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 Nguyen[1]
[1]Iowa State University
[2]The 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

OpenNMS 1.6.10	OpenNMS 1.7.10

+ public SnmpPeer(InetAddress);
+ void setPort(int);
+ void setServerPort(int);
	+ public SnmpPeer(InetAddress); @**Deprecated**
	+ void setPort(int); @**Deprecated**
	+ void setServerPort(int); @**Deprecated**
	+ public SnmpPeer(InetAddress, int, InetAddress, int);

JBoss 3.2.5	JBoss 3.2.6
SnmpPeer peer = new SnmpPeer(this.address); ~~peer.setPort(this.port);~~ ~~peer.setServerPort(this.localPort);~~	SnmpPeer peer = new SnmpPeer(this.address, this.port, this.localAddress, this.localPort);

API Usage and Adaptation Example 2

DefaultTableXYDataset in JFreeChart 0.9.15	DefaultTableXYDataset in JFreeChart 0.9.17

+ public DefaultTableXYDataset(XYSeries set);
 	
+ public void addSeries(XYSeries set);
	+ public DefaultTableXYDataset(XYSeries set); **@Deprecated**
	+ public DefaultTableXYDataset(boolean autoPrune);
	+ public void addSeries(XYSeries set);

XYSeries in JFreeChart 0.9.15	XYSeries in JFreeChart 0.9.17
+ public XYSeries(String name, boolean allowDuplicateXValues);	+ public XYSeries(String name, boolean allowDuplicateXValues); **@Deprecated** **+ public XYSeries(String name, *boolean autoSort*,** **boolean allowDuplicateXValues);**

Class ManageSnapshotServlet in JBoss 3.2.7	Class ManageSnapshotServlet in JBoss 3.2.8

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, …);
	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, …);

API Usage and Adaptation Example 3

Apache Axis APIs

package org.apache.axis.providers.java;
class EJBProvider {
    …   makeNewServiceObject
    protected Object ~~getNewServiceObject~~ (…)
    … }

JBoss

package org.jboss.net.axis.server;
class EJBProvider extends org.apache.axis.providers.java.EJBProvider {
    …   makeNewServiceObject
    protected Object ~~getNewServiceObject~~ (…)
    … }

API Usage and Adaptation Example 4

Apache Axis APIs

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

JBoss

```
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

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

JBoss

```
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

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

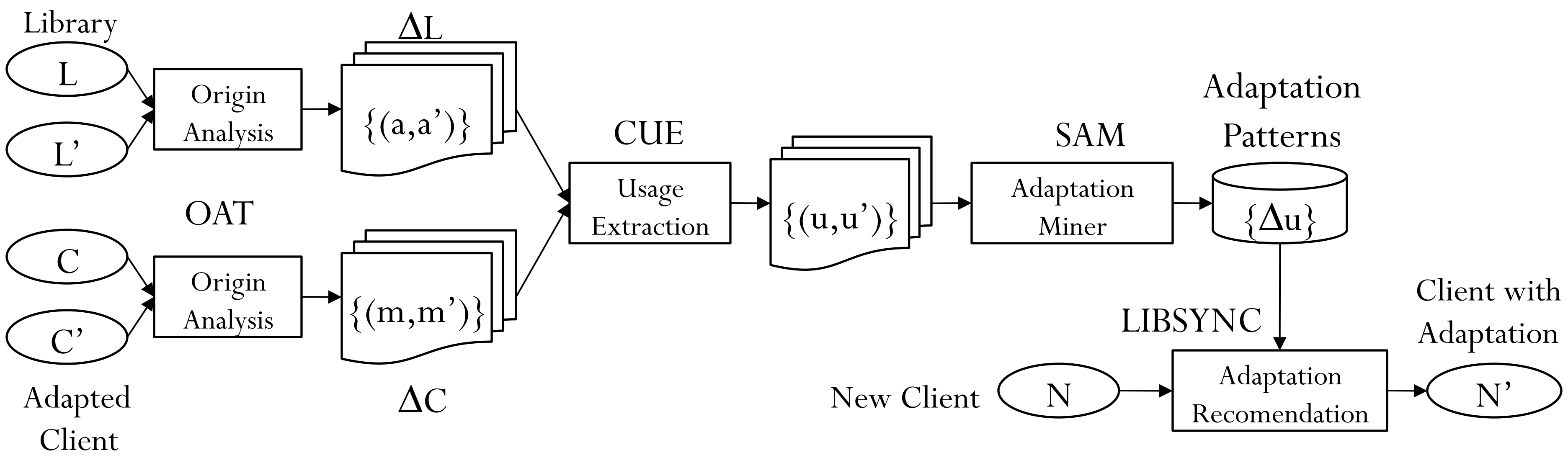
JBoss

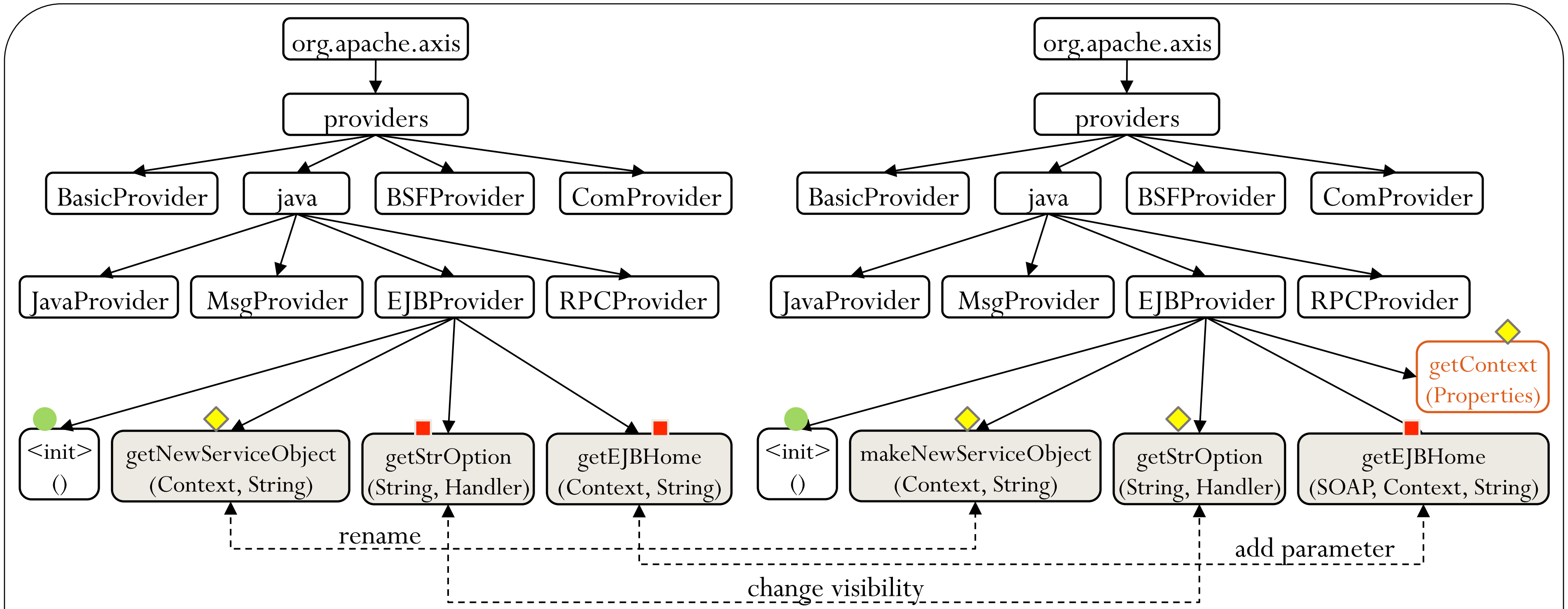
package org.jboss.net.jmx.adaptor;
class AttributeSerializer extends Serializer {
    …   Element
    public ~~boolean~~ writeSchema(Class clazz, Types types)…
    … }
class ObjectNameSerializer extends Serializer {
    …   Element
    public ~~boolean~~ writeSchema(Class clazz, Types types)…
    … }

API Usages and Adaptation in Object-Orient Program

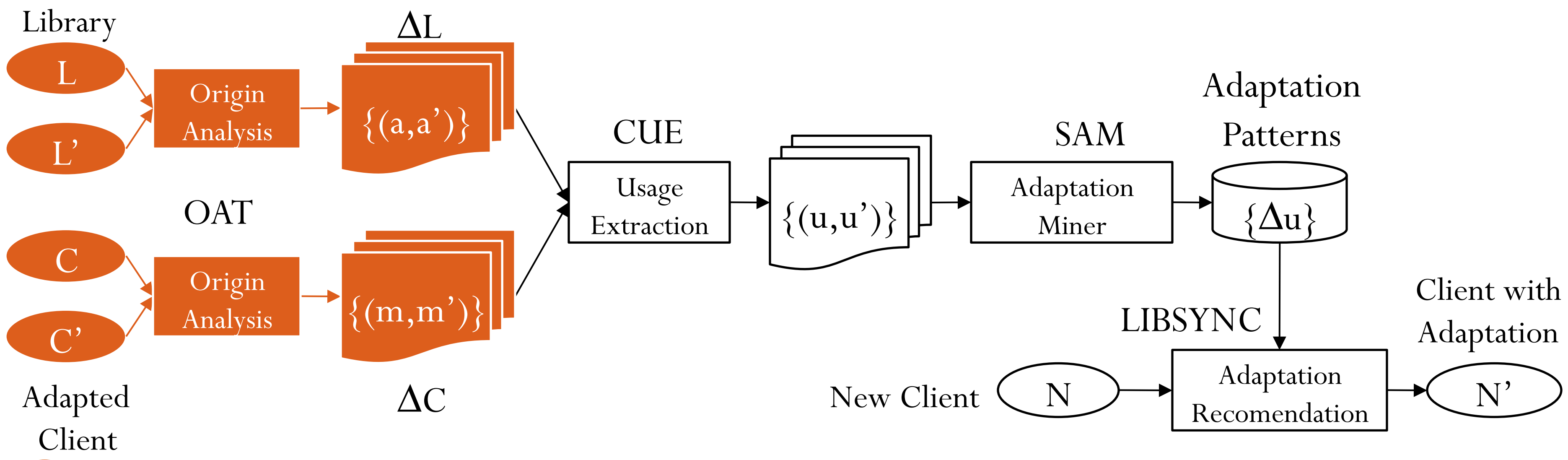
- There are two ways of using libraries' functionalities
  - Method invocations
  - Inheritance
- API usages in client code must follow certain specifications from libraries
  - Control and data dependencies among API calls
  - Interactions between multiple objects
  - Constraints on inheritance
- An adaptation tool should take the specifications of both ways of usages on APIs into consideration

Graph-based Approach for API Adaptation

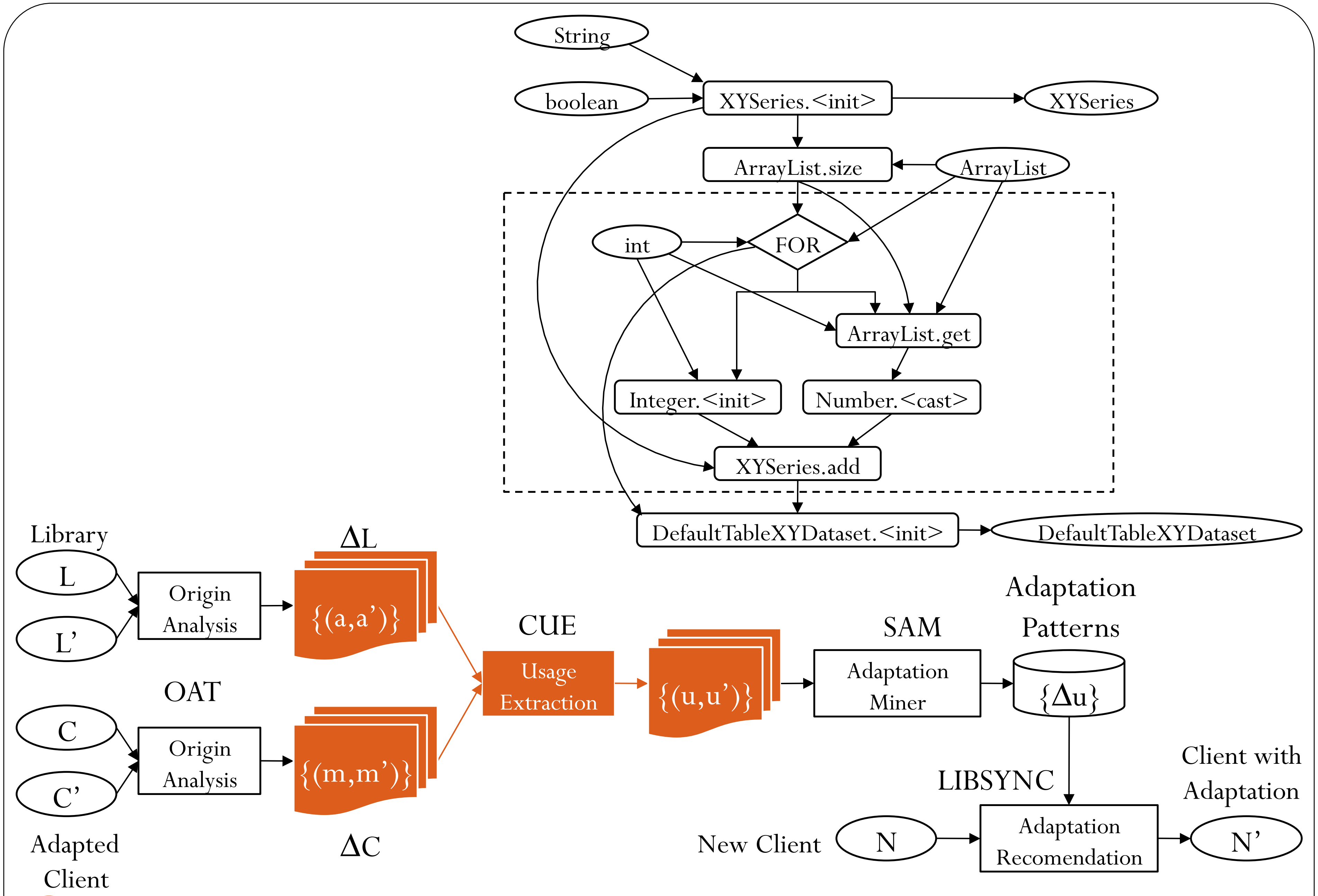




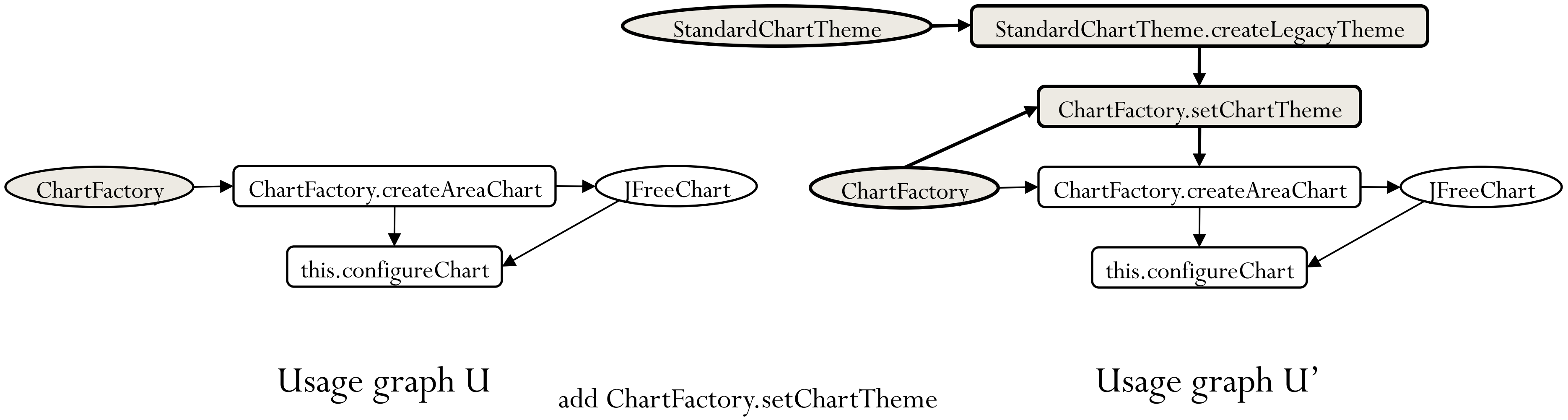
org.apache.axis
providers
BasicProvider
java
BSFProvider
ComProvider
JavaProvider
MsgProvider
EJBProvider
RPCProvider
<init> ()
getNewServiceObject (Context, String)
getStrOption (String, Handler)
getEJBHome (Context, String)
org.apache.axis
providers
BasicProvider
java
BSFProvider
ComProvider
JavaProvider
MsgProvider
EJBProvider
RPCProvider
getContext (Properties)
<init> ()
makeNewServiceObject (Context, String)
getStrOption (String, Handler)
getEJBHome (SOAP, Context, String)
rename
add parameter
change visibility



Library
L
L'
Origin Analysis
ΔL
{(a,a')}
OAT
C
C'
Origin Analysis
{(m,m')}
ΔC
Adapted Client
CUE
Usage Extraction
{(u,u')}
SAM
Adaptation Miner
Adaptation Patterns
{Δu}
LIBSYNC
New Client
N
Adaptation Recomendation
Client with Adaptation
N'

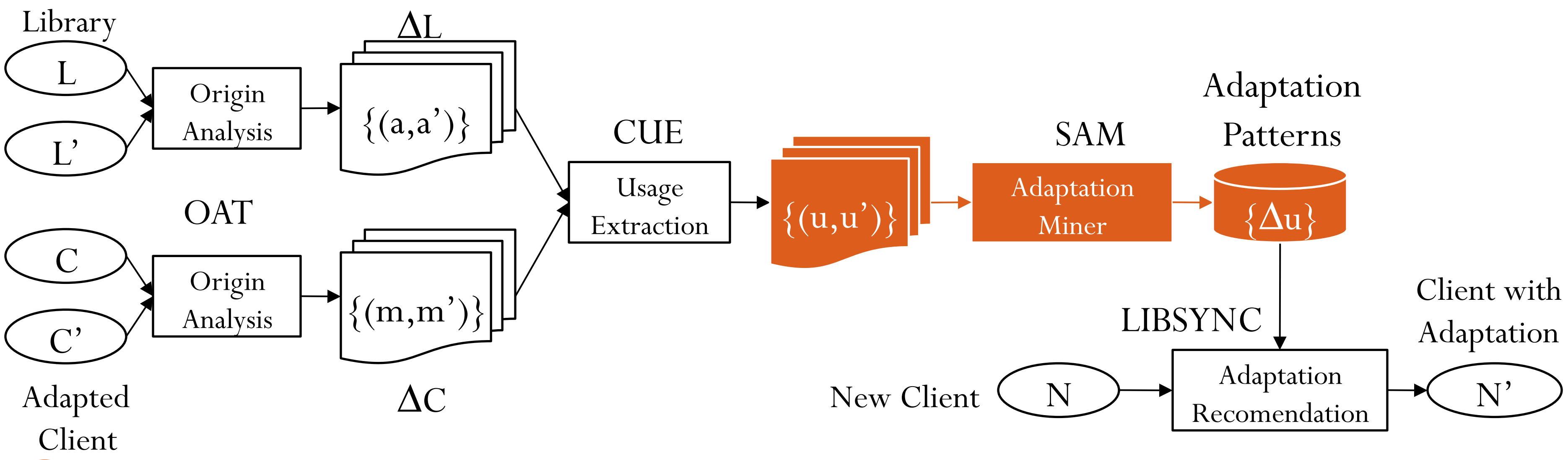


String
boolean
XYSeries.<init>
XYSeries
ArrayList.size
ArrayList
int
FOR
ArrayList.get
Integer.<init>
Number.<cast>
XYSeries.add
DefaultTableXYDataset.<init>
DefaultTableXYDataset
Library
L
L'
Origin Analysis
ΔL
{(a,a')}
OAT
C
C'
Origin Analysis
{(m,m')}
ΔC
Adapted Client
CUE
Usage Extraction
{(u,u')}
SAM
Adaptation Miner
Adaptation Patterns
{Δu}
LIBSYNC
New Client
N
Adaptation Recomendation
Client with Adaptation
N'

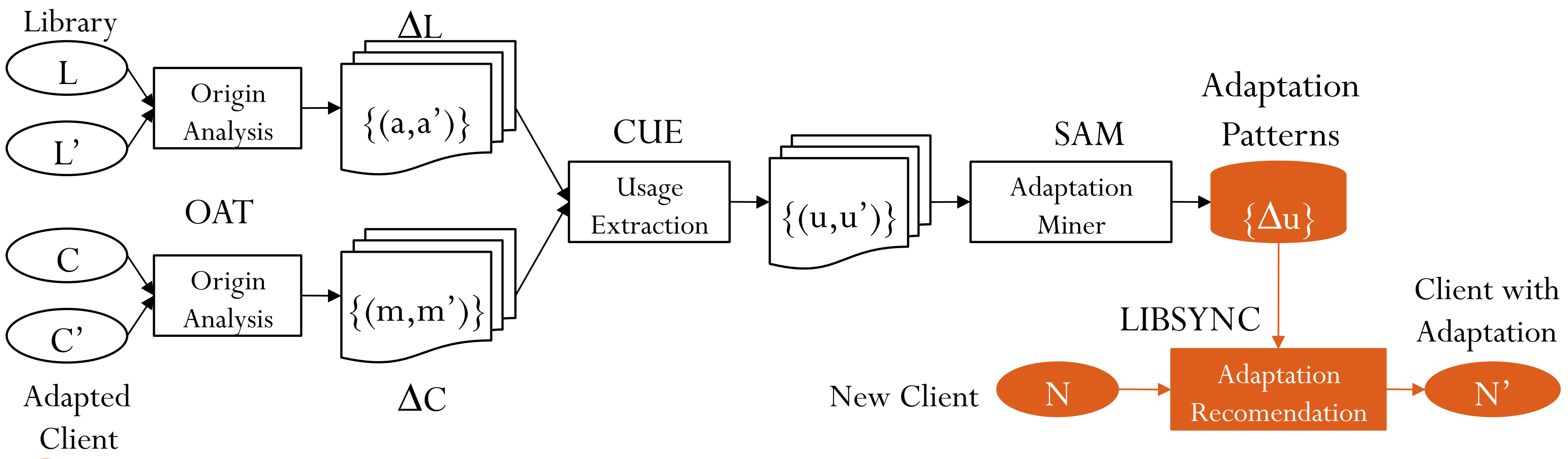


add ChartFactory.setChartTheme

add StandardChartTheme.createLegacyTheme



Graph-based Approach for API Adaptation



Graph-based Representation of API Usage

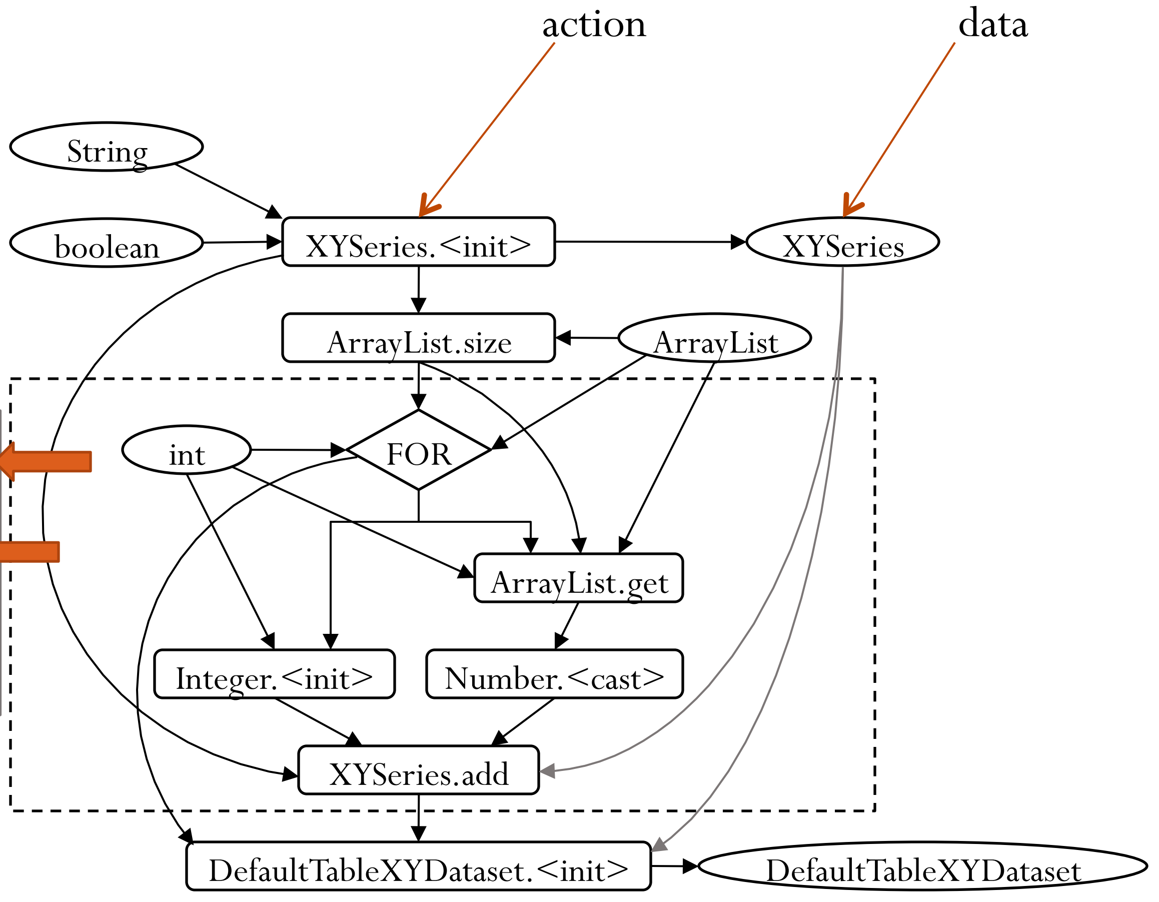
- i-Usage graph
  - capture the API usages through their invocations and data access
- x-Usage graph
  - capture the API usages through inheritance

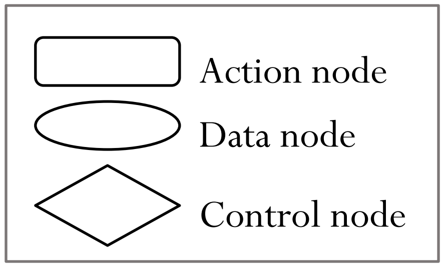
i-Usage Graph

- 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

i-Usage Graph

```
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);
```

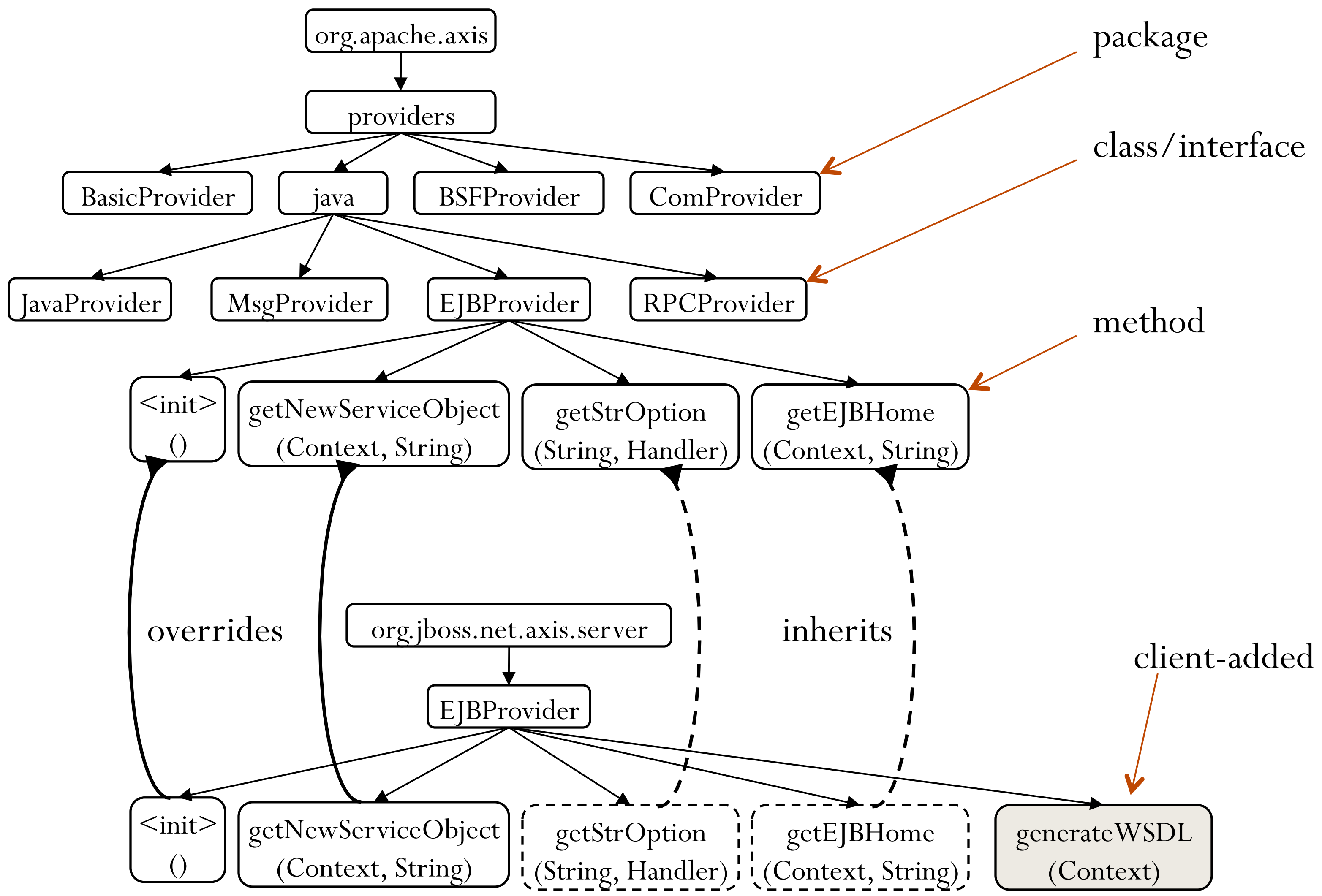




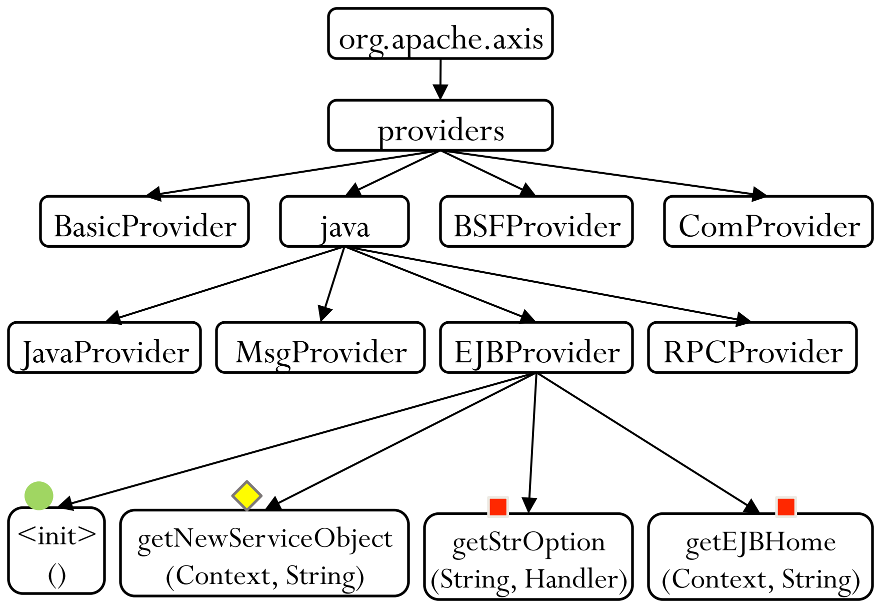
x-Usage Graph

- 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

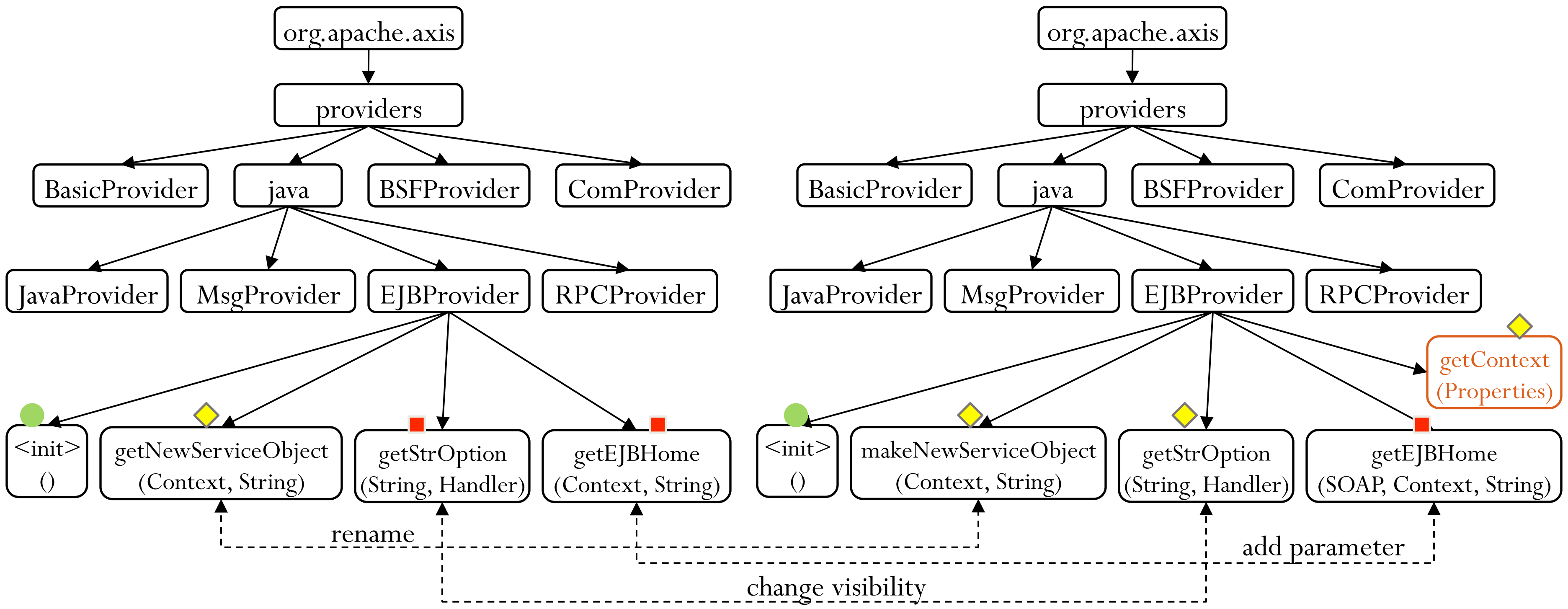


Tree-based Origin Analysis



- 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

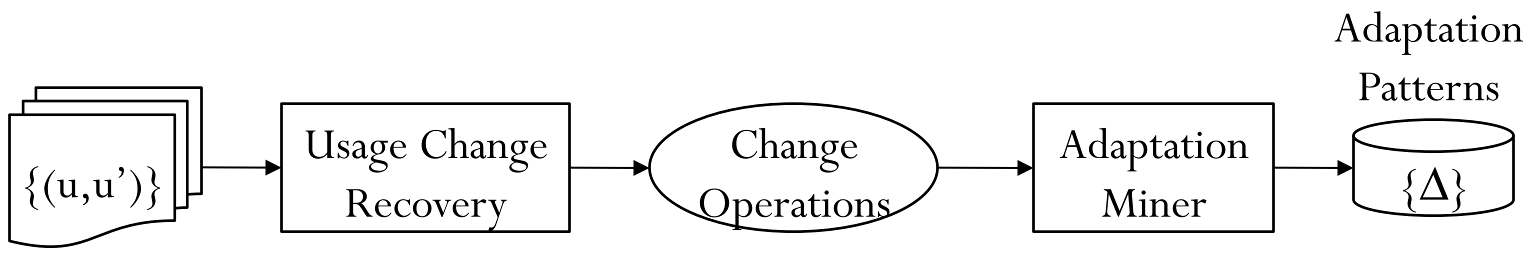


- Map the corresponding entities between two versions
- Derive the change (if any) of program entities

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 ΔL
- 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 ΔL
- Mine the frequent sub-sets of change operations

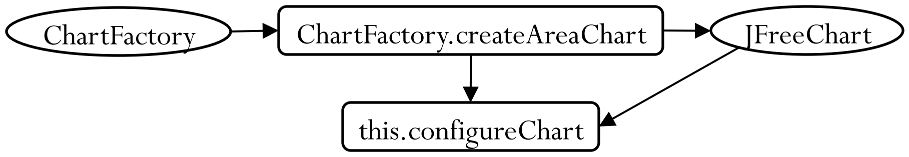
Usage Change Recovery via Graph Differencing

- 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

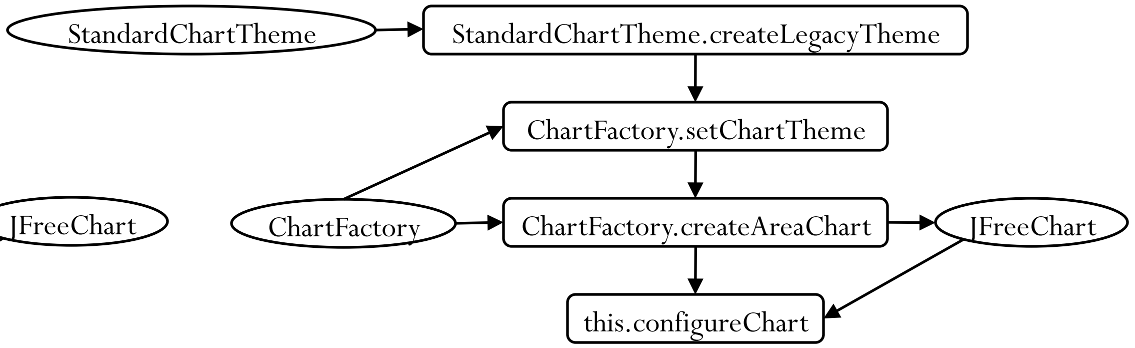
Usage Change Recovery

```
JFreeChart jfreeChart=ChartFactory.createAreaChart(…);
configureChart(jfreeChart);
```

```
ChartFactory.setChartTheme(StandardChartTheme.createLegacyTheme());
JFreeChart jfreeChart=ChartFactory.create
configureChart(jfreeChart);
```



Usage graph U

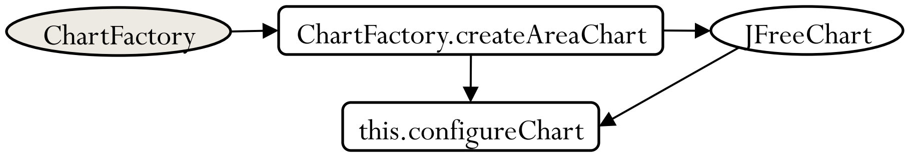

StandardChartTheme
StandardChartTheme.createLegacyTheme
ChartFactory.setChartTheme
ChartFactory
ChartFactory.createAreaChart
JFreeChart
this.configureChart

Usage graph U’

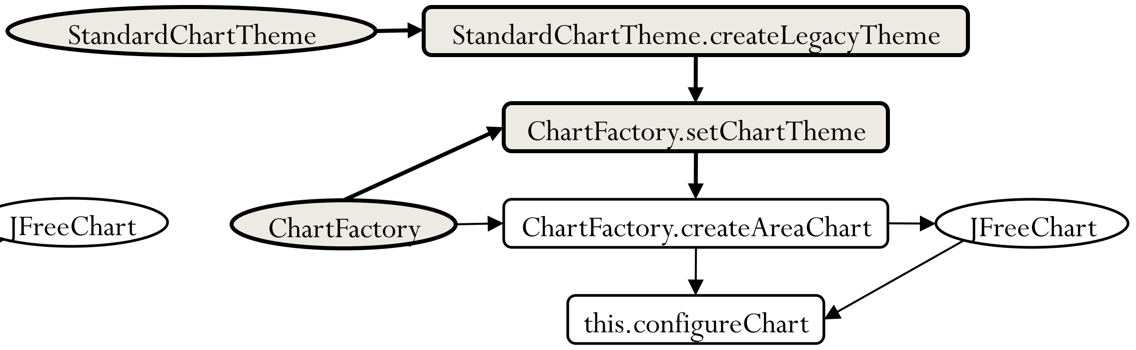
Usage Change Recovery

```
JFreeChart jfreeChart=ChartFactory.createAreaChart(…);
configureChart(jfreeChart);
```

```
ChartFactory.setChartTheme(StandardChartTheme.createLegacyTheme());
JFreeChart jfreeChart=ChartFactory.create
configureChart(jfreeChart);
```



Usage graph U


StandardChartTheme
StandardChartTheme.createLegacyTheme
ChartFactory.setChartTheme
ChartFactory
ChartFactory.createAreaChart
JFreeChart
this.configureChart

Usage graph U'

- Usage change operations:
  - add ChartFactory.setChartTheme
  - add StandardChartTheme.createLegacyTheme

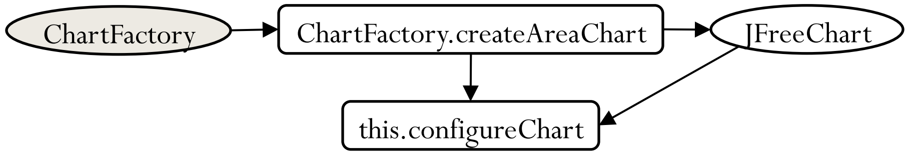
Usage Adaptation Mining

- 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 Δ:

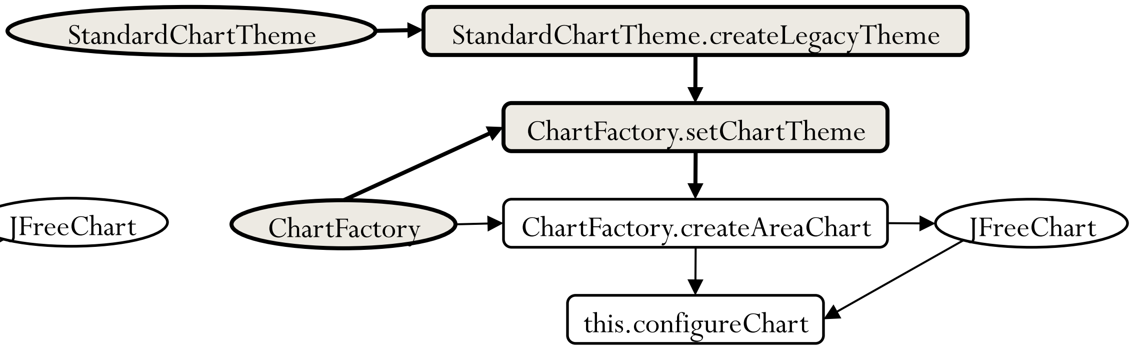
$$RF(\Delta) = Freq(\Delta) \,/\, NUsage(\Delta)$$

  - Δ : (sub)set of operations to change usage U to U'
  - Freq(Δ): number of pairs(U, U') containing Δ
  - NUsage(Δ): number of usages U containing the reference model U_0 of Δ

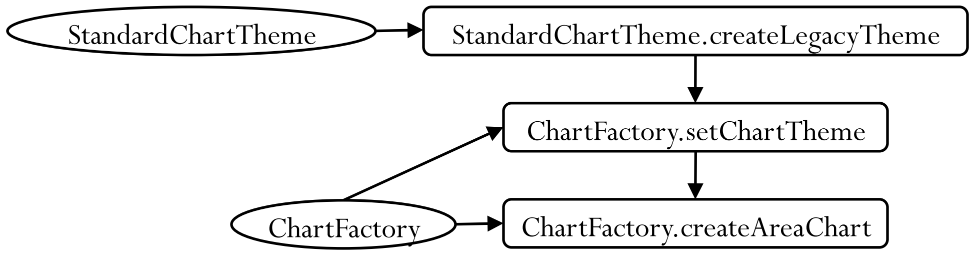
Reference Model of i-Usage Change

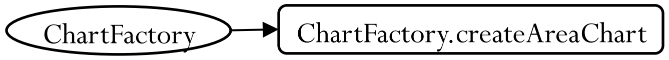


Usage graph U


StandardChartTheme
StandardChartTheme.createLegacyTheme
ChartFactory.setChartTheme
ChartFactory
ChartFactory.createAreaChart
JFreeChart
this.configureChart

Usage graph U’


StandardChartTheme
StandardChartTheme.createLegacyTheme
ChartFactory.setChartTheme
ChartFactory
ChartFactory.createAreaChart



Reference model U_0

Reference model U'_0

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

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' APIs ΔL
  - Locations for x-Usage recommendations are methods that override any changed API method in ΔL
  - Locations for i-Usage recommendations are methods that contain an invocation to any method
    - in change set of APIs in ΔL
    - overrides any changed API method in ΔL
    - inherits a changed API method in ΔL

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
    - $\mathrm{sim}(U_0, V)$ = number of aligned nodes between U_0 and V / size of U_0
    - $U_0^* = \mathrm{argmax}\{\mathrm{sim}(U_0, V)\}$
  - Recommend the corresponding Δ^* as the adaptation operations to V

Example of Recommendation for x-Usage

Change in Apache Axis APIs

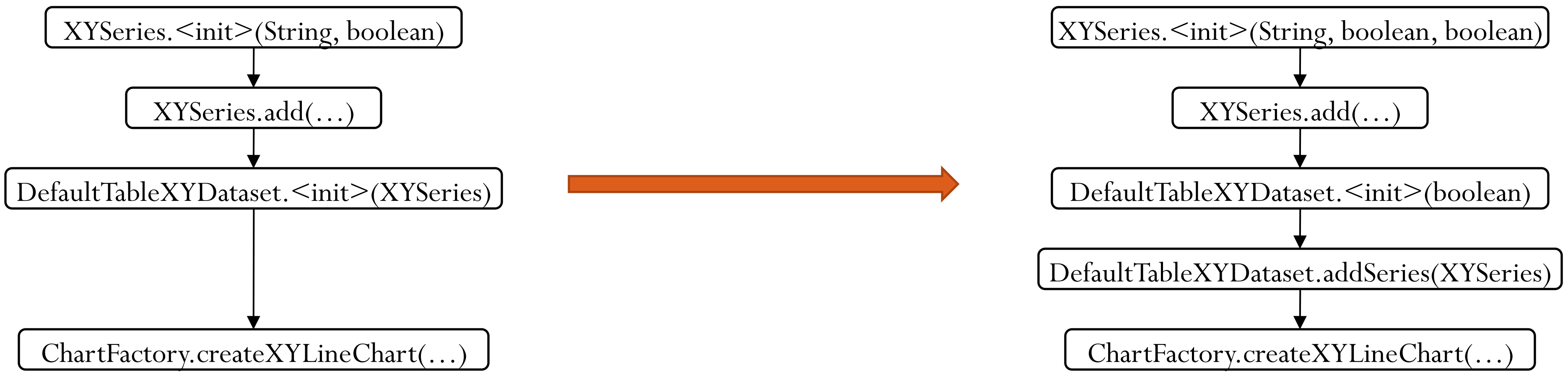
```
package org.apache.axis.providers.java;
class EJBProvider {
                            makeNewServiceObject
   …
   protected Object ~~getNewServiceObject~~(…)
   … }
```

Adaptation in JBoss

```
package org.jboss.net.axis.server;
class EJBProvider extends org.apache.axis.providers.java.EJBProvider {
                            makeNewServiceObject
   …
   protected Object ~~getNewServiceObject~~(…)
   … }
```

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)



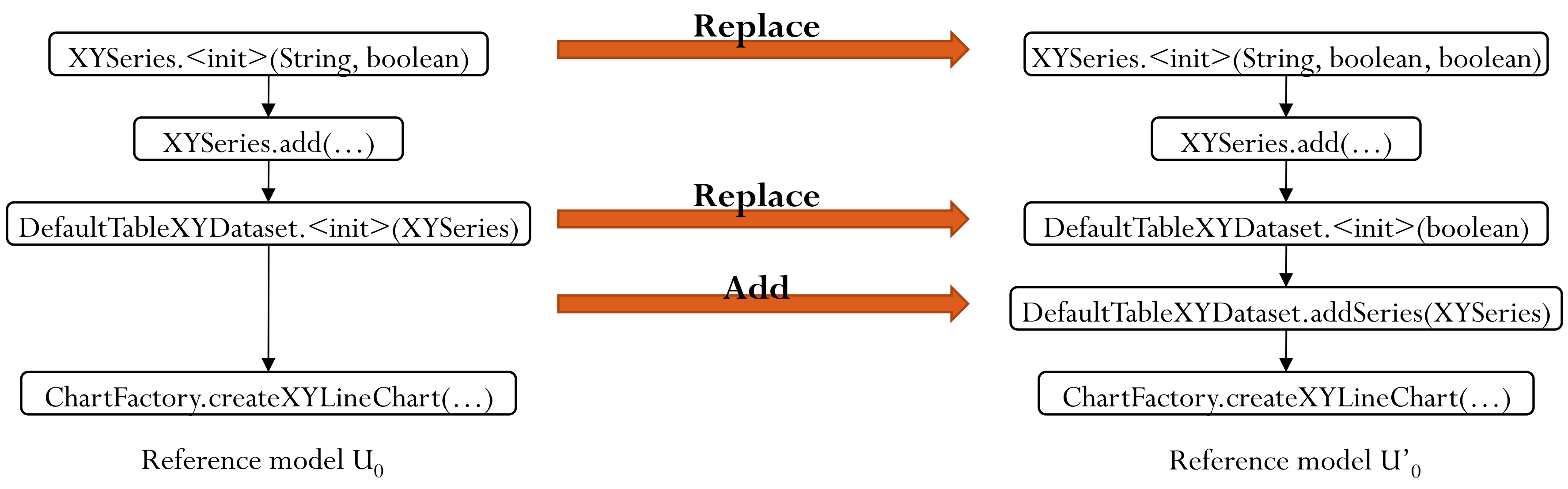
XYSeries.<init>(String, boolean)
XYSeries.add(…)
DefaultTableXYDataset.<init>(XYSeries)
ChartFactory.createXYLineChart(…)
XYSeries.<init>(String, boolean, boolean)
XYSeries.add(…)
DefaultTableXYDataset.<init>(boolean)
DefaultTableXYDataset.addSeries(XYSeries)
ChartFactory.createXYLineChart(…)

Reference model U_0 Reference model U'_0

Class ManageSnapshotServlet in JBoss 3.2.7	Class ManageSnapshotServlet in JBoss 3.2.8

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, …);
	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


XYSeries.<init>(String, boolean)
XYSeries.add(…)
DefaultTableXYDataset.<init>(XYSeries)
ChartFactory.createXYLineChart(…)
Reference model U0
Replace
Replace
Add
XYSeries.<init>(String, boolean, boolean)
XYSeries.add(…)
DefaultTableXYDataset.<init>(boolean)
DefaultTableXYDataset.addSeries(XYSeries)
ChartFactory.createXYLineChart(…)
Reference model U'0

Class ManageSnapshotServlet in JBoss 3.2.7	Class ManageSnapshotServlet in JBoss 3.2.8

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, …);
	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, …);

Evaluation

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

Client	Life Cycle	Releases	Methods	Used libraries
jBoss	10/2003 - 05/2009	47	10K – 40K	45 – 262
JasperReport	01/2004 - 02/2010	56	1K – 11K	7 – 47
Spring	2/2005 - 06/2008	29	10K – 18K	45 – 262

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*

Mine on	Adapt to	Usages	Recommend	Correct	Miss
3.2.5 – 3.2.8	3.2.5-4.0.5	6	4	4	2
4.0.5 – 4.2.3	4.0.5-5.0.1	26	25	25	1

Accuracy of x-Usage Adaptation Recommendation

- On the wide range of all versions of JBoss

Type of change	Recommend	Correct	Wrong
Name	6	4	2
Class name	1	1	0
Package name	2	2	0
Deprecated	3	3	0
Change parameter type	4	4	0
Del parameter	7	7	0
Change return type	6	6	0
Change exception	1	1	0
Add parameter-Change Exception	1	1	0
Add parameter-Change Return type	2	2	0

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