A Graph-based Approach to API Usage Adaptation

Hoan Nguyen,¹ Tung Nguyen,¹
Gary Wilson Jr.,² Anh Nguyen,¹
Miryung Kim,² Tien Nguyen¹
¹Iowa State University
²The University of Texas at Austin
API Usage and Adaptation

- Library enables the reuse of existing software components and helps reduce the cost of software development and maintenance.
- APIs (Application Programming Interfaces) provide accesses to the library’s functionalities.
- When the library evolves, its APIs may change in
  - Name,
  - Parameters,
  - The order of method invocations, etc.
- The changes in APIs might also lead to the changes to their usages in the client code.
API Usage and Adaptation Example 1

<table>
<thead>
<tr>
<th>OpenNMS 1.6.10</th>
<th>OpenNMS 1.7.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ public SnmpPeer(InetAddress);</td>
<td>+ public SnmpPeer(InetAddress); [@Deprecated]</td>
</tr>
<tr>
<td>+ void setPort(int);</td>
<td>+ void setPort(int); [@Deprecated]</td>
</tr>
<tr>
<td>+ void setServerPort(int);</td>
<td>+ void setServerPort(int); [@Deprecated]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JBoss 3.2.5</th>
<th>JBoss 3.2.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>SnmpPeer peer = new SnmpPeer(this.address); peer.setPort(this.port); peer.setServerPort(this.localPort);</td>
<td>SnmpPeer peer = new SnmpPeer(this.address, this.port, this.localAddress, this.localPort);</td>
</tr>
</tbody>
</table>
### API Usage and Adaptation Example 2

<table>
<thead>
<tr>
<th>DefaultTableXYDataset in JFreeChart 0.9.15</th>
<th>DefaultTableXYDataset in JFreeChart 0.9.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ public DefaultTableXYDataset(YSeries set);</td>
<td>+ public DefaultTableXYDataset(YSeries set);</td>
</tr>
<tr>
<td>+ public void addSeries(YSeries set);</td>
<td>+ public void addSeries(YSeries set);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XYSeries in JFreeChart 0.9.15</th>
<th>XYSeries in JFreeChart 0.9.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ public XYSeries(String name, boolean allowDuplicateXValues);</td>
<td>+ public XYSeries(String name, boolean allowDuplicateXValues);</td>
</tr>
<tr>
<td></td>
<td>+ public XYSeries(String name, boolean autoSort, boolean allowDuplicateXValues);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class ManageSnapshotServlet in JBoss 3.2.7</th>
<th>Class ManageSnapshotServlet in JBoss 3.2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYSeries set = new XYSeries(attribute, false); for (int i = 0; i &lt; data.size(); i++) set.add(new Integer(i), (Number)data.get(i)); DefaultTableXYDataset dataset = new DefaultTableXYDataset(set); JFreeChart chart = ChartFactory.createXYLineChart(…, dataset, …);</td>
<td>XYSeries set = new XYSeries(attribute, false, false); for (int i = 0; i &lt; data.size(); i++) set.add(new Integer(i), (Number)data.get(i)); DefaultTableXYDataset dataset = new DefaultTableXYDataset(false); dataset.addSeries(set); JFreeChart chart = ChartFactory.createXYLineChart(…, dataset, …);</td>
</tr>
</tbody>
</table>
API Usage and Adaptation Example 3

Apache Axis APIs

```java
package org.apache.axis.providers.java;

class EJBProvider {
    ...

    protected Object makeNewServiceObject(...)
    ...
}
```

JBoss

```java
package org.jboss.net.axis.server;

class EJBProvider extends org.apache.axis.providers.java.EJBProvider {
    ...

    protected Object makeNewServiceObject(...)
    ...
}
```
API Usage and Adaptation Example 4

Apache Axis APIs

```java
package org.apache.axis.encoding;
class Serializer {
    ...
    public abstract boolean writeSchema(Types t);
    ...
}
```

JBoss

```java
package org.jboss.net.jmx.adaptor;
class AttributeSerializer extends Serializer {
    ...
    public boolean writeSchema(Types types) {
    ...
}
class ObjectNameSerializer extends Serializer {
    ...
    public boolean writeSchema(Types types) {
    ...
} 
```
API Usage and Adaptation Example 4

Apache Axis APIs

```java
package org.apache.axis.encoding;
class Serializer{
    ...
    public abstract boolean writeSchema(Class c, Types t);
    ...
}
```

JBoss

```java
package org.jboss.net.jmx.adaptor;
class AttributeSerializer extends Serializer {
    ...
    public boolean writeSchema(Types types)...
    ...
}
class ObjectNameSerializer extends Serializer {
    ...
    public boolean writeSchema(Types types)...
    ...
}
```
API Usage and Adaptation Example 4

Apache Axis APIs

```java
package org.apache.axis.encoding;

class Serializer {
    ...
    public abstract boolean writeSchema(Class c, Types t);
    ...
}
```

JBoss

```java
package org.jboss.net.jmx.adaptor;

class AttributeSerializer extends Serializer {
    ...
    public boolean writeSchema(Class clazz, Types types) {
    ...
}

class ObjectNameSerializer extends Serializer {
    ...
    public boolean writeSchema(Class clazz, Types types) {
    ...
}
API Usages and Adaptation in Object-Orient Program

- There are two ways of using libraries’ functionalities
  - Method invocations
  - Inheritance

- API usages in client code must follow certain specifications from libraries
  - Control and data dependencies among API calls
  - Interactions between multiple objects
  - Constraints on inheritance

- An adaptation tool should take the specifications of both ways of usages on APIs into consideration
Graph-based Approach for API Adaptation

Graph-based Approach for API Adaptation

Library $L$, Adapted Client $C$, New Client $N$, Client with Adaptation $N'$

$\Delta L$: Origin Analysis

$\{a, a'\}$

$\Delta C$: Origin Analysis

$\{(m, m')\}$

CUE: Usage Extraction

$\{(u, u')\}$

SAM: Adaptation Miner

$\{\Delta u\}$

LIBSYNC

Adaptation Patterns

Adaptation Recommendation
Graph-based Representation of API Usage

- i-Usage graph
  - capture the API usages through their invocations and data access
- x-Usage graph
  - capture the API usages through inheritance
i-Usage Graph

- Directed, labeled, acyclic graph:
  - Action node: method invocation
  - Data node: variable
  - Control node: branching point of a control structure
  - Edge: control and data dependency between two nodes
  - Label: method name, data type or type of control structure

- Is built by traversing the AST via control and data dependencies keeping only nodes related to the APIs
XYSeries set = new XYSeries(attribute, false, false);
for (int i = 0; i < data.size(); i++)
  set.add(new Integer(i), (Number)data.get(i));
DefaultTableXYDataset dataset =
  new DefaultTableXYDataset(false);
x-Usage Graph

- Directed, labeled, acyclic graph:
  - Node: class/interface or method
  - Edge: inheritance relation
    - o-edge: overriding relation
    - i-edge: inheritance relation
  - Label: fully qualified name (and signature)
x-Usage Graph

package

BasicProvider

java

BSFProvider

ComProvider

JavaProvider

MsgProvider

EJBProvider

RPCProvider

method

<init>(())

getNewServiceObject((Context,String))

getStrOption((String,Handler))

getEJBHome((Context,String))

inherits

overrides

org.jboss.net.axis.server

EJBProvider

<init>(())

getNewServiceObject((Context,String))

getStrOption((String,Handler))

getEJBHome((Context,String))

generateWSDL((Context))

client-added
Tree-based Origin Analysis

- Represent a program $P$ as a tree $T(P)$
  - Node: a program entity such as a package, a class/interface or a method
  - Edge: containing relation
Tree-based Origin Analysis

- Map the corresponding entities between two versions
- Derive the change (if any) of program entities
Tree-based Origin Analysis

- **Mapping criteria:**
  - Names,
  - Other attributes: super class/interface(s) for a class or parameter list, return type for a method,
  - Contents.

- **Mapping strategy:** avoid comparisons of all pairs of entities by using a top-down approach
  - Packages are mapped first, then classes and methods,
  - Entities with the mapped containing entities are compared first.
API Adaptation Pattern Mining

- Given a set of client programs adapted for the library of interest
- Use OAT to detect the change set of library’s APIs \( \Delta L \)
- Use OAT to map all the clients’ methods of two versions
- Recover usage change for each pair of mapped methods
- Keep the usage changes containing APIs in \( \Delta L \)
- Mine the frequent sub-sets of change operations
Usage Change Recovery via Graph Differencing

- Given a pair of mapped methods M and M’
- Build their corresponding i-Usages U and U’
- Align nodes between two usage graphs using maximum weighted matching
  - matching criteria: node’s label and neighboring structure
- Derive the usage change as a set of graph edit operations on nodes: delete, add and update/replace
  - aligned nodes with changed attribute are considered as updated
  - un-aligned nodes are considered as deleted or added
Usage Change Recovery

```java
JFreeChart jfreeChart = ChartFactory.createAreaChart(...);
this.configureChart(jfreeChart);
```

```java
ChartFactory.setChartTheme(StandardChartTheme.createLegacyTheme());
JFreeChart jfreeChart = ChartFactory.createAreaChart(...);
this.configureChart(jfreeChart);
```

Usage graph U

Usage graph U’
Usage Change Recovery

Usage graph U

Usage graph U’

• Usage change operations:
  • add ChartFactory.setChartTheme
  • add StandardChartTheme.createLegacyTheme
Usage Adaptation Mining

- Usage adaptation: set of usage change operations
- Usage adaptation pattern: a frequent usage adaptation, which is a frequent sub-set of usage change operations
- Relative frequency of a set of change operations $\Delta$:
  \[
  RF(\Delta) = \frac{\text{Freq}(\Delta)}{\text{NUsage}(\Delta)}
  \]
  - $\Delta$: (sub)set of operations to change usage $U$ to $U'$
  - Freq($\Delta$): number of pairs ($U$, $U'$) containing $\Delta$
  - NUsage($\Delta$): number of usages $U$ containing the reference model $U_0$ of $\Delta$
Reference Model of i-Usage Change

- Reference model captures both the usage change and its context by including nodes surrounding the change

Usage graph \( U \)

Reference model \( U_0 \)

Usage graph \( U' \)

Reference model \( U'_0 \)
API Adaptation Recommendation

- Adaptation scenarios
  - Mine from already-adapted locations of the same snapshot to recommend to other locations
  - Mine from already-adapted branches of the same system to recommend to other branches
  - Mine from already-adapted systems to recommend to other systems
API Adaptation Recommendation

- Location recommendation
  - Given a client program and two versions of its library
  - Use OAT to detect the change set of the library’s APIs $\Delta L$
  - Locations for $x$-Usage recommendations are methods that override any changed API method in $\Delta L$
  - Locations for $i$-Usage recommendations are methods that contain an invocation to any method
    - in change set of APIs in $\Delta L$
    - overrides any changed API method in $\Delta L$
    - inherits a changed API method in $\Delta L$
API Adaptation Recommendation

- **Operation recommendation**
  - Given the set of change patterns mined from already-adapted code \( F = \{ (\Delta, U_0, U'_0) \} \) and a usage \( V \) to be adapted
  - Find the reference model \( U_0^* \) best matched with \( V \)
    - \( \text{sim}(U_0, V) = \frac{\text{number of aligned nodes between } U_0 \text{ and } V}{\text{size of } U_0} \)
    - \( U_0^* = \arg\max \{ \text{sim}(U_0, V) \} \)
  - Recommend the corresponding \( \Delta^* \) as the adaptation operations to \( V \)
Example of Recommendation for x-Usage

Change in Apache Axis APIs

```java
class EJBProvider {
    ...  
    makeNewServiceObject
    ...  
}
```

Adaptation in JBoss

```java
class EJBProvider extends org.apache.axis.providers.java.EJBProvider {
    ...  
    makeNewServiceObject
    ...  
}
```
Example of Recommendation for i-Usage

**Replace** `XYSeries.<init>(String, boolean)` with `XYSeries.<init>(String, boolean, boolean)`

**Replace** `DefaultTableXYDataset.<init>(XYSeries)` with `DefaultTableXYDataset.<init>(boolean)`

**Add** `DefaultTableXYDataset.addSeries(XYSeries)`

---

Class `ManageSnapshotServlet` in JBoss 3.2.7

```
XYSeries set = new XYSeries(attribute, false);
for (int i = 0; i < data.size(); i++)
    set.add(new Integer(i), (Number)data.get(i));
DefaultTableXYDataset dataset = new DefaultTableXYDataset(set);
JFreeChart chart = ChartFactory.createXYLineChart(..., dataset, ...);
```

Class `ManageSnapshotServlet` in JBoss 3.2.8

```
XYSeries set = new XYSeries(attribute, false, false);
for (int i = 0; i < data.size(); i++)
    set.add(new Integer(i), (Number)data.get(i));
DefaultTableXYDataset dataset = new DefaultTableXYDataset(false);
dataset.addSeries(set);
JFreeChart chart = ChartFactory.createXYLineChart(..., dataset, ...);
```
Example of Recommendation for i-Usage

Reference model $U_0$

Replace

```java
XYSeries.<init>(String, boolean)
```  
```
XYSeries.<init>(String, boolean, boolean)
```  
```
DefaultTableXYDataset.<init>(XYSeries)
```  
```
DefaultTableXYDataset.<init>(boolean)
```  
```
ChartFactory.createXYLineChart(…)
```  
```
ChartFactory.createXYLineChart(…)
```  

Add

```java
XYSeries.add(…)
```  
```
XYSeries.add(…)
```  
```
DefaultTableXYDataset.addSeries(XYSeries)
```  
```
DefaultTableXYDataset.addSeries(XYSeries)
```  

Reference model $U'_0$

<table>
<thead>
<tr>
<th>Class ManageSnapshotServlet in JBoss 3.2.7</th>
<th>Class ManageSnapshotServlet in JBoss 3.2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYSeries set = new XYSeries(attribute, false);</td>
<td>XYSeries set = new XYSeries(attribute, false, false);</td>
</tr>
<tr>
<td>for (int i = 0; i &lt; data.size(); i++)</td>
<td>for (int i = 0; i &lt; data.size(); i++)</td>
</tr>
<tr>
<td>set.add(new Integer(i), (Number)data.get(i));</td>
<td>set.add(new Integer(i), (Number)data.get(i));</td>
</tr>
<tr>
<td>DefaultTableXYDataset dataset = new DefaultTableXYDataset(set);</td>
<td>DefaultTableXYDataset dataset = new DefaultTableXYDataset(false);</td>
</tr>
<tr>
<td>JFreeChart chart = ChartFactory.createXYLineChart(…, dataset, …);</td>
<td>JFreeChart chart = ChartFactory.createXYLineChart(…, dataset, …);</td>
</tr>
</tbody>
</table>
Evaluation

- Accuracy of i-Usage operation recommendation
- Accuracy of x-Usage adaptation recommendation

<table>
<thead>
<tr>
<th>Client</th>
<th>Life Cycle</th>
<th>Releases</th>
<th>Methods</th>
<th>Used libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring</td>
<td>2/2005 - 06/2008</td>
<td>29</td>
<td>10K – 18K</td>
<td>45 – 262</td>
</tr>
</tbody>
</table>

Subject systems
Accuracy of i-Usage Adaptation Recommendation

- Mine adaptation patterns from one branch of JBoss
- Adapt to versions in another branch of the same system
- An adaptation to a usage at version $v$ is considered correct if the usage was actually changed in the same way as recommended at some version later than $v$

<table>
<thead>
<tr>
<th>Mine on</th>
<th>Adapt to</th>
<th>Usages</th>
<th>Recommend</th>
<th>Correct</th>
<th>Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.5 – 3.2.8</td>
<td>3.2.5-4.0.5</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4.0.5 – 4.2.3</td>
<td>4.0.5-5.0.1</td>
<td>26</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
</tbody>
</table>
Accuracy of x-Usage Adaptation Recommendation

- On the wide range of all versions of JBoss

<table>
<thead>
<tr>
<th>Type of change</th>
<th>Recommend</th>
<th>Correct</th>
<th>Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Class name</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Package name</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Deprecated</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Change parameter type</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Del parameter</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Change return type</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Change exception</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Add parameter-Change Exception</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Add parameter-Change Return type</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Conclusions

- A graph-based approach to API adaptation
  - Capturing the contexts of API usages
  - Recovering usage adaptation patterns
  - Adapting the complex usages of APIs

- Future work
  - Large scale study on the co-evolution between APIs and client code