual Machine with DSU support
- Key insight: Extend existing VM services
- No DSU-related overhead during normal execution
- Support updates to real world applications

*Dynamic software updating in managed languages can be achieved in a safe, flexible and efficient manner by naturally extending existing VM services.*

*DSU support should be a standard feature of future VMs.*
Jvolve - System overview

- Update
- Preparation
- Tool

Current version

Foo.java

New version

Foo.java

Foo.java

Foo.java

changed methods

Jvolve Transformers.java

changed classes

bytecodes

machine codes

Stacks

Code

Heap

Slide 6
Supported updates

- Changes within the body of a method
  ```java
  public static void main(String args[]) {
      System.out.println("Hello, World.");
      System.out.println("Hello again, World.");
  }
  ```

- Class signature updates
  - Add, remove, change the type signature of fields and methods
  ```java
  public class Line {
      private final Point2D p1;
      private final Point3D p1;
      ...
  }
  ```

- Signature updates require an object transformer function
Check for update safety

Version 1 process

Code
bytecodes
machine codes
Heap

Check update safety

Update Stacks

Update Code

Update Data

Version 2 process

Stacks

bytecodes
machine codes
Heap
Safe point for the update

- Update must be atomic
- Updates happen at “safe points”
- Safe points are VM yield points, and restrict what methods can be on stack
- Extend the thread scheduler to suspend all application threads
- If any stack has a restricted method, delay the update
Restricted methods

(1) Methods changed by the update
(2) Methods identified by the user as unsafe based on semantic information about the application

Install return barriers that trigger DSU upon unsafe method’s return

(3) Methods whose bytecode is unchanged, but compiled representation is changed by the update

- Offsets of fields and methods hard-coded in machine code
- Inlined callees may have changed

Utilize on-stack replacement to recompile base-compiled methods
Restricted methods

(1) Methods changed by the update

(2) Methods identified by the user as unsafe based on semantic information about the application

Install return barriers that trigger DSU upon unsafe method’s return

(3) Methods whose bytecode is unchanged, but compiled representation is changed by the update
   - Offsets of fields and methods hard-coded in machine code
   - Inlined callees may have changed

Utilize on-stack replacement to recompile base-compiled methods
Install a return barrier for \(d()\). Wait till it returns. On-stack replace new machine code for \(c()\).
Update code

Version 1 process

Code
bytecodes
machine codes
Heap

Update Code
Update Data
Check update safety
Update Stacks

Version 2 process

Code
bytecodes
machine codes
Heap

Update Stacks
Update code

- Modify class loader to recognize new versions of classes
- Install new versions of classes and methods
- Rely on Just-in-time Compiler to compile new versions of methods on demand
- Extend On-stack replacement to update active methods
Update data

Version 1 process

Version 2 process

Check update safety

Update Stacks

Update Code

Update Data

Stacks

Heap

bytecodes

machine codes
public class User {
    private final String username, domain, password;
    - private String[] forwardAddresses;
    + private EmailAddress[] forwardAddresses;
    public User(...) {...}
    public String[] getForwardedAddresses() {...}

    public void setForwardedAddresses(String[] f) {...}
}

public class ConfigurationManager {
    private User loadUser(...) {
        ...
        User user = new User(...);
        String[] f = ...

        user.setForwardedAddresses(f);
        return user;
    }
}
public class User {
    private final String username, domain, password;
    - private String[] forwardAddresses;
    + private EmailAddress[] forwardAddresses;
    public User(...) {...}
    - public String[] getForwardedAddresses() {...}
    + public EmailAddress[] getForwardedAddresses() {...}
    - public void setForwardedAddresses(String[] f) {...}
    + public void setForwardedAddresses(EmailAddress[] f) {...}
}

public class ConfigurationManager {
    private User loadUser(...) {
        ...
        User user = new User(...);
        - String[] f = ...;
        + EmailAddress[] f = ...;
        user.setForwardedAddresses(f);
        return user;
    }
}
public class v131_User {
    private final String username, domain, password;
    private String[] forwardAddresses;
}
public class JvolveTransformers {
...
    public static void jvolveClass(User unused) {} 
    public static void jvolveObject(User to, v131_User from) {
        to.username = from.username;  
        to.domain = from.domain;  
        to.password = from.password;
        // to.forwardAddresses = null;  
        int len = from.forwardAddresses.length;
        to.forwardAddresses = new EmailAddress[len];
        for (int i = 0; i < len; i++) {
            to.forwardAddresses[i] =
                new EmailAddress(from.forwardAddresses[i]);
        }
    }
}
public class v131_User {
    private final String username, domain, password;
    private String[] forwardAddresses;
}

public class JvolveTransformers {
    ...

    public static void jvolveClass(User unused) {}
    public static void jvolveObject(User to, v131_User from) {
        to.username = from.username;
        to.domain = from.domain;
        to.password = from.password;
        // to.forwardAddresses = null;
        int len = from.forwardAddresses.length;
        to.forwardAddresses = new EmailAddress[len];
        for (int i = 0; i < len; i++) {
            to.forwardAddresses[i] =
                new EmailAddress(from.forwardAddresses[i]);
        }
    }
}
Transforming objects in the GC

Before

After

Happens in two steps
- Garbage collector creates an additional empty copy for updated objects
- Walk through and transform all these objects
Jvolve GC

Slide 19
Jvolve GC

To space (old version objects)

To space

From space

a
b
c

a
b
c

Pointer

Forwarding pointer

To space (old version objects)
jvolveObject(Node to, old_Node from) {
    to.data = from.data;
    to.next = from.next;
    if (to.next != null) {
        to.next.prev = to;
    }
}
jvolveObject(Node to, old_Node from) {
   to.data = from.data;
   to.next = from.next;
   if (to.next != null) {
      to.next.prev = to;
   }
}
jvolveObject(Node to, old_Node from) {
    to.data = from.data;
    to.next = from.next;
    if (to.next != null) {
        to.next.prev = to;
    }
}
jvolveObject(Node to, old_Node from) {
    to.data = from.data;
    to.next = from.next;
    if (to.next != null) {
        to.next.prev = to;
    }
}
Jvolve GC

Slide 19
Jvolve GC

From space

To space

To space (old version objects)
Application Experience

- Jetty webserver
  - 11 versions, 5.1.0 through 5.1.10, 1.5 years
  - 45 KLOC
- JavaEmailServer
  - 10 versions, 1.2.1 through 1.4, 2 years
  - 4 KLOC
- CrossFTP server
  - 4 versions, 1.05 through 1.08, more than a year
  - 18 KLOC
What works

Support 20 of 22 updates

- 13 updates change class signature by adding new fields
- Several updates require On-stack replacement support
- Two versions update an infinite loop, postponing the update indefinitely
**Jvolve performance**

No overhead during steady-state execution

![Graph showing throughput and latency for Jvolve and Jvolve (updated) configurations.](Image)
**Conclusion**

- **Jvolve**, a Java VM with support for Dynamic Software Updating
- Most-featured, best-performing DSU system for Java
- Naturally extends existing VM services
- Supports about two years worth of updates

*Dynamic software updating in managed languages can be achieved in a safe, flexible and efficient manner.*

Source code and other information:  
http://www.cs.utexas.edu/~suriya/jvolve